ACM 常用算法模板

kuangbin

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

Email:kuangbin2009@126.com

Last build at July 8, 2018

Contents

1 字符串处理

1.1 KMP

```
/*
 * next[] 的含义：x[i-next[i]...i-1]=x[0...next[i]-1]
 * next[i] 为满足 x[i-z...i-1]=x[0...z-1] 的最大 z 值（就是 x 的自身匹配）
 */
void kmp_pre(char x[],int m,int next[]){
    int i,j;
    j=next[0]=-1;
    i=0;
    while(i<m){
        while(-1!=j && x[i]!=x[j])j=next[j];
        next[++i]=++j;
    }
}
/*
 * kmpNext[i] 的意思:next'[i]=next[next[...[next[i]]]](直到
   next'[i]<0 或者 x[next'[i]]!=x[i])
 * 这样的预处理可以快一些
 */
void preKMP(char x[],int m,int kmpNext[]){
    int i,j;
    j=kmpNext[0]=-1;
    i=0;
    while(i<m){
        while(-1!=j && x[i]!=x[j])j=kmpNext[j];
        if(x[++i]==x[++j])kmpNext[i]=kmpNext[j];
        else kmpNext[i]=j;
    }
}
/*
 * 返回 x 在 y 中出现的次数，可以重叠
 */
int next[10010];
int KMP_Count(char x[],int m,char y[],int n){//x 是模式串，y 是主串
    int i,j;
    int ans=0;
    //preKMP(x,m,next);
    kmp_pre(x,m,next);
    i=j=0;
    while(i<n){
        while(-1!=j && y[i]!=x[j])j=next[j];
        i++;j++;
        if(j>=m){
            ans++;
            j=next[j];
        }
    }
    return ans;
```

```
}
//经典题目：POJ 3167
/*
 * POJ 3167 Cow Patterns
 * 模式串可以浮动的模式匹配问题
 * 给出模式串的相对大小，需要找出模式串匹配次数和位置
 * 比如说模式串：1, 4, 4, 2, 3, 1 而主串：5,6,2,10,10,7,3,2,9
 * 那么 2,10,10,7,3,2 就是匹配的
 *
 * 统计比当前数小，和于当前数相等的，然后进行 kmp
 */
const int MAXN=100010;
const int MAXM=25010;
int a[MAXN];
int b[MAXN];
int n,m,s;
int as[MAXN][30];
int bs[MAXM][30];
void init(){
    for(int i=0;i<n;i++){
        if(i==0){
            for(int j=1;j<=25;j++)as[i][j]=0;
        }
        else{
            for(int j=1;j<=25;j++)as[i][j]=as[i-1][j];
        }
        as[i][a[i]]++;
    }
    for(int i=0;i<m;i++){
        if(i==0){
            for(int j=1;j<=25;j++)bs[i][j]=0;
        }
        else{
            for(int j=1;j<=25;j++)bs[i][j]=bs[i-1][j];
        }
        bs[i][b[i]]++;
    }
}
int next[MAXM];
void kmp_pre(){
    int i,j;
    j=next[0]=-1;
    i=0;
    while(i<m){
        int t11=0,t12=0,t21=0,t22=0;
        for(int k=1;k<b[i];k++){
            if(i-j>0)t11+=bs[i][k]-bs[i-j-1][k];
            else t11+=bs[i][k];
        }
        if(i-j>0)t12=bs[i][b[i]]-bs[i-j-1][b[i]];
        else t12=bs[i][b[i]];

        for(int k=1;k<b[j];k++){
            t21+=bs[j][k];
        }
        t22=bs[j][b[j]];
        if(j==-1 || (t11==t21&&t12==t22)){
            next[++i]=++j;
        }
        else j=next[j];
    }
}
vector<int>ans;
void kmp(){
    ans.clear();
    int i,j;
    kmp_pre();
    i=j=0;
    while(i<n){
        int t11=0,t12=0,t21=0,t22=0;
        for(int k=1;k<a[i];k++){
            if(i-j>0)t11+=as[i][k]-as[i-j-1][k];
            else t11+=as[i][k];
        }
        if(i-j>0)t12=as[i][a[i]]-as[i-j-1][a[i]];
        else t12=as[i][a[i]];

        for(int k=1;k<b[j];k++){
            t21+=bs[j][k];
        }
        t22=bs[j][b[j]];
        if(j==-1 || (t11==t21&&t12==t22)){
            i++;j++;
            if(j>=m){
                ans.push_back(i-m+1);
                j=next[j];
            }
        }
        else j=next[j];
    }
}
int main(){
    while(scanf("%d%d%d",&n,&m,&s)==3){
        for(int i=0;i<n;i++)scanf("%d",&a[i]);
        for(int i=0;i<m;i++)scanf("%d",&b[i]);
        init();
        kmp();
        printf("%d\n",ans.size());
        for(int i=0;i<ans.size();i++)
            printf("%d\n",ans[i]);
    }
    return 0;
```

```
}
```

1.2 e-KMP

```
/*
 * 扩展 KMP 算法
 */
//next[i]:x[i...m-1] 与 x[0...m-1] 的最长公共前缀
//extend[i]:y[i...n-1] 与 x[0...m-1] 的最长公共前缀
void pre_EKMP(char x[],int m,int next[]){
    next[0] = m;
    int j = 0;
    while( j+1 < m && x[j] == x[j+1] )j++;
    next[1] = j;
    int k = 1;
    for(int i = 2; i < m; i++){
        int p = next[k]+k-1;
        int L = next[i-k];
        if( i+L < p+1 )next[i] = L;
        else{
            j = max(0,p-i+1);
            while( i+j < m && x[i+j] == x[j])j++;
            next[i] = j;
            k = i;
        }
    }
}
void EKMP(char x[],int m,char y[],int n,int next[],int extend[]){
    pre_EKMP(x,m,next);
    int j = 0;
    while(j < n && j < m && x[j] == y[j])j++;
    extend[0] = j;
    int k = 0;
    for(int i = 1;i < n;i++){
        int p = extend[k]+k-1;
        int L = next[i-k];
        if(i+L < p+1)extend[i] = L;
        else{
            j = max(0,p-i+1);
            while( i+j < n && j < m && y[i+j] == x[j] )j++;
            extend[i] = j;
            k = i;
        }
    }
}
```

1.3 Manacher

```
/*
 * 求最长回文子串
 */
const int MAXN=110010;
```

```
char Ma[MAXN*2];
int Mp[MAXN*2];
void Manacher(char s[],int len){
    int l=0;
    Ma[l++]='$';
    Ma[l++]='#';
    for(int i=0;i<len;i++){
        Ma[l++]=s[i];
        Ma[l++]='#';
    }
    Ma[l]=0;
    int mx=0,id=0;
    for(int i=0;i<l;i++){
        Mp[i]=mx>i?min(Mp[2*id-i],mx-i):1;
        while(Ma[i+Mp[i]]==Ma[i-Mp[i]])Mp[i]++;
        if(i+Mp[i]>mx){
            mx=i+Mp[i];
            id=i;
        }
    }
}
/*
 * abaaba
 * i:     0 1 2 3 4 5 6 7 8 9 10 11 12 13
 * Ma[i]: $ # a # b # a # a # b  #  a  #
 * Mp[i]: 1 1 2 1 4 1 2 7 2 1 4  1  2  1
 */
char s[MAXN];
int main(){
    while(scanf("%s",s)==1){
        int len=strlen(s);
        Manacher(s,len);
        int ans=0;
        for(int i=0;i<2*len+2;i++)
            ans=max(ans,Mp[i]-1);
        printf("%d\n",ans);
    }
    return 0;
}
```

1.4 AC 自动机

```
//======================
// HDU 2222
// 求目标串中出现了几个模式串
//====================
struct Trie{
    int next[500010][26],fail[500010],end[500010];
    int root,L;
    int newnode(){
        for(int i = 0;i < 26;i++)
            next[L][i] = -1;
        end[L++] = 0;
        return L-1;
    }
    void init(){
        L = 0;
        root = newnode();
    }
    void insert(char buf[]){
        int len = strlen(buf);
        int now = root;
        for(int i = 0;i < len;i++){
            if(next[now][buf[i]-'a'] == -1)
                next[now][buf[i]-'a'] = newnode();
            now = next[now][buf[i]-'a'];
        }
        end[now]++;
    }
    void build(){
        queue<int>Q;
        fail[root] = root;
        for(int i = 0;i < 26;i++)
            if(next[root][i] == -1)
                next[root][i] = root;
            else{
                fail[next[root][i]] = root;
                Q.push(next[root][i]);
            }
        while( !Q.empty() ){
            int now = Q.front();
            Q.pop();
            for(int i = 0;i < 26;i++)
                if(next[now][i] == -1)
                    next[now][i] = next[fail[now]][i];
                else
                {
                    fail[next[now][i]]=next[fail[now]][i];
                    Q.push(next[now][i]);
                }
        }
    }
    int query(char buf[]){
        int len = strlen(buf);
        int now = root;
        int res = 0;
        for(int i = 0;i < len;i++){
            now = next[now][buf[i]-'a'];
            int temp = now;
            while( temp != root ){
                res += end[temp];
                end[temp] = 0;
                temp = fail[temp];
            }
        }
        return res;
    }
    void debug(){
        for(int i = 0;i < L;i++){
            printf("id␣=␣%3d,fail␣=␣%3d,end␣=␣%3d,chi␣=␣[",i,fail[i],end[i]);
            for(int j = 0;j < 26;j++)
                printf("%2d",next[i][j]);
            printf("]\n");
        }
    }
};
char buf[1000010];
Trie ac;
int main(){
    int T;
    int n;
    scanf("%d",&T);
    while( T— ){
        scanf("%d",&n);
        ac.init();
        for(int i = 0;i < n;i++){
            scanf("%s",buf);
            ac.insert(buf);
        }
        ac.build();
        scanf("%s",buf);
        printf("%d\n",ac.query(buf));
    }
    return 0;
}
```

1.5 后缀数组

1.5.1 DA

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

```
*
*/
const int MAXN=20010;
int t1[MAXN],t2[MAXN],c[MAXN];//求 SA 数组需要的中间变量，不需要赋值
//待排序的字符串放在 s 数组中，从 s[0] 到 s[n-1]，长度为 n，且最大值小于 m,
//除 s[n-1] 外的所有 s[i] 都大于 0，r[n-1]=0
//函数结束以后结果放在 sa 数组中
bool cmp(int *r,int a,int b,int l){
    return r[a] == r[b] && r[a+l] == r[b+l];
}
void da(int str[],int sa[],int rank[],int height[],int n,int m){
    n++;
    int i, j, p, *x = t1, *y = t2;
    //第一轮基数排序，如果 s 的最大值很大，可改为快速排序
    for(i = 0;i < m;i++)c[i] = 0;
    for(i = 0;i < n;i++)c[x[i] = str[i]]++;
    for(i = 1;i < m;i++)c[i] += c[i-1];
    for(i = n-1;i >= 0;i--)sa[--c[x[i]]] = i;
    for(j = 1;j <= n; j <<= 1){
        p = 0;
        //直接利用 sa 数组排序第二关键字
        for(i = n-j; i < n; i++)y[p++] = i;//后面的 j 个数第二关键字为
           空的最小
        for(i = 0; i < n; i++)if(sa[i] >= j)y[p++] = sa[i] - j;
        //这样数组 y 保存的就是按照第二关键字排序的结果
        //基数排序第一关键字
        for(i = 0; i < m; i++)c[i] = 0;
        for(i = 0; i < n; i++)c[x[y[i]]]++;
        for(i = 1; i < m;i++)c[i] += c[i-1];
        for(i = n-1; i >= 0;i--)sa[--c[x[y[i]]]] = y[i];
        //根据 sa 和 x 数组计算新的 x 数组
        swap(x,y);
        p = 1; x[sa[0]] = 0;
        for(i = 1;i < n;i++)
            x[sa[i]] = cmp(y,sa[i-1],sa[i],j)?p-1:p++;
        if(p >= n)break;
        m = p;//下次基数排序的最大值
    }
    int k = 0;
    n--;
    for(i = 0;i <= n;i++)rank[sa[i]] = i;
    for(i = 0;i < n;i++){
        if(k)k--;
        j = sa[rank[i]-1];
        while(str[i+k] == str[j+k])k++;
        height[rank[i]] = k;
    }
}
int rank[MAXN],height[MAXN];
int RMQ[MAXN];
int mm[MAXN];
int best[20][MAXN];
void initRMQ(int n){
    mm[0]=-1;
    for(int i=1;i<=n;i++)
        mm[i]=((i&(i-1))==0)?mm[i-1]+1:mm[i-1];
    for(int i=1;i<=n;i++)best[0][i]=i;
    for(int i=1;i<=mm[n];i++)
        for(int j=1;j+(1<<i)-1<=n;j++){
            int a=best[i-1][j];
            int b=best[i-1][j+(1<<(i-1))];
            if(RMQ[a]<RMQ[b])best[i][j]=a;
            else best[i][j]=b;
        }
}
int askRMQ(int a,int b){
    int t;
    t=mm[b-a+1];
    b-=(1<<t)-1;
    a=best[t][a];b=best[t][b];
    return RMQ[a]<RMQ[b]?a:b;
}
int lcp(int a,int b){
    a=rank[a];b=rank[b];
    if(a>b)swap(a,b);
    return height[askRMQ(a+1,b)];
}
char str[MAXN];
int r[MAXN];
int sa[MAXN];
int main()
{
    while(scanf("%s",str) == 1){
        int len = strlen(str);
        int n = 2*len + 1;
        for(int i = 0;i < len;i++)r[i] = str[i];
        for(int i = 0;i < len;i++)r[len + 1 + i] = str[len - 1 - i
           ];
        r[len] = 1;
        r[n] = 0;
        da(r,sa,rank,height,n,128);
         for(int i=1;i<=n;i++)RMQ[i]=height[i];
        initRMQ(n);
        int ans=0,st;
        int tmp;
        for(int i=0;i<len;i++){
            tmp=lcp(i,n-i);//偶对称
            if(2*tmp>ans){
                ans=2*tmp;
                st=i-tmp;
            }
            tmp=lcp(i,n-i-1);//奇数对称
            if(2*tmp-1>ans){
                ans=2*tmp-1;
                st=i-tmp+1;
            }
        }
        str[st+ans]=0;
        printf("%s\n",str+st);
    }
    return 0;
}
```

1.5.2 DC3

da[] 和 str[] 数组要开大三倍，相关数组也是三倍

```
/*
 * 后缀数组
 * DC3 算法，复杂度 O(n)
 * 所有的相关数组都要开三倍
 */
const int MAXN = 2010;
#define F(x) ((x)/3+((x)%3==1?0:tb))
#define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)
int wa[MAXN*3],wb[MAXN*3],wv[MAXN*3],wss[MAXN*3];
int c0(int *r,int a,int b){
    return r[a] == r[b] && r[a+1] == r[b+1] && r[a+2] == r[b+2];
}
int c12(int k,int *r,int a,int b){
    if(k == 2)
        return r[a] < r[b] || ( r[a] == r[b] && c12(1,r,a+1,b+1) );
    else return r[a] < r[b] || ( r[a] == r[b] && wv[a+1] < wv[b+1]
       );
}
void sort(int *r,int *a,int *b,int n,int m){
    int i;
    for(i = 0;i < n;i++)wv[i] = r[a[i]];
    for(i = 0;i < m;i++)wss[i] = 0;
    for(i = 0;i < n;i++)wss[wv[i]]++;
    for(i = 1;i < m;i++)wss[i] += wss[i-1];
    for(i = n-1;i >= 0;i--)
        b[--wss[wv[i]]] = a[i];
}
void dc3(int *r,int *sa,int n,int m){
    int i, j, *rn = r + n;
    int *san = sa + n, ta = 0, tb = (n+1)/3, tbc = 0, p;
    r[n] = r[n+1] = 0;
    for(i = 0;i < n;i++)if(i %3 != 0)wa[tbc++] = i;
    sort(r + 2, wa, wb, tbc, m);
    sort(r + 1, wb, wa, tbc, m);
    sort(r, wa, wb, tbc, m);
    for(p = 1, rn[F(wb[0])] = 0, i = 1;i < tbc;i++)
        rn[F(wb[i])] = c0(r, wb[i-1], wb[i]) ? p - 1 : p++;
    if(p < tbc)dc3(rn,san,tbc,p);
    else for(i = 0;i < tbc;i++)san[rn[i]] = i;
    for(i = 0;i < tbc;i++) if(san[i] < tb)wb[ta++] = san[i] * 3;
    if(n % 3 == 1)wb[ta++] = n - 1;
    sort(r, wb, wa, ta, m);
    for(i = 0;i < tbc;i++)wv[wb[i] = G(san[i])] = i;
    for(i = 0, j = 0, p = 0;i < ta && j < tbc;p++)
        sa[p] = c12(wb[j] % 3, r, wa[i], wb[j]) ? wa[i++] : wb[j
           ++];
    for(;i < ta;p++)sa[p] = wa[i++];
    for(;j < tbc;p++)sa[p] = wb[j++];
}
//str 和 sa 也要三倍
void da(int str[],int sa[],int rank[],int height[],int n,int m){
    for(int i = n;i < n*3;i++)
        str[i] = 0;
    dc3(str, sa, n+1, m);
    int i,j,k = 0;
    for(i = 0;i <= n;i++)rank[sa[i]] = i;
    for(i = 0;i < n; i++){
        if(k) k--;
        j = sa[rank[i]-1];
        while(str[i+k] == str[j+k]) k++;
        height[rank[i]] = k;
    }
}
```

1.6 后缀自动机

1.6.1 基本函数

```
const int CHAR = 26;
const int MAXN = 250010;
struct SAM_Node{
    SAM_Node *fa,*next[CHAR];
    int len;
    long long cnt;
    void clear(){
        fa = 0;
        memset(next,0,sizeof(next));
        cnt = 0;
    }
}pool[MAXN*2];
SAM_Node *root,*tail;
SAM_Node* newnode(int len){
    SAM_Node* cur = tail++;
    cur->clear();
    cur->len = len;
    return cur;
}
void SAM_init(){
    tail = pool;
    root = newnode(0);
}
SAM_Node* extend(SAM_Node* last,int x){
    SAM_Node *p = last, *np = newnode(p->len+1);
    while(p && !p->next[x])
        p->next[x] = np, p = p->fa;
    if(!p)np->fa = root;
    else {
        SAM_Node* q = p->next[x];
        if(q->len == p->len+1)np->fa = q;
        else {
            SAM_Node* nq = newnode(p->len+1);
            memcpy(nq->next,q->next,sizeof(q->next));
            nq->fa = q->fa;
            q->fa = np->fa = nq;
            while(p && p->next[x] == q)
                p->next[x] = nq, p = p->fa;
        }
    }
    return np;
}
```

1.6.2 例题

CC TSUBSTR
给了一个 Trie 树，求 Trie 子树上的第 k 大的子串。

```
/*
 *  http://www.codechef.com/problems/TSUBSTR/
Input:
8 4
abcbbaca
1 2
2 3
1 4
4 5
4 6
4 7
1 8
abcdefghijklmnopqrstuvwxyz 5
abcdefghijklmnopqrstuvwxyz 1
bcadefghijklmnopqrstuvwxyz 5
abcdefghijklmnopqrstuvwxyz 100

Output:
12
aba

ba
-1
 */
const int CHAR = 26;
const int MAXN = 250010;
struct SAM_Node{
    SAM_Node *fa,*next[CHAR];
    int len;
    long long cnt;
    void clear(){
        fa = 0;
        memset(next,0,sizeof(next));
        cnt = 0;
    }
}pool[MAXN*2];
SAM_Node *root,*tail;
SAM_Node* newnode(int len){
    SAM_Node* cur = tail++;
    cur->clear();
    cur->len = len;
    return cur;
}
void SAM_init(){
    tail = pool;
    root = newnode(0);
}
SAM_Node* extend(SAM_Node* last,int x){
    SAM_Node *p = last, *np = newnode(p->len+1);
    while(p && !p->next[x])
        p->next[x] = np, p = p->fa;
    if(!p)np->fa = root;
    else {
        SAM_Node* q = p->next[x];
        if(q->len == p->len+1)np->fa = q;
        else {
            SAM_Node* nq = newnode(p->len+1);
            memcpy(nq->next,q->next,sizeof(q->next));
            nq->fa = q->fa;
            q->fa = np->fa = nq;
            while(p && p->next[x] == q)
                p->next[x] = nq, p = p->fa;
        }
    }
    return np;
}
char str[MAXN];
struct Edge
{
    int to,next;
}edge[MAXN*2];
int head[MAXN],tot;
void addedge(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
```

```
}

SAM_Node *end[MAXN];
int topcnt[MAXN];// 拓扑排序使用
SAM_Node *topsam[MAXN*2];
char s2[40];
int order[40];

int main()
{
    int n,Q;
    while(scanf("%d%d",&n,&Q) == 2){
        scanf("%s",str+1);
        memset(head,-1,sizeof(head));tot = 0;
        int u,v;
        for(int i = 1;i < n;i++){
            scanf("%d%d",&u,&v);
            addedge(u,v);  addedge(v,u);
        }
        addedge(0,1);
        SAM_init();
        memset(end,0,sizeof(end));
        end[0] = root;
        queue<int>q;
        q.push(0);
        while(!q.empty()){
            u = q.front();
            q.pop();
            for(int i = head[u];i != -1;i = edge[i].next){
                v = edge[i].to;
                if(end[v] != 0)continue;
                end[v] = extend(end[u],str[v]-'a');
                q.push(v);
            }
        }
        memset(topcnt,0,sizeof(topcnt));
        int num = tail - pool;
        for(int i = 0;i < num;i++)topcnt[pool[i].len]++;
        for(int i = 1;i <= n;i++)topcnt[i] += topcnt[i-1];
        for(int i = 0;i < num;i++)topsam[--topcnt[pool[i].len]] = &
           pool[i];

        for(int i = num-1;i >= 0;i--){
            SAM_Node *p = topsam[i];
            p->cnt = 1;
            for(int i = 0;i < 26;i++)
                if(p->next[i])
                    p->cnt += p->next[i]->cnt;
        }
        printf("%lld\n",root->cnt);
        long long k;
        while(Q--){
            scanf("%s",s2);
            for(int i = 0;i < 26;i++)order[i] = s2[i]-'a';
            scanf("%lld",&k);
            if(k > root->cnt){
                printf("-1\n");
                continue;
            }
            SAM_Node *p = root;
            //这里的第 k 个子串是从空串算起的
            while( (--k) > 0 ){
                for(int i = 0;i < 26;i++)
                    if(p->next[order[i]]){
                        if(k <= p->next[order[i]]->cnt){
                            printf("%c",'a'+order[i]);
                            p = p->next[order[i]];
                            break; //这个不要忘记
                        }
                        else k -= p->next[order[i]]->cnt;
                    }
            }
            printf("\n");
        }
    }
    return 0;
}
```

CF129 E
给了 n 个字符串，求每个字符串有多少个至少出现在 k 个字符串中的子串
fail 树，两遍 dfs, 经典题。

```
/* http://codeforces.com/contest/204/problem/E
input
3 1
abc
a
ab
output
6 1 3
input
7 4
rubik
furik
abab
baba
aaabbbababa
abababababa
zero
output
1 0 9 9 21 30 0
 */
const int CHAR = 26;
const int MAXN = 100010;
//************SAM*****************
struct SAM_Node{
    SAM_Node *fa,*next[CHAR];
    int len;
    void clear(){
        fa = 0;
        memset(next,0,sizeof(next));
    }
}pool[MAXN*2];
SAM_Node *root,*tail;
SAM_Node* newnode(int len){
    SAM_Node* cur = tail++;
    cur->clear();
    cur->len = len;
    return cur;
}
void SAM_init(){
    tail = pool;
    root = newnode(0);
}
SAM_Node* extend(SAM_Node* last,int x){
    SAM_Node *p = last, *np = newnode(p->len+1);
    while(p && !p->next[x])
        p->next[x] = np, p = p->fa;
    if(!p)np->fa = root;
    else {
        SAM_Node* q = p->next[x];
        if(q->len == p->len+1)np->fa = q;
        else {
            SAM_Node* nq = newnode(p->len+1);
            memcpy(nq->next,q->next,sizeof(q->next));
            nq->fa = q->fa;
            q->fa = np->fa = nq;
            while(p && p->next[x] == q)
                p->next[x] = nq, p = p->fa;
        }
    }
    return np;
}
//***********Trie***********
struct Trie_Node{
    int next[CHAR];
    vector<int>belongs;
}trie[MAXN];
int trie_root,trie_tot;
int trie_newnode(){
    memset(trie[trie_tot].next,-1,sizeof(trie[trie_tot].next));
    trie[trie_tot].belongs.clear();
    return trie_tot++;
}
```

```
void Trie_init(){
    trie_tot = 0;
    trie_root = trie_newnode();
}
void insert(char buf[],int id){
    int now = trie_root;
    int len = strlen(buf);
    for(int i = 0;i < len;i++){
        if(trie[now].next[buf[i]-'a'] == -1)
            trie[now].next[buf[i]-'a'] = trie_newnode();
        now = trie[now].next[buf[i]-'a'];
        trie[now].belongs.push_back(id);
    }
}
//***** fail 树***************
struct Edge{
    int to,next;
}edge[MAXN*2];
int head[MAXN*2],tot;
void addedge(int u,int v){
    edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;
}
int MtoT[MAXN*2];//SAM 结点映射到 Trie 结点
int cnt[MAXN*2];
int F[MAXN*2];
int find(int x){
    if(F[x] == -1)return x;
    return F[x] = find(F[x]);
}
void bing(int u,int v)//注意方向性
{
    int t1 = find(u);
    int t2 = find(v);
    if(t1 != t2)F[t1] = t2;
}
int L[MAXN];
void Tarjan(int u){
    for(int i = head[u];i != -1;i = edge[i].next){
        Tarjan(edge[i].to);
        bing(edge[i].to,u);
    }
    if(MtoT[u]){
        int tt = MtoT[u];
        int sz = trie[tt].belongs.size();
        for(int i = 0;i < sz;i++){
            int v = trie[tt].belongs[i];
            cnt[find(L[v])]--;
            cnt[u]++;
            L[v] = u;
        }
    }
```

```
}
void dfs1(int u){
    for(int i = head[u];i != -1;i = edge[i].next){
        dfs1(edge[i].to);
        cnt[u] += cnt[edge[i].to];
    }
}
long long ans[MAXN];
void dfs2(int u){
    for(int i = head[u];i != -1;i = edge[i].next){
        int v = edge[i].to;
        cnt[v] += cnt[u];
        dfs2(v);
    }
    if(MtoT[u]){
        int tt = MtoT[u];
        int sz = trie[tt].belongs.size();
        for(int i = 0;i < sz;i++){
            int v = trie[tt].belongs[i];
            ans[v] += cnt[u];
        }
    }
}

char str[MAXN];
SAM_Node *end[MAXN];
int main()
{
    int n,k;
    while(scanf("%d%d",&n,&k) == 2){
        Trie_init();
        for(int i = 0;i < n;i++){
            scanf("%s",str);
            insert(str,i);
        }
        SAM_init();
        //根据 Trie 建立 SAM
        memset(end,0,sizeof(end));
        end[0] = root;
        memset(MtoT,0,sizeof(MtoT));
        MtoT[root-pool] = 0;
        queue<int>q;
        q.push(trie_root);
        while(!q.empty()){
            int u = q.front();
            q.pop();
            for(int i = 0;i < 26;i++){
                if(trie[u].next[i] == -1)continue;
                int v = trie[u].next[i];
                end[v] = extend(end[u],i);
                MtoT[end[v]-pool] = v;
```

```
                    q.push(v);
                }
            }
            //建立 fail 树
            int num = tail - pool;
            memset(head,-1,sizeof(head));
            tot = 0;
            for(SAM_Node *p = pool+1;p < tail;p++)
                addedge(p->fa - pool,p - pool);
            memset(cnt,0,sizeof(cnt));
            memset(F,-1,sizeof(F));
            memset(L,0,sizeof(L));
            Tarjan(0);
            dfs1(0);
            for(int i = 0;i < num;i++){
                if(cnt[i] >= k)cnt[i] = pool[i].len - pool[i].fa->len;
                else cnt[i] = 0;
            }
            memset(ans,0,sizeof(ans));
            dfs2(0);
            for(int i = 0;i < n;i++){
                printf("%I64d",ans[i]);
                if(i < n-1)printf("␣");
                else printf("\n");
            }
        }
        return 0;
}
```

1.7 字符串 hash

HDU4622 求区间不相同子串个数

```
const int HASH = 10007;
const int MAXN = 2010;
struct HASHMAP{
    int head[HASH],next[MAXN],size;
    unsigned long long state[MAXN];
    int f[MAXN];
    void init(){
        size = 0;
        memset(head,-1,sizeof(head));
    }
    int insert(unsigned long long val,int _id){
        int h = val%HASH;
        for(int i = head[h]; i != -1;i = next[i])
            if(val == state[i]){
                int tmp = f[i];
                f[i] = _id;
                return tmp;
            }
        f[size] = _id;
        state[size] = val;
        next[size] = head[h];
        head[h] = size++;
        return 0;
    }
}H;
const int SEED = 13331;
unsigned long long P[MAXN];
unsigned long long S[MAXN];
char str[MAXN];
int ans[MAXN][MAXN];
int main(){
    P[0] = 1;
    for(int i = 1;i < MAXN;i++)
        P[i] = P[i-1] * SEED;
    int T;
    scanf("%d",&T);
    while(T--){
        scanf("%s",str);
        int n = strlen(str);
        S[0] = 0;
        for(int i = 1;i <= n;i++)
            S[i] = S[i-1]*SEED + str[i-1];
        memset(ans,0,sizeof(ans));
        for(int L = 1; L <= n;L++){
            H.init();
            for(int i = 1;i + L - 1 <= n;i++){
                int l = H.insert(S[i+L-1] - S[i-1]*P[L],i);
                ans[i][i+L-1] ++;
                ans[l][i+L-1]--;
            }
        }
        for(int i = n;i >= 0;i--)
            for(int j = i;j <= n;j++)
                ans[i][j] += ans[i+1][j] + ans[i][j-1] - ans[i+1][j
                   -1];
        int m,u,v;
        scanf("%d",&m);
        while(m--){
            scanf("%d%d",&u,&v);
            printf("%d\n",ans[u][v]);
        }
    }
    return 0;
}
```

2 数学

2.1 素数

2.1.1 素数筛选（判断 <MAXN 的数是否素数）

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

2.1.2 素数筛选（筛选出小于等于 MAXN 的素数）

```
/*
 * 素数筛选，存在小于等于 MAXN 的素数
 * prime[0] 存的是素数的个数
 */
const int MAXN=10000;
int prime[MAXN+1];
void getPrime(){
    memset(prime,0,sizeof(prime));
    for(int i=2;i<=MAXN;i++){
        if(!prime[i])prime[++prime[0]]=i;
        for(int j=1;j<=prime[0]&&prime[j]<=MAXN/i;j++){
            prime[prime[j]*i]=1;
            if(i%prime[j]==0) break;
        }
    }
}
```

2.1.3 大区间素数筛选（POJ 2689）

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

```
const int MAXN=100010;
int prime[MAXN+1];
void getPrime(){
    memset(prime,0,sizeof(prime));
    for(int i=2;i<=MAXN;i++){
        if(!prime[i])prime[++prime[0]]=i;
        for(int j=1;j<=prime[0]&&prime[j]<=MAXN/i;j++){
            prime[prime[j]*i]=1;
            if(i%prime[j]==0)break;
        }
    }
}
bool notprime[1000010];
int prime2[1000010];
void getPrime2(int L,int R){
    memset(notprime,false,sizeof(notprime));
    if(L<2)L=2;
    for(int i=1;i<=prime[0]&&(long long)prime[i]*prime[i]<=R;i++){
        int s=L/prime[i]+(L%prime[i]>0);
        if(s==1)s=2;
        for(int j=s;(long long)j*prime[i]<=R;j++)
            if((long long)j*prime[i]>=L)
                notprime[j*prime[i]-L]=true;
    }
    prime2[0]=0;
    for(int i=0;i<=R-L;i++)
        if(!notprime[i])
            prime2[++prime2[0]]=i+L;
}
int main(){
    getPrime();
    int L,U;
    while(scanf("%d%d",&L,&U)==2){
        getPrime2(L,U);
        if(prime2[0]<2)printf("There␣are␣no␣adjacent␣primes.\n");
        else{
            int x1=0,x2=100000000,y1=0,y2=0;
            for(int i=1;i<prime2[0];i++){
                if(prime2[i+1]-prime2[i]<x2-x1){
                    x1=prime2[i];
                    x2=prime2[i+1];
                }
                if(prime2[i+1]-prime2[i]>y2-y1){
                    y1=prime2[i];
                    y2=prime2[i+1];
                }
            }
            printf("%d,%d␣are␣closest,␣%d,%d␣are␣most␣distant.\n",
               x1,x2,y1,y2);
        }
    }
}
```

2.2 素数筛选和合数分解

```
//******************************************
//素数筛选和合数分解
const int MAXN=10000;
int prime[MAXN+1];
void getPrime(){
    memset(prime,0,sizeof(prime));
    for(int i=2;i<=MAXN;i++){
        if(!prime[i])prime[++prime[0]]=i;
        for(int j=1;j<=prime[0]&&prime[j]<=MAXN/i;j++){
            prime[prime[j]*i]=1;
            if(i%prime[j]==0) break;
        }
    }
}
long long factor[100][2];
int fatCnt;
int getFactors(long long x){
    fatCnt=0;
    long long tmp=x;
    for(int i=1;prime[i]<=tmp/prime[i];i++){
        factor[fatCnt][1]=0;
        if(tmp%prime[i]==0){
            factor[fatCnt][0]=prime[i];
            while(tmp%prime[i]==0){
                factor[fatCnt][1]++;
                tmp/=prime[i];
            }
            fatCnt++;
        }
    }
    if(tmp!=1){
        factor[fatCnt][0]=tmp;
        factor[fatCnt++][1]=1;
    }
    return fatCnt;
}
//******************************************
```

2.3 扩展欧几里得算法（求 ax+by=gcd 的解以及逆元）

```
//*******************************
//返回 d=gcd(a,b); 和对应于等式 ax+by=d 中的 x,y
long long extend_gcd(long long a,long long b,long long &x,long long
    &y){
    if(a==0&&b==0) return -1;//无最大公约数
    if(b==0){x=1;y=0;return a;}
    long long d=extend_gcd(b,a%b,y,x);
    y-=a/b*x;
    return d;
}
//********* 求逆元 ******************
//ax = 1(mod n)
long long mod_reverse(long long a,long long n){
    long long x,y;
    long long d=extend_gcd(a,n,x,y);
    if(d==1) return (x%n+n)%n;
    else return -1;
}
```

2.4 求逆元

2.4.1 扩展欧几里德法

见上面的写法

2.4.2 简洁写法

注意：这个只能求 $a < m$ 的情况，而且必须保证 a 和 m 互质

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

2.4.3 利用欧拉函数

mod 为素数, 而且 a 和 m 互质

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

2.5 模线性方程组

```
long long extend_gcd(long long a,long long b,long long &x,long long
     &y){
    if(a == 0 && b == 0)return -1;
    if(b ==0 ){x = 1; y = 0;return a;}
    long long d = extend_gcd(b,a%b,y,x);
    y -= a/b*x;
    return d;
}
int m[10],a[10];//模数为 m, 余数为 a,X % m = a
bool solve(int &m0,int &a0,int m,int a){
    long long y,x;
    int g = extend_gcd(m0,m,x,y);
    if( abs(a - a0)%g )return false;
    x *= (a - a0)/g;
    x %= m/g;
    a0 = (x*m0 + a0);
    m0 *= m/g;
    a0 %= m0;
    if( a0 < 0 )a0 += m0;
    return true;
}
/*
 * 无解返回 false, 有解返回 true;
 * 解的形式最后为 a0 + m0 * t (0<=a0<m0)
 */
bool MLES(int &m0 ,int &a0,int n)//解为  X = a0 + m0 * k
{
    bool flag = true;
    m0 = 1;
    a0 = 0;
    for(int i = 0;i < n;i++)
        if( !solve(m0,a0,m[i],a[i]) )
    {
        flag = false;
        break;
    }
    return flag;
}
```

2.6 随机素数测试和大数分解 (POJ 1811)

```
/* ***************************************************
 * Miller_Rabin 算法进行素数测试
 * 速度快可以判断一个 < 2^63 的数是不是素数
 *
 ****************************************************/
const int S = 8; //随机算法判定次数一般 8~10 就够了
// 计算 ret = (a*b)%c     a,b,c < 2^63
long long mult_mod(long long a,long long b,long long c){
    a %= c;
    b %= c;
    long long ret = 0;
    long long tmp = a;
    while(b){
        if(b & 1){
            ret += tmp;
            if(ret > c)ret -= c;//直接取模慢很多
        }
        tmp <<= 1;
        if(tmp > c)tmp -= c;
        b >>= 1;
    }
    return ret;
}
// 计算 ret = (a^n)%mod
long long pow_mod(long long a,long long n,long long mod){
    long long ret = 1;
    long long temp = a%mod;
    while(n){
        if(n & 1)ret = mult_mod(ret,temp,mod);
        temp = mult_mod(temp,temp,mod);
        n >>= 1;
    }
    return ret;
}
// 通过 a^(n-1)=1(mod n)来判断 n 是不是素数
// n-1 = x*2^t 中间使用二次判断
// 是合数返回 true, 不一定是合数返回 false
bool check(long long a,long long n,long long x,long long t){
    long long ret = pow_mod(a,x,n);
    long long last = ret;
    for(int i = 1;i <= t;i++){
        ret = mult_mod(ret,ret,n);
        if(ret == 1 && last != 1 && last != n-1)return true;//合数
        last = ret;
    }
    if(ret != 1)return true;
    else return false;
}
//**************************************************
// Miller_Rabin 算法
// 是素数返回 true,(可能是伪素数)
// 不是素数返回 false
//**************************************************
bool Miller_Rabin(long long n){
    if( n < 2)return false;
    if( n == 2)return true;
    if( (n&1) == 0)return false;//偶数
    long long x = n - 1;
    long long t = 0;
    while( (x&1)==0 ){x >>= 1; t++;}

    srand(time(NULL));/* *************** */

    for(int i = 0;i < S;i++){
        long long a  = rand()%(n-1) + 1;
        if( check(a,n,x,t) )
            return false;
    }
    return true;
}

//**********************************************
// pollard_rho 算法进行质因素分解
//*********************************************
long long factor[100];//质因素分解结果（刚返回时时无序的）
int tol;//质因素的个数, 编号 0~tol-1

```

```
long long gcd(long long a,long long b){
    long long t;
    while(b){
        t = a;
        a = b;
        b = t%b;
    }
    if(a >= 0)return a;
    else return -a;
}

 //找出一个因子
long long pollard_rho(long long x,long long c){
    long long i = 1, k = 2;
    srand(time(NULL));
    long long x0 = rand()%(x-1) + 1;
    long long y = x0;
    while(1){
        i ++;
        x0 = (mult_mod(x0,x0,x) + c)%x;
        long long d = gcd(y - x0,x);
        if( d != 1 && d != x)return d;
        if(y == x0)return x;
        if(i == k){y = x0; k += k;}
    }
}
 //对 n 进行素因子分解，存入 factor. k 设置为 107 左右即可
void findfac(long long n,int k){
    if(n == 1)return;
    if(Miller_Rabin(n))
    {
        factor[tol++] = n;
        return;
    }
    long long p = n;
    int c = k;
    while( p >= n)p = pollard_rho(p,c--);//值变化，防止死循环 k
    findfac(p,k);
    findfac(n/p,k);
}
 //POJ 1811
 //给出一个N(2 <= N < 2^54),如果是素数，输出"Prime"，否则输出最小的素因子
int main(){
    int T;
    long long n;
    scanf("%d",&T);
    while(T--){
        scanf("%I64d",&n);
        if(Miller_Rabin(n))printf("Prime\n");
        else{
            tol = 0;
            findfac(n,107);
            long long ans = factor[0];
            for(int i = 1;i < tol;i++)
            ans = min(ans,factor[i]);
            printf("%I64d\n",ans);
        }
    }
    return 0;
}
```

2.7 欧拉函数

2.7.1 分解质因素求欧拉函数

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

2.7.2 筛法欧拉函数

```
int euler[3000001];
void getEuler(){
    memset(euler,0,sizeof(euler));
    euler[1] = 1;
    for(int i = 2;i <= 3000000;i++)
        if(!euler[i])
           for(int j = i;j <= 3000000; j += i){
               if(!euler[j])
                   euler[j] = j;
               euler[j] = euler[j]/i*(i-1);
           }
}
```

2.7.3 求单个数的欧拉函数

```
long long eular(long long n){
    long long ans = n;
    for(int i = 2;i*i <= n;i++){
        if(n % i == 0){
            ans -= ans/i;
            while(n % i == 0)
                n /= i;
        }
    }
    if(n > 1)ans -= ans/n;
    return ans;
}
```

2.7.4 线性筛（同时得到欧拉函数和素数表）

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

```
int phi[MAXN+10];
int prime[MAXN+10];
int tot;//素数的个数
void phi_and_prime_table(int N){
    memset(check,false,sizeof(check));
    phi[1] = 1;
    tot = 0;
    for(int i = 2; i <= N; i++){
        if( !check[i] ){
            prime[tot++] = i;
            phi[i] = i-1;
        }
        for(int j = 0; j < tot; j++){
            if(i * prime[j] > N)break;
            check[i * prime[j]] = true;
            if( i % prime[j] == 0){
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            else{
                phi[i * prime[j]] = phi[i] * (prime[j] - 1);
            }
        }
    }
}
```

2.8 高斯消元（浮点数）

```
#define eps 1e-9
const int MAXN=220;
double a[MAXN][MAXN],x[MAXN];//方程的左边的矩阵和等式右边的值，求解之后 x
   存的就是结果
int equ,var;//方程数和未知数个数
/*
* 返回 0 表示无解,1 表示有解
*/
int Gauss(){
    int i,j,k,col,max_r;
    for(k=0,col=0;k<equ&&col<var;k++,col++){
        max_r=k;
        for(i=k+1;i<equ;i++)
          if(fabs(a[i][col])>fabs(a[max_r][col]))
            max_r=i;
        if(fabs(a[max_r][col])<eps)return 0;
        if(k!=max_r){
            for(j=col;j<var;j++)
              swap(a[k][j],a[max_r][j]);
            swap(x[k],x[max_r]);
        }
        x[k]/=a[k][col];
        for(j=col+1;j<var;j++)a[k][j]/=a[k][col];
        a[k][col]=1;
        for(i=0;i<equ;i++)
          if(i!=k){
              x[i]-=x[k]*a[i][col];
              for(j=col+1;j<var;j++)a[i][j]-=a[k][j]*a[i][col];
              a[i][col]=0;
          }
    }
    return 1;
}
```

2.9 FFT

```
//HDU 1402 求高精度乘法
const double PI = acos(-1.0);
//复数结构体
struct Complex{
    double x,y;//实部和虚部 x+yi
    Complex(double _x = 0.0,double _y = 0.0){
        x = _x;
        y = _y;
    }
    Complex operator -(const Complex &b)const{
        return Complex(x-b.x,y-b.y);
    }
    Complex operator +(const Complex &b)const{
        return Complex(x+b.x,y+b.y);
    }
    Complex operator *(const Complex &b)const{
        return Complex(x*b.x-y*b.y,x*b.y+y*b.x);
    }
};
/*
 * 进行 FFT 和 IFFT 前的反转变换。
 * 位置 i 和（i 二进制反转后位置）互换
 * len 必须为 2 的幂
 */
void change(Complex y[],int len){
    int i,j,k;
    for(i = 1, j = len/2;i <len-1;i++){
        if(i < j)swap(y[i],y[j]);
        //交换互为小标反转的元素，i<j 保证交换一次
        //i 做正常的 +1，j 左反转类型的 +1，始终保持 i 和 j 是反转的
        k = len/2;
        while(j >= k){
            j -= k;
            k /= 2;
        }
        if(j < k)j += k;
    }
}
/*
 * 做 FFT
 * len 必须为2^k形式
 * on==1 时是 DFT, on==-1 时是 IDFT
 */
void fft(Complex y[],int len,int on){
    change(y,len);
    for(int h = 2; h <= len; h <<= 1){
        Complex wn(cos(-on*2*PI/h),sin(-on*2*PI/h));
        for(int j = 0;j < len;j+=h){
            Complex w(1,0);
            for(int k = j;k < j+h/2;k++){
                Complex u = y[k];
                Complex t = w*y[k+h/2];
                y[k] = u+t;
                y[k+h/2] = u-t;
                w = w*wn;
            }
        }
    }
    if(on == -1)
        for(int i = 0;i < len;i++)
            y[i].x /= len;
}
const int MAXN = 200010;
Complex x1[MAXN],x2[MAXN];
char str1[MAXN/2],str2[MAXN/2];
int sum[MAXN];
int main(){
    while(scanf("%s%s",str1,str2)==2){
        int len1 = strlen(str1);
        int len2 = strlen(str2);
        int len = 1;
        while(len < len1*2 || len < len2*2)len<<=1;
        for(int i = 0;i < len1;i++)
            x1[i] = Complex(str1[len1-1-i]-'0',0);
        for(int i = len1;i < len;i++)
            x1[i] = Complex(0,0);
        for(int i = 0;i < len2;i++)
            x2[i] = Complex(str2[len2-1-i]-'0',0);
        for(int i = len2;i < len;i++)
            x2[i] = Complex(0,0);
        //求 DFT
        fft(x1,len,1);
        fft(x2,len,1);
        for(int i = 0;i < len;i++)
            x1[i] = x1[i]*x2[i];
        fft(x1,len,-1);
        for(int i = 0;i < len;i++)
            sum[i] = (int)(x1[i].x+0.5);
        for(int i = 0;i < len;i++){
            sum[i+1]+=sum[i]/10;
            sum[i]%=10;
        }
        len = len1+len2-1;
        while(sum[len] <= 0 && len > 0)len--;
        for(int i = len;i >= 0;i--)
            printf("%c",sum[i]+'0');
        printf("\n");
    }
    return 0;
}

//HDU 4609
//给出 n 条线段长度，问任取 3 根，组成三角形的概率。
//n<=10^5   用 FFT 求可以组成三角形的取法有几种
const int MAXN = 400040;
Complex x1[MAXN];
int a[MAXN/4];
long long num[MAXN];//100000*100000 会超 int
long long sum[MAXN];
int main(){
    int T;
    int n;
    scanf("%d",&T);
    while(T--){
        scanf("%d",&n);
        memset(num,0,sizeof(num));
        for(int i = 0;i < n;i++){
            scanf("%d",&a[i]);
            num[a[i]]++;
        }
        sort(a,a+n);
        int len1 = a[n-1]+1;
        int len = 1;
        while( len < 2*len1 )len <<= 1;
        for(int i = 0;i < len1;i++)
            x1[i] = Complex(num[i],0);
        for(int i = len1;i < len;i++)
            x1[i] = Complex(0,0);
        fft(x1,len,1);
        for(int i = 0;i < len;i++)
            x1[i] = x1[i]*x1[i];
        fft(x1,len,-1);
        for(int i = 0;i < len;i++)
            num[i] = (long long)(x1[i].x+0.5);
        len = 2*a[n-1];
        //减掉取两个相同的组合
        for(int i = 0;i < n;i++)
            num[a[i]+a[i]]--;
        for(int i = 1;i <= len;i++)num[i]/=2;
        sum[0] = 0;
        for(int i = 1;i <= len;i++)
            sum[i] = sum[i-1]+num[i];
        long long cnt = 0;
        for(int i = 0;i < n;i++){
            cnt += sum[len]-sum[a[i]];
            //减掉一个取大，一个取小的
            cnt -= (long long)(n-1-i)*i;
            //减掉一个取本身，另外一个取其它
            cnt -= (n-1);
            cnt -= (long long)(n-1-i)*(n-i-2)/2;
        }
        long long tot = (long long)n*(n-1)*(n-2)/6;
        printf("%.7lf\n",(double)cnt/tot);
    }
    return 0;
}
```

2.10 高斯消元法求方程组的解

2.10.1 一类开关问题，对 2 取模的 01 方程组

POJ 1681 需要枚举自由变元，找解中 1 个数最少的

```
//对 2 取模的 01 方程组
const int MAXN = 300;
//有 equ 个方程，var 个变元。增广矩阵行数为 equ，列数为 var+1，分别为 0 到 var
int equ,var;
int a[MAXN][MAXN]; //增广矩阵
int x[MAXN]; //解集
int free_x[MAXN];//用来存储自由变元（多解枚举自由变元可以使用）
int free_num;//自由变元的个数

//返回值为 -1 表示无解，为 0 是唯一解，否则返回自由变元个数
int Gauss(){
    int max_r,col,k;
    free_num = 0;
    for(k = 0, col = 0 ; k < equ && col < var ; k++, col++){
        max_r = k;
        for(int i = k+1;i < equ;i++){
            if(abs(a[i][col]) > abs(a[max_r][col]))
                max_r = i;
        }
        if(a[max_r][col] == 0){
            k--;
            free_x[free_num++] = col;//这个是自由变元
            continue;
        }
        if(max_r != k){
            for(int j = col; j < var+1; j++)
                swap(a[k][j],a[max_r][j]);
        }
        for(int i = k+1;i < equ;i++){
            if(a[i][col] != 0){
                for(int j = col;j < var+1;j++)
```

```
                    a[i][j] ^= a[k][j];
                }
            }
        }
    for(int i = k;i < equ;i++)
        if(a[i][col] != 0)
            return -1;//无解
    if(k < var) return var-k;//自由变元个数
    //唯一解，回代
    for(int i = var-1; i >= 0;i--){
        x[i] = a[i][var];
        for(int j = i+1;j < var;j++)
            x[i] ^= (a[i][j] && x[j]);
    }
    return 0;
}
int n;
void init(){
    memset(a,0,sizeof(a));
    memset(x,0,sizeof(x));
    equ = n*n;
    var = n*n;
    for(int i = 0;i < n;i++)
        for(int j = 0;j < n;j++){
            int t = i*n+j;
            a[t][t] = 1;
            if(i > 0)a[(i-1)*n+j][t] = 1;
            if(i < n-1)a[(i+1)*n+j][t] = 1;
            if(j > 0)a[i*n+j-1][t] = 1;
            if(j < n-1)a[i*n+j+1][t] = 1;
        }
}
void solve(){
    int t = Gauss();
    if(t == -1){
        printf("inf\n");
        return;
    }
    else if(t == 0){
        int ans = 0;
        for(int i = 0;i < n*n;i++)
            ans += x[i];
        printf("%d\n",ans);
        return;
    }
    else
    {
        //枚举自由变元
        int ans = 0x3f3f3f3f;
        int tot = (1<<t);
        for(int i = 0;i < tot;i++){
            int cnt = 0;
            for(int j = 0;j < t;j++){
                if(i&(1<<j)){
                    x[free_x[j]] = 1;
                    cnt++;
                }
                else x[free_x[j]] = 0;
            }
            for(int j = var-t-1;j >= 0;j--){
                int idx;
                for(idx = j;idx < var;idx++)
                    if(a[j][idx])
                        break;
                x[idx] = a[j][var];
                for(int l = idx+1;l < var;l++)
                    if(a[j][l])
                        x[idx] ^= x[l];
                cnt += x[idx];
            }
            ans = min(ans,cnt);
        }
        printf("%d\n",ans);
    }
}
char str[30][30];
int main(){
    int T;
    scanf("%d",&T);
    while(T--){
        scanf("%d",&n);
        init();
        for(int i = 0;i < n;i++){
            scanf("%s",str[i]);
            for(int j = 0;j < n;j++){
                if(str[i][j] == 'y')
                    a[i*n+j][n*n] = 0;
                else a[i*n+j][n*n] = 1;
            }
        }
        solve();
    }
    return 0;
}
```

2.10.2 解同余方程组

POJ 2947 Widget Factory

```
//求解对 MOD 取模的方程组
const int MOD = 7;
const int MAXN = 400;
int a[MAXN][MAXN];//增广矩阵
int x[MAXN];//最后得到的解集
inline int gcd(int a,int b){
    while(b != 0){
        int t = b;
        b = a%b;
        a = t;
    }
    return a;
}
inline int lcm(int a,int b){
    return a/gcd(a,b)*b;
}
long long inv(long long a,long long m){
    if(a == 1)return 1;
    return inv(m%a,m)*(m-m/a)%m;
}
int Gauss(int equ,int var){
    int max_r,col,k;
    for(k = 0, col = 0; k < equ && col < var; k++,col++){
        max_r = k;
        for(int i = k+1; i < equ;i++)
            if(abs(a[i][col]) > abs(a[max_r][col]))
                max_r = i;
        if(a[max_r][col] == 0){
            k--;
            continue;
        }
        if(max_r != k)
            for(int j = col; j < var+1;j++)
                swap(a[k][j],a[max_r][j]);
        for(int i = k+1;i < equ;i++){
            if(a[i][col] != 0){
                int LCM = lcm(abs(a[i][col]),abs(a[k][col]));
                int ta = LCM/abs(a[i][col]);
                int tb = LCM/abs(a[k][col]);
                if(a[i][col]*a[k][col] < 0)tb = -tb;
                for(int j = col;j < var+1;j++)
                    a[i][j] = ((a[i][j]*ta - a[k][j]*tb)%MOD + MOD)
                       %MOD;
            }
        }
    }
    for(int i = k;i < equ;i++)
        if(a[i][col] != 0)
            return -1;//无解
    if(k < var) return var-k;//多解
    for(int i = var-1;i >= 0;i--){
        int temp = a[i][var];
        for(int j = i+1; j < var;j++){
            if(a[i][j] != 0){
                temp -= a[i][j]*x[j];
                temp = (temp%MOD + MOD)%MOD;
            }
        }
        x[i] = (temp*inv(a[i][i],MOD))%MOD;
    }
    return 0;
}
int change(char s[]){
    if(strcmp(s,"MON") == 0) return 1;
    else if(strcmp(s,"TUE")==0) return 2;
    else if(strcmp(s,"WED")==0) return 3;
    else if(strcmp(s,"THU")==0) return 4;
    else if(strcmp(s,"FRI")==0) return 5;
    else if(strcmp(s,"SAT")==0) return 6;
    else return 7;
}
int main(){
    int n,m;
    while(scanf("%d%d",&n,&m) == 2){
        if(n == 0 && m == 0)break;
        memset(a,0,sizeof(a));
        char str1[10],str2[10];
        int k;
        for(int i = 0;i < m;i++){
            scanf("%d%s%s",&k,str1,str2);
            a[i][n] = ((change(str2) - change(str1) + 1)%MOD + MOD)
               %MOD;
            int t;
            while(k--){
                scanf("%d",&t);
                t--;
                a[i][t] ++;
                a[i][t]%=MOD;
            }
        }
        int ans = Gauss(m,n);
        if(ans == 0){
            for(int i = 0;i < n;i++)
                if(x[i] <= 2)
                    x[i] += 7;
            for(int i = 0;i < n-1;i++)printf("%d␣",x[i]);
            printf("%d\n",x[n-1]);
        }
        else if(ans == -1)printf("Inconsistent␣data.\n");
        else printf("Multiple␣solutions.\n");
    }
    return 0;
}
```

2.11 整数拆分

```
//HDU 4651
//把数 n 拆成几个数（小于等于 n）相加的形式，问有多少种拆法。
const int MOD = 1e9+7;
int dp[100010];
void init(){
    memset(dp,0,sizeof(dp));
    dp[0] = 1;
    for(int i = 1;i <= 100000;i++){
        for(int j = 1, r = 1; i - (3 * j * j - j) / 2 >= 0; j++, r
           *= -1){
            dp[i] += dp[i -(3 * j * j - j) / 2] * r;
            dp[i] %= MOD;
            dp[i] = (dp[i]+MOD)%MOD;
            if( i - (3 * j * j + j) / 2 >= 0 ){
                dp[i] += dp[i - (3 * j * j + j) / 2] * r;
                dp[i] %= MOD;
                dp[i] = (dp[i]+MOD)%MOD;
            }
        }
    }
}
int main(){
    int T;
    int n;
    init();
    scanf("%d",&T);
    while(T--){
        scanf("%d",&n);
        printf("%d\n",dp[n]);
    }
    return 0;
}

//HDU 4658
//数 n(<=10^5) 的划分,相同的数重复不能超过 k 个。
const int MOD = 1e9+7;
int dp[100010];
void init(){
    memset(dp,0,sizeof(dp));
    dp[0] = 1;
    for(int i = 1;i <= 100000;i++){
        for(int j = 1, r = 1; i - (3 * j * j - j) / 2 >= 0; j++, r
           *= -1){
            dp[i] += dp[i -(3 * j * j - j) / 2] * r;
            dp[i] %= MOD;
            dp[i] = (dp[i]+MOD)%MOD;
            if( i - (3 * j * j + j) / 2 >= 0 ){
                dp[i] += dp[i - (3 * j * j + j) / 2] * r;
                dp[i] %= MOD;
                dp[i] = (dp[i]+MOD)%MOD;
            }
        }
    }
}
int solve(int n,int k){
    int ans = dp[n];
    for(int j = 1, r = -1; n - k*(3 * j * j - j) / 2 >= 0; j++, r
       *= -1){
        ans += dp[n -k*(3 * j * j - j) / 2] * r;
        ans %= MOD;
        ans = (ans+MOD)%MOD;
        if( n - k*(3 * j * j + j) / 2 >= 0 ){
            ans += dp[n - k*(3 * j * j + j) / 2] * r;
            ans %= MOD;
            ans = (ans+MOD)%MOD;
        }
    }
    return ans;
}
int main(){
    init();
    int T;
    int n,k;
    scanf("%d",&T);
    while(T--){
        scanf("%d%d",&n,&k);
        printf("%d\n",solve(n,k));
    }
    return 0;
}
```

2.12 求 A^B 的约数之和对 MOD 取模

```
//参考 POJ 1845
//里面有一种求1+p+p^2+p…^3+p^n的方法。
//需要素数筛选和合数分解的程序，需要先调用 getPrime();
long long pow_m(long long a,long long n){
    long long ret = 1;
    long long tmp = a%MOD;
    while(n){
        if(n&1)ret = (ret*tmp)%MOD;
        tmp = tmp*tmp%MOD;
        n >>= 1;
    }
    return ret;
}
//计算1+p+p^2+...+p^n
long long sum(long long p,long long n){
    if(p == 0)return 0;
    if(n == 0)return 1;
    if(n & 1){
        return ((1+pow_m(p,n/2+1))%MOD*sum(p,n/2)%MOD)%MOD;
    }
    else return ((1+pow_m(p,n/2+1))%MOD*sum(p,n/2-1)+pow_m(p,n/2)%
       MOD)%MOD;
}
//返回A^B的约数之和 % MOD
long long solve(long long A,long long B){
    getFactors(A);
    long long ans = 1;
    for(int i = 0;i < fatCnt;i++){
        ans *= sum(factor[i][0],B*factor[i][1])%MOD;
        ans %= MOD;
    }
    return ans;
}
```

2.13 莫比乌斯反演

2.13.1 莫比乌斯函数

```
const int MAXN = 1000000;
bool check[MAXN+10];
int prime[MAXN+10];
int mu[MAXN+10];
void Moblus(){
    memset(check,false,sizeof(check));
    mu[1] = 1;
    int tot = 0;
    for(int i = 2; i <= MAXN; i++){
        if( !check[i] ){
            prime[tot++] = i;
            mu[i] = -1;
        }
        for(int j = 0; j < tot; j++){
            if(i * prime[j] > MAXN) break;
            check[i * prime[j]] = true;
            if( i % prime[j] == 0){
                mu[i * prime[j]] = 0;
                break;
            }
            else{
                mu[i * prime[j]] = -mu[i];
            }
        }
    }
}
```

2.13.2 例题：BZOJ2301

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

```
const int MAXN = 100000;
bool check[MAXN+10];
int prime[MAXN+10];
int mu[MAXN+10];
void Moblus(){
    memset(check,false,sizeof(check));
    mu[1] = 1;
    int tot = 0;
    for(int i = 2; i <= MAXN; i++){
        if( !check[i] ){
            prime[tot++] = i;
            mu[i] = -1;
        }
        for(int j = 0; j < tot; j ++){
            if( i * prime[j] > MAXN) break;
            check[i * prime[j]] = true;
            if( i % prime[j] == 0){
                mu[i * prime[j]] = 0;
                break;
            }
            else{
                mu[i * prime[j]] = -mu[i];
            }
        }
    }
}
int sum[MAXN+10];
//找 [1,n],[1,m] 内互质的数的对数
long long solve(int n,int m){
    long long ans = 0;
    if(n > m)swap(n,m);
    for(int i = 1, la = 0; i <= n; i = la+1){
        la = min(n/(n/i),m/(m/i));
        ans += (long long)(sum[la] - sum[i-1])*(n/i)*(m/i);
    }
    return ans;
}
int main(){
    Moblus();
    sum[0] = 0;
    for(int i = 1;i <= MAXN;i++)
        sum[i] = sum[i-1] + mu[i];
    int a,b,c,d,k;
    int T;
    scanf("%d",&T);
    while(T--){
        scanf("%d%d%d%d%d",&a,&b,&c,&d,&k);
        long long ans = solve(b/k,d/k) - solve((a-1)/k,d/k) - solve
           (b/k,(c-1)/k) + solve((a-1)/k,(c-1)/k);
        printf("%lld\n",ans);
    }
    return 0;
}
```

2.14 Baby-Step Giant-Step

```
//(POJ 2417,3243)
//baby_step giant_step
// a^x = b (mod n) n 是素数和不是素数都可以
// 求解上式 0<=x < n 的解
#define MOD 76543
int hs[MOD],head[MOD],next[MOD],id[MOD],top;
void insert(int x,int y){
    int k = x%MOD;
    hs[top] = x, id[top] = y, next[top] = head[k], head[k] = top++;
}
int find(int x){
    int k = x%MOD;
    for(int i = head[k]; i != -1; i = next[i])
        if(hs[i] == x)
            return id[i];
    return -1;
}
int BSGS(int a,int b,int n){
    memset(head,-1,sizeof(head));
    top = 1;
    if(b == 1)return 0;
    int m = sqrt(n*1.0), j;
    long long x = 1, p = 1;
    for(int i = 0; i < m; ++i, p = p*a%n)insert(p*b%n,i);
    for(long long i = m; ;i += m){
        if( (j = find(x = x*p%n)) != -1 )return i-j;
        if(i > n)break;
    }
    return -1;
}
```

2.15 自适应 simpson 积分

```
double simpson(double a,double b){
    double c = a + (b-a)/2;
    return (F(a) + 4*F(c) + F(b))*(b-a)/6;
}
double asr(double a,double b,double eps,double A){
    double c = a + (b-a)/2;
    double L = simpson(a,c), R = simpson(c,b);
    if(fabs(L + R - A) <= 15*eps)return L + R + (L + R - A)/15.0;
    return asr(a,c,eps/2,L) + asr(c,b,eps/2,R);
}
double asr(double a,double b,double eps){
    return asr(a,b,eps,simpson(a,b));
}
```

2.16 斐波那契数列取模循环节

必要时要上 unsigned long long
HDU3977

```
long long gcd(long long a,long long b){
    if(b == 0)return a;
    return gcd(b,a%b);
}
long long lcm(long long a,long long b){
    return a/gcd(a,b)*b;
}
struct Matrix{
    long long mat[2][2];
};
Matrix mul_M(Matrix a,Matrix b,long long mod){
    Matrix ret;
    for(int i = 0;i < 2;i++)
        for(int j = 0;j < 2;j++){
            ret.mat[i][j] = 0;
            for(int k = 0;k < 2;k++){
                ret.mat[i][j] += a.mat[i][k]*b.mat[k][j]%mod;
                if(ret.mat[i][j] >= mod)ret.mat[i][j] -= mod;
            }
        }
    return ret;
}
Matrix pow_M(Matrix a,long long n,long long mod){
    Matrix ret;
    memset(ret.mat,0,sizeof(ret.mat));
    for(int i = 0;i < 2;i++)ret.mat[i][i] = 1;
    Matrix tmp = a;
    while(n){
        if(n&1)ret = mul_M(ret,tmp,mod);
        tmp = mul_M(tmp,tmp,mod);
        n >>= 1;
    }
    return ret;
}
long long pow_m(long long a,long long n,long long mod)//a^b % mod{
    long long ret = 1;
    long long tmp = a%mod;
    while(n){
        if(n&1)ret = ret*tmp%mod;
        tmp = tmp*tmp%mod;
        n >>= 1;
    }
    return ret;
}
//素数筛选和合数分解
const int MAXN = 1000000;
int prime[MAXN+1];
void getPrime(){
    memset(prime,0,sizeof(prime));
    for(int i = 2;i <= MAXN;i++){
        if(!prime[i])prime[++prime[0]] = i;
        for(int j = 1;j <= prime[0] && prime[j] <= MAXN/i;j++){
            prime[prime[j]*i] = 1;
            if(i%prime[j] == 0)break;
        }
    }
}
long long factor[100][2];
int fatCnt;
int getFactors(long long x){
    fatCnt = 0;
    long long tmp = x;
    for(int i = 1;prime[i] <= tmp/prime[i];i++){
        factor[fatCnt][1] = 0;
        if(tmp%prime[i] == 0){
            factor[fatCnt][0] = prime[i];
            while(tmp%prime[i] == 0){
                factor[fatCnt][1]++;
                tmp /= prime[i];
            }
            fatCnt++;
        }
    }
    if(tmp != 1){
        factor[fatCnt][0] = tmp;
        factor[fatCnt++][1] = 1;
    }
    return fatCnt;
}
//勒让德符号
int legendre(long long a,long long p){
    if(pow_m(a,(p-1)>>1,p) == 1)return 1;
    else return -1;
}
int f0 = 1;
int f1 = 1;
long long getFib(long long n,long long mod){
    if(mod == 1)return 0;
    Matrix A;
    A.mat[0][0] = 0;
    A.mat[1][0] = 1;
    A.mat[0][1] = 1;
    A.mat[1][1] = 1;
    Matrix B = pow_M(A,n,mod);
    long long ret = f0*B.mat[0][0] + f1*B.mat[1][0];
    return ret%mod;
}
long long fac[1000000];
long long G(long long p){
    long long num;
    if(legendre(5,p) == 1)num = p-1;
    else num = 2*(p+1);
    //找出 num 的所有约数
    int cnt = 0;
    for(long long i = 1;i*i <= num;i++)
        if(num%i == 0){
            fac[cnt++] = i;
            if(i*i != num)
                fac[cnt++] = num/i;
        }
    sort(fac,fac+cnt);
    long long ans;
    for(int i = 0;i < cnt;i++){
        if(getFib(fac[i],p) == f0 && getFib(fac[i]+1,p) == f1){
            ans = fac[i];
            break;
        }
    }
    return ans;
}
long long find_loop(long long n){
    getFactors(n);
    long long ans = 1;
    for(int i = 0;i < fatCnt;i++){
        long long record = 1;
        if(factor[i][0] == 2)record = 3;
        else if(factor[i][0] == 3)record = 8;
        else if(factor[i][0] == 5)record = 20;
        else  record = G(factor[i][0]);
        for(int j = 1;j < factor[i][1];j++)
            record *= factor[i][0];
        ans = lcm(ans,record);
    }
    return ans;
}
void init(){
    getPrime();
}
int main(){
    init();
    int T;
    int iCase = 0;
    int n;
    scanf("%d",&T);
    while(T--){
        iCase++;
        scanf("%d",&n);
        printf("Case␣#%d:␣%I64d\n",iCase,find_loop(n));
    }
    return 0;
}
```

2.17 原根

定义：设 $m > 1, gcd(a,m) = 1$, 使得 $a^d \equiv 1(mod \ \ m)$ 成立的最小的正整数 d 为 a 对模 m 的阶，记为 $\delta_m(a)$.
如果 $\delta_m(a) = \varphi(m)$, 则称 a 是模 m 的原根.
定理：若 $m > 1, (a,m) = 1$, 正整数 d 满足 $a^d \equiv 1(mod \ \ m)$, 则 $\delta_m(a)$ 整除 d.
定理：模 m 有原根的充要条件是 $m = 2, 4, p^n, 2p^n$, 其中 p 是奇质数，n 是任意正整数.
定理：如果模 m 有原根，那么它一定有 $\varphi(\varphi(m))$ 个原根.
定理：如果 p 是素数，那么素数 p 一定有原根，并且模 p 的原根的个数为 $\varphi(p-1)$.
求模素数 p 原根的方法：对 $p-1$ 素因子分解，即 $p-1 = p_1^{a_1} p_2^{a_2} ... p_k^{a_k}$ 的标准分解式，若恒有

$$g^{\frac{p-1}{p_i}} \neq 1(mod \ \ p)$$

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

求素数的最小原根程序

```
//*******************************************
//素数筛选和合数分解
const int MAXN=100000;
int prime[MAXN+1];
void getPrime(){
    memset(prime,0,sizeof(prime));
    for(int i=2;i<=MAXN;i++){
        if(!prime[i])prime[++prime[0]]=i;
        for(int j=1;j<=prime[0]&&prime[j]<=MAXN/i;j++){
            prime[prime[j]*i]=1;
            if(i%prime[j]==0) break;
        }
    }
}
long long factor[100][2];
int fatCnt;
int getFactors(long long x){
    fatCnt=0;
    long long tmp=x;
    for(int i=1;prime[i]<=tmp/prime[i];i++){
        factor[fatCnt][1]=0;
        if(tmp%prime[i]==0){
            factor[fatCnt][0]=prime[i];
            while(tmp%prime[i]==0){
                factor[fatCnt][1]++;
                tmp/=prime[i];
            }
            fatCnt++;
        }
    }
    if(tmp!=1){
        factor[fatCnt][0]=tmp;
        factor[fatCnt++][1]=1;
    }
    return fatCnt;
}
```

```
//*****************************************
long long pow_m(long long a,long long n,long long mod){
    long long ret = 1;
    long long tmp = a%mod;
    while(n){
        if(n&1)ret = ret*tmp%mod;
        tmp = tmp*tmp%mod;
        n >>= 1;
    }
    return ret;
}
//求素数 P 的最小的原根
void solve(int P){
    if(P == 2){
        printf("1\n");
        return;
    }
    getFactors(P-1);
    for(int g = 2; g < P;g++){
        bool flag = true;
        for(int i = 0;i < fatCnt;i++){
            int t = (P-1)/factor[i][0];
            if(pow_m(g,t,P) == 1){
                flag = false;
                break;
            }
        }
        if(flag){
            printf("%d\n",g);
            return;
        }
    }
}
int main(){
    getPrime();
    int T;
    int P;
    scanf("%d",&T);
    while(T--){
        scanf("%d",&P);
        solve(P);
    }
    return 0;
}
```

2.18 快速数论变换

2.18.1 HDU4656 卷积取模

HDU4656

$x_k = b * c^{(2k)} + d$, $F(x) = a_0x_0 + a_1x_1 + a_2x_2 + ... + a_{n-1}x_{n-1}$ Given $n, b, c, d, a_0, ..., a_{n-1}$, calculate $F(x_0), ..., F(x_{n-1})$.

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

其中 p_j 和 q_k 都是卷积，可以使用 NTT 进行快速计算。

```
//**********************************
//快速数论变换 (NTT)
//求 A 和 B 的卷积，结果对 P 取模
//做长度为 N1 的变换，选取两个质数 P1 和 P2
//P1-1 和 P2-1 必须是 N1 的倍数
//E1 和 E2 分别是 P1,P2 的原根
//F1 是 E1 模 P1 的逆元,F2 是 E2 模 P2 的逆元
//I1 是 N1 对模 P1 的逆元,I2 是 N1 对模 P2 的逆元
//
//然后使用中国剩余定理，保证了结果是小于 MM=P1*P2 的
//M1 = (P2 对 P1 的逆元)*P2
//M2 = (P1 对 P2 的逆元)*P1

const int P = 1000003;//结果对 P 取模
const int N1 = 262144;// 2^18
const int N2 = N1+1;//数组大小
const int P1 = 998244353;//P1 = 2^23 * 7 * 17 + 1
const int P2 = 995622913;//P2 = 2^19 * 3 * 3 * 211 + 1
const int E1 = 996173970;
const int E2 = 88560779;
const int F1 = 121392023;//E1*F1 = 1(mod P1)
const int F2 = 840835547;//E2*F2 = 1(mod P2)
const int I1 = 998240545;//I1*N1 = 1(mod P1)
const int I2 = 995619115;//I2*N1 = 1(mod P2)
const long long M1 = 397550359381069386LL;
const long long M2 = 596324591238590904LL;
const long long MM = 993874950619660289LL;//MM = P1*P2
//计算 x*y 对 z 取模
long long mul(long long x,long long y,long long z){
    return (x*y - (long long)(x/(long double)z*y+1e-3)*z+z)%z;
}
int trf(int x1,int x2){
    return (mul(M1,x1,MM)+mul(M2,x2,MM))%MM%P;
}
int A[N2],B[N2],C[N2];
int A1[N2],B1[N2],C1[N2];
void fft(int *A,int PM,int PW){
    for(int m = N1,h;h = m/2, m >= 2;PW = (long long)PW*PW%PM,m=h)
        for(int i = 0,w=1;i < h;i++, w = (long long)w*PW%PM)
            for(int j = i;j < N1;j += m){
                int k = j+h, x = (A[j]-A[k]+PM)%PM;
                (A[j]+=A[k])%=PM;
                A[k] = (long long)w*x%PM;
            }
    for(int i = 0,j = 1;j < N1-1;j++){
        for(int k = N1/2; k > (i^=k);k /= 2);
        if(j < i)swap(A[i],A[j]);
    }
}
//计算 A 和 B 的卷积，结果保存在 C 中，结果对 P 取模
void mul(){
    memset(C,0,sizeof(C));
    memcpy(A1,A,sizeof(A));
    memcpy(B1,B,sizeof(B));
    fft(A1,P1,E1); fft(B1,P1,E1);
    for(int i = 0;i < N1;i++)C1[i] = (long long)A1[i]*B1[i]%P1;
    fft(C1,P1,F1);
    for(int i = 0;i < N1;i++)C1[i] = (long long)C1[i]*I1%P1;
    fft(A,P2,E2); fft(B,P2,E2);
    for(int i = 0;i < N1;i++)C[i] = (long long)A[i]*B[i]%P2;
    fft(C,P2,F2);
    for(int i = 0;i < N1;i++)C[i] = (long long)C[i]*I2%P2;
    for(int i = 0;i < N1;i++)C[i] = trf(C1[i],C[i]);
}
int INV[P];//逆元
const int MAXN = 100010;
int F[MAXN];//阶乘
int a[MAXN];
int pd[MAXN];
int pb[MAXN];
int pc2[MAXN];
int p[MAXN];
int main()
{
    //预处理逆元
    INV[1] = 1;
    for(int i = 2;i < P;i++)
        INV[i] = (long long)P/i*(P-INV[P%i])%P;
    F[0] = 1;
    for(int i = 1;i < MAXN;i++)
        F[i] = (long long)F[i-1]*i%P;
    int n,b,c,d;
    while(scanf("%d%d%d%d",&n,&b,&c,&d) == 4){
        for(int i = 0;i < n;i++)scanf("%d",&a[i]);
        pd[0] = 1;
        for(int i = 1;i < n;i++)
            pd[i] = (long long)pd[i-1]*d%P;
        memset(A,0,sizeof(A));
        memset(B,0,sizeof(B));
        for(int i = 0;i < n;i++)
            A[i] = (long long)a[n-1-i]*F[n-1-i]%P;
        for(int i = 0;i < n;i++)
            B[i] = (long long)pd[i]*INV[F[i]]%P;
        mul();
        for(int i = 0;i < n;i++)p[i] = C[i];
        reverse(p,p+n);
        memset(A,0,sizeof(A));
        pb[0] = 1;
        for(int i = 1;i < n;i++)
            pb[i] = (long long)pb[i-1]*b%P;
        pc2[0] = 1;
        int c2 = (long long)c*c%P;
        for(int i = 1, s = c;i < n;i++){
            pc2[i] = (long long)pc2[i-1]*s%P;
            s = (long long)s*c2%P;
        }
        for(int i = 0;i < n;i++)
            A[i] = (long long)pb[i]*INV[F[i]]%P*p[i]%P*pc2[i]%P;
        memset(B,0,sizeof(B));
        B[0] = 1;
        for(int i = 1;i < n;i++)
            B[i] = B[N1-i] = INV[pc2[i]];
        mul();
        for(int i = 0;i < n;i++)C[i] = (long long)C[i]*pc2[i]%P;
        for(int i = 0;i < n;i++)
            printf("%d\n",C[i]);
    }
    return 0;
}
```

2.19 其它公式

2.19.1 Polya

设 G 是 p 个对象的一个置换群，用 k 种颜色去染这 p 个对象，若一种染色方案在群 G 的作用下变为另一种方案，则这两个方案当作是同一种方案，这样的不同染色方案数为：
$L = \frac{1}{|G|} \times \Sigma(k^{C(f)}), f \in G$

$C(f)$ 为循环节，$|G|$ 表示群的置换方法数

对于有 n 个位置的手镯，有 n 种旋转置换和 n 种翻转置换

对于旋转置换：
$C(f_i) = gcd(n,i)$,i 表示一次转过 i 颗宝石，$i = 0$ 时 $c = n$；

对于翻转置换：

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

如果 n 为奇数： $C(f) = \frac{n}{2} + 1$

3 数据结构

3.1 划分树

```
/*
 * 划分树（查询区间第 k 大）
 */
const int MAXN = 100010;
int tree[20][MAXN];//表示每层每个位置的值
int sorted[MAXN];//已经排序好的数
int toleft[20][MAXN];//toleft[p][i] 表示第 i 层从 1 到 i 有数分入左边

void build(int l,int r,int dep){
    if(l == r)return;
    int mid = (l+r)>>1;
    int same = mid - l + 1;//表示等于中间值而且被分入左边的个数
    for(int i = l; i <= r; i++) //注意是 l, 不是 one
        if(tree[dep][i] < sorted[mid])
            same--;
    int lpos = l;
    int rpos = mid+1;
    for(int i = l;i <= r;i++){
        if(tree[dep][i] < sorted[mid])
            tree[dep+1][lpos++] = tree[dep][i];
        else if(tree[dep][i] == sorted[mid] && same > 0){
            tree[dep+1][lpos++] = tree[dep][i];
            same--;
        }
        else
            tree[dep+1][rpos++] = tree[dep][i];
        toleft[dep][i] = toleft[dep][l-1] + lpos - l;
    }
    build(l,mid,dep+1);
    build(mid+1,r,dep+1);
}

//查询区间第 k 大的数,[L,R] 是大区间, [l,r] 是要查询的小区间
int query(int L,int R,int l,int r,int dep,int k){
    if(l == r)return tree[dep][l];
    int mid = (L+R)>>1;
    int cnt = toleft[dep][r] - toleft[dep][l-1];
    if(cnt >= k){
        int newl = L + toleft[dep][l-1] - toleft[dep][L-1];
        int newr = newl + cnt - 1;
        return query(L,mid,newl,newr,dep+1,k);
    }
    else{
        int newr = r + toleft[dep][R] - toleft[dep][r];
        int newl = newr - (r-l-cnt);
        return query(mid+1,R,newl,newr,dep+1,k-cnt);
    }
}
int main(){
    int n,m;
    while(scanf("%d%d",&n,&m)==2){
        memset(tree,0,sizeof(tree));
        for(int i = 1;i <= n;i++){
            scanf("%d",&tree[0][i]);
            sorted[i] = tree[0][i];
        }
        sort(sorted+1,sorted+n+1);
        build(1,n,0);
        int s,t,k;
        while(m--){
            scanf("%d%d%d",&s,&t,&k);
            printf("%d\n",query(1,n,s,t,0,k));
        }
    }
    return 0;
}
```

3.2 RMQ

3.2.1 一维

求最大值，数组下标从 1 开始。
求最小值，或者最大最小值下标，或者数组从 0 开始对应修改即可。

```
const int MAXN = 50010;
int dp[MAXN][20];
int mm[MAXN];
//初始化 RMQ, b 数组下标从 1 开始，从 0 开始简单修改
void initRMQ(int n,int b[]){
    mm[0] = -1;
    for(int i = 1; i <= n;i++){
        mm[i] = ((i&(i-1)) == 0)?mm[i-1]+1:mm[i-1];
        dp[i][0] = b[i];
    }
    for(int j = 1; j <= mm[n];j++)
        for(int i = 1;i + (1<<j) -1 <= n;i++)
            dp[i][j] = max(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);
}
//查询最大值
int rmq(int x,int y){
    int k = mm[y-x+1];
    return max(dp[x][k],dp[y-(1<<k)+1][k]);
}
```

3.2.2 二维

```
/*
 * 二维 RMQ，预处理复杂度 n*m*log*(n)*log(m)
 * 数组下标从 1 开始
 */
int val[310][310];
int dp[310][310][9][9];//最大值
int mm[310];//二进制位数减一，使用前初始化
void initRMQ(int n,int m){
    for(int i = 1;i <= n;i++)
        for(int j = 1;j <= m;j++)
            dp[i][j][0][0] = val[i][j];
    for(int ii = 0; ii <= mm[n]; ii++)
        for(int jj = 0; jj <= mm[m]; jj++)
            if(ii+jj)
                for(int i = 1; i + (1<<ii) - 1 <= n;i++)
                    for(int j = 1; j + (1<<jj) - 1 <= m;j++){
                        if(ii)dp[i][j][ii][jj] = max(dp[i][j][ii
                           -1][jj],dp[i+(1<<(ii-1))][j][ii-1][jj]);
                        else dp[i][j][ii][jj] = max(dp[i][j][ii][jj
                           -1],dp[i][j+(1<<(jj-1))][ii][jj-1]);
                    }
}
//查询矩形内的最大值 (x1<=x2,y1<=y2)
int rmq(int x1,int y1,int x2,int y2){
    int k1 = mm[x2-x1+1];
    int k2 = mm[y2-y1+1];
    x2 = x2 - (1<<k1) + 1;
    y2 = y2 - (1<<k2) + 1;
    return max(max(dp[x1][y1][k1][k2],dp[x1][y2][k1][k2]),max(dp[x2
       ][y1][k1][k2],dp[x2][y2][k1][k2]));
}
int main(){
    //在外面对 mm 数组进行初始化
    mm[0] = -1;
    for(int i = 1;i <= 305;i++)
        mm[i] = ((i&(i-1))==0)?mm[i-1]+1:mm[i-1];
    int n,m;
    int Q;
    int r1,c1,r2,c2;
    while(scanf("%d%d",&n,&m) == 2){
        for(int i = 1;i <= n;i++)
            for(int j = 1;j <= m;j++)
                scanf("%d",&val[i][j]);
        initRMQ(n,m);
        scanf("%d",&Q);
        while(Q--){
            scanf("%d%d%d%d",&r1,&c1,&r2,&c2);
            if(r1 > r2)swap(r1,r2);
            if(c1 > c2)swap(c1,c2);
            int tmp = rmq(r1,c1,r2,c2);
            printf("%d␣",tmp);
            if(tmp == val[r1][c1] || tmp == val[r1][c2] || tmp ==
               val[r2][c1] || tmp == val[r2][c2])
                printf("yes\n");
            else printf("no\n");
        }
    }
    return 0;
}
```

3.3 树链剖分

3.3.1 点权

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

```
const int MAXN = 50010;
struct Edge{
    int to,next;
}edge[MAXN*2];
int head[MAXN],tot;
int top[MAXN];//top[v] 表示 v 所在的重链的顶端节点
int fa[MAXN];//父亲节点
int deep[MAXN];//深度
int num[MAXN];//num[v] 表示以 v 为根的子树的节点数
int p[MAXN];//p[v] 表示 v 对应的位置
int fp[MAXN];//fp 和 p 数组相反
int son[MAXN];//重儿子
int pos;
void init(){
    tot = 0;
    memset(head,-1,sizeof(head));
    pos = 1;//使用树状数组，编号从头 1 开始
    memset(son,-1,sizeof(son));
}
void addedge(int u,int v){
    edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;
}
void dfs1(int u,int pre,int d){
    deep[u] = d;
    fa[u] = pre;
    num[u] = 1;
    for(int i = head[u];i != -1; i = edge[i].next){
        int v = edge[i].to;
        if(v != pre){
            dfs1(v,u,d+1);
            num[u] += num[v];
            if(son[u] == -1 || num[v] > num[son[u]])
                son[u] = v;
        }
    }
}
void getpos(int u,int sp){
    top[u] = sp;
    p[u] = pos++;
    fp[p[u]] = u;
    if(son[u] == -1) return;
    getpos(son[u],sp);
    for(int i = head[u];i != -1;i = edge[i].next){
        int v = edge[i].to;
        if( v != son[u] && v != fa[u])
            getpos(v,v);
    }
}

//树状数组
int lowbit(int x){
    return x&(-x);
}
int c[MAXN];
int n;
int sum(int i){
    int s = 0;
    while(i > 0)
    {
        s += c[i];
        i -= lowbit(i);
    }
    return s;
}
void add(int i,int val){
    while(i <= n){
        c[i] += val;
        i += lowbit(i);
    }
}
//u->v 的路径上点的值改变 val
void Change(int u,int v,int val){
    int f1 = top[u], f2 = top[v];
    int tmp = 0;
    while(f1 != f2){
        if(deep[f1] < deep[f2]){
            swap(f1,f2);
            swap(u,v);
        }
        add(p[f1],val);
        add(p[u]+1,-val);
        u = fa[f1];
        f1 = top[u];
    }
    if(deep[u] > deep[v]) swap(u,v);
    add(p[u],val);
    add(p[v]+1,-val);
}
int a[MAXN];
int main(){
    int M,P;
    while(scanf("%d%d%d",&n,&M,&P) == 3){
        int u,v;
        int C1,C2,K;
        char op[10];
        init();
        for(int i = 1;i <= n;i++){
            scanf("%d",&a[i]);
        }
        while(M--){
            scanf("%d%d",&u,&v);
            addedge(u,v);
            addedge(v,u);
        }
        dfs1(1,0,0);
        getpos(1,1);
        memset(c,0,sizeof(c));
        for(int i = 1;i <= n;i++){
            add(p[i],a[i]);
            add(p[i]+1,-a[i]);
        }
        while(P--){
            scanf("%s",op);
            if(op[0] == 'Q'){
                scanf("%d",&u);
                printf("%d\n",sum(p[u]));
            }
            else{
                scanf("%d%d%d",&C1,&C2,&K);
                if(op[0] == 'D')
                    K = -K;
                Change(C1,C2,K);
            }
        }
    }
    return 0;
}
```

3.3.2 边权

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

```
const int MAXN = 10010;
struct Edge{
    int to,next;
}edge[MAXN*2];
int head[MAXN],tot;
int top[MAXN];//top[v] 表示 v 所在的重链的顶端节点
int fa[MAXN]; //父亲节点
int deep[MAXN];//深度
int num[MAXN];//num[v] 表示以 v 为根的子树的节点数
int p[MAXN];//p[v] 表示 v 与其父亲节点的连边在线段树中的位置
int fp[MAXN];//和 p 数组相反
int son[MAXN];//重儿子
int pos;
void init(){
    tot = 0;
    memset(head,-1,sizeof(head));
    pos = 0;
    memset(son,-1,sizeof(son));
}
void addedge(int u,int v){
    edge[tot].to = v;edge[tot].next = head[u];head[u] = tot++;
}
//第一遍 dfs 求出 fa,deep,num,son
void dfs1(int u,int pre,int d){
    deep[u] = d;
    fa[u] = pre;
    num[u] = 1;
    for(int i = head[u];i != -1; i = edge[i].next){
        int v = edge[i].to;
        if(v != pre){
            dfs1(v,u,d+1);
            num[u] += num[v];
            if(son[u] == -1 || num[v] > num[son[u]])
                son[u] = v;
        }
    }
}
//第二遍 dfs 求出 top 和 p
void getpos(int u,int sp){
    top[u] = sp;
    p[u] = pos++;
    fp[p[u]] = u;
    if(son[u] == -1) return;
    getpos(son[u],sp);
    for(int i = head[u] ; i != -1; i = edge[i].next){
        int v = edge[i].to;
        if(v != son[u] && v != fa[u])
            getpos(v,v);
    }
}

//线段树
struct Node{
    int l,r;
    int Max;
}segTree[MAXN*3];
void build(int i,int l,int r){
    segTree[i].l = l;
    segTree[i].r = r;
    segTree[i].Max = 0;
    if(l == r)return;
    int mid = (l+r)/2;
    build(i<<1,l,mid);
    build((i<<1)|1,mid+1,r);
}
void push_up(int i){
    segTree[i].Max = max(segTree[i<<1].Max,segTree[(i<<1)|1].Max);
}
// 更新线段树的第 k 个值为 val
void update(int i,int k,int val){
    if(segTree[i].l == k && segTree[i].r == k){
        segTree[i].Max = val;
        return;
    }
    int mid = (segTree[i].l + segTree[i].r)/2;
    if(k <= mid)update(i<<1,k,val);
    else update((i<<1)|1,k,val);
    push_up(i);
}
//查询线段树中 [l,r] 的最大值
int query(int i,int l,int r){
    if(segTree[i].l == l && segTree[i].r == r)
        return segTree[i].Max;
    int mid = (segTree[i].l + segTree[i].r)/2;
    if(r <= mid)return query(i<<1,l,r);
    else if(l > mid)return query((i<<1)|1,l,r);
    else return max(query(i<<1,l,mid),query((i<<1)|1,mid+1,r));
}
//查询 u->v 边的最大值
int find(int u,int v){
    int f1 = top[u], f2 = top[v];
    int tmp = 0;
    while(f1 != f2){
        if(deep[f1] < deep[f2]){
            swap(f1,f2);
            swap(u,v);
        }
        tmp = max(tmp,query(1,p[f1],p[u]));
        u = fa[f1]; f1 = top[u];
    }
    if(u == v)return tmp;
    if(deep[u] > deep[v]) swap(u,v);
    return max(tmp,query(1,p[son[u]],p[v]));
}
int e[MAXN][3];
int main(){
    int T;
    int n;
    scanf("%d",&T);
    while(T--){
        init();
        scanf("%d",&n);
        for(int i = 0;i < n-1;i++){
            scanf("%d%d%d",&e[i][0],&e[i][1],&e[i][2]);
            addedge(e[i][0],e[i][1]);
            addedge(e[i][1],e[i][0]);
        }
        dfs1(1,0,0);
        getpos(1,1);
        build(1,0,pos-1);
        for(int i = 0;i < n-1; i++){
            if(deep[e[i][0]] > deep[e[i][1]])
                swap(e[i][0],e[i][1]);
            update(1,p[e[i][1]],e[i][2]);
        }
        char op[10];
        int u,v;
        while(scanf("%s",op) == 1){
            if(op[0] == 'D')break;
            scanf("%d%d",&u,&v);
            if(op[0] == 'Q')
                printf("%d\n",find(u,v));//查询 u->v 路径上边权的最大值
            else update(1,p[e[u-1][1]],v);//修改第 u 条边的长度为 v
        }
    }
    return 0;
}
```

3.4 伸展树（splay tree）

3.4.1 例题：HDU1890

```
const int MAXN = 100010;
struct Node;
Node* null;
struct Node{
    Node *ch[2],*fa;
    int size;
    int rev;
    Node(){
        ch[0] = ch[1] = fa = null; rev = 0;
    }
    inline void push_up(){
        if(this == null)return;
        size = ch[0]->size + ch[1]->size + 1;
    }
    inline void setc(Node* p,int d){
        ch[d] = p;
        p->fa = this;
    }
    inline bool d(){
        return fa->ch[1] == this;
    }
    void clear(){
        size = 1;
        ch[0] = ch[1] = fa = null;
        rev = 0;
    }
    void Update_Rev(){
        if(this == null)return;
        swap(ch[0],ch[1]);
        rev ^= 1;
    }
    inline void push_down(){
        if(this == null)return;
        if(rev){
            ch[0]->Update_Rev();
            ch[1]->Update_Rev();
            rev = 0;
        }
    }
    inline bool isroot(){
        return fa == null || this != fa->ch[0] && this != fa->ch
           [1];
    }
};
inline void rotate(Node* x)
{
    Node *f = x->fa, *ff = x->fa->fa;
    f->push_down();
    x->push_down();
    int c = x->d(), cc = f->d();
    f->setc(x->ch[!c],c);
    x->setc(f,!c);
    if(ff->ch[cc] == f)ff->setc(x,cc);
    else x->fa = ff;
    f->push_up();
}
inline void splay(Node* &root,Node* x,Node* goal)
{
    while(x->fa != goal){
        if(x->fa->fa == goal)rotate(x);
        else {
            x->fa->fa->push_down();
            x->fa->push_down();
            x->push_down();
            bool f = x->fa->d();
            x->d() == f ? rotate(x->fa):rotate(x);
            rotate(x);
        }
    }
    x->push_up();
    if(goal == null)root = x;
}
```

```
Node* get_kth(Node* r,int k)
{
    Node* x = r;
    x->push_down();
    while(x->ch[0]->size+1 != k){
        if(k < x->ch[0]->size+1)x = x->ch[0];
        else{
            k -= x->ch[0]->size+1;
            x = x->ch[1];
        }
        x->push_down();
    }
    return x;
}
Node* get_next(Node* p){
    p->push_down();
    p = p->ch[1];
    p->push_down();
    while(p->ch[0] != null){
        p = p->ch[0];
        p->push_down();
    }
    return p;
}
Node pool[MAXN],*tail;
Node *node[MAXN];
Node *root;
void build(Node* &x,int l,int r,Node* fa)
{
    if(l > r)return;
    int mid = (l+r)/2;
    x = tail++;
    x->clear();
    x->fa = fa;
    node[mid] = x;
    build(x->ch[0],l,mid-1,x);
    build(x->ch[1],mid+1,r,x);
    x->push_up();
}
void init(int n)
{
    tail = pool;
    null = tail++;
    null->fa = null->ch[0] = null->ch[1] = null;
    null->size = 0; null->rev = 0;
    Node *p = tail++;
    p->clear();
    root = p;
    p = tail++;
    p->clear();
    root->setc(p,1);
    build(root->ch[1]->ch[0],1,n,root->ch[1]);
    root->ch[1]->push_up();
    root->push_up();
}
int a[MAXN];
int b[MAXN];
bool cmp(int i,int j)
{
    if(a[i] !=  a[j])return a[i] < a[j];
    else return i < j;
}
int main()
{
    int n;
    while(scanf("%d",&n) == 1 && n){
        for(int i = 1;i <= n;i++){
            scanf("%d",&a[i]);
            b[i] = i;
        }
        init(n);
        sort(b+1,b+n+1,cmp);
        for(int i = 1;i <= n;i++){
            splay(root,node[b[i]],null);
            int sz = root->ch[0]->size;
            printf("%d",root->ch[0]->size);
            if(i == n)printf("\n");
            else printf("␣");
            splay(root,get_kth(root,i),null);
            splay(root,get_kth(root,sz+2),root);
            root->ch[1]->ch[0]->Update_Rev();
        }
    }
    return 0;
}
```

3.4.2 例题：HDU3726

```
const int MAXN = 20010;
struct Node;
Node* null;
struct Node
{
    Node *ch[2],*fa;//指向儿子和父亲结点
    int size,key;
    Node(){
        ch[0] = ch[1] = fa = null;
    }
    inline void setc(Node* p,int d){
        ch[d] = p;
        p->fa = this;
    }
    inline bool d(){
        return fa->ch[1] == this;
    }
    void push_up(){
        size = ch[0]->size + ch[1]->size + 1;
    }
    void clear(){
        size = 1;
        ch[0] = ch[1] = fa = null;
    }
    inline bool isroot()
    {
        return fa == null || this != fa->ch[0] && this != fa->ch
           [1];
    }
};
inline void rotate(Node* x)
{
    Node *f = x->fa, *ff = x->fa->fa;
    int c = x->d(), cc = f->d();
    f->setc(x->ch[!c],c);
    x->setc(f,!c);
    if(ff->ch[cc] == f)ff->setc(x,cc);
    else x->fa = ff;
    f->push_up();
}
inline void splay(Node* &root,Node* x,Node* goal)
{
    while(x->fa != goal){
        if(x->fa->fa == goal)rotate(x);
        else {
            bool f = x->fa->d();
            x->d() == f ? rotate(x->fa) : rotate(x);
            rotate(x);
        }
    }
    x->push_up();
    if(goal == null)root = x;
}
//找到 r 子树里面的第 k 个
Node* get_kth(Node* r,int k)
{
    Node* x = r;
    while(x->ch[0]->size+1 != k){
        if(k < x->ch[0]->size+1)x = x->ch[0];
        else {
            k -= x->ch[0]->size+1;
            x = x->ch[1];
        }
    }
    return x;
}
```

```
//在 root 的树中删掉 x
void erase(Node* &root,Node* x)
{
    splay(root,x,null);
    Node* t = root;
    if(t->ch[1] != null){
        root = t->ch[1];
        splay(root,get_kth(root,1),null);
        root->setc(t->ch[0],0);
    }
    else{
        root = root->ch[0];
    }
    root->fa = null;
    if(root != null)root->push_up();
}
void insert(Node* &root,Node* x)
{
    if(root == null){
        root = x;
        return;
    }
    Node* now = root;
    Node* pre = root->fa;
    while(now != null){
        pre = now;
        now = now->ch[x->key >= now->key];
    }
    x->clear();
    pre->setc(x,x->key >= pre->key);
    splay(root,x,null);
}
void merge(Node* &A,Node* B)
{
    if(A->size <= B->size)swap(A,B);
    queue<Node*>Q;
    Q.push(B);
    while(!Q.empty()){
        Node* fr = Q.front();
        Q.pop();
        if(fr->ch[0] != null)Q.push(fr->ch[0]);
        if(fr->ch[1] != null)Q.push(fr->ch[1]);
        fr->clear();
        insert(A,fr);
    }
}
Node pool[MAXN],*tail;

struct Edge
{
    int u,v;
```

```
}edge[60010];
int a[MAXN];
bool del[60010];
struct QUERY
{
    char op[10];
    int u,v;
}query[500010];
int y[500010];

Node* node[MAXN];
Node* root[MAXN];
int F[MAXN];
int find(int x)
{
    if(F[x] == -1)return x;
    return F[x] = find(F[x]);
}
void debug(Node *root)
{
    if(root == null)return;
    debug(root->ch[0]);
    printf("size:␣%d,␣key␣=␣%d\n",root->size,root->key);
    debug(root->ch[1]);
}

int main()
{
    int n,m;
    int iCase = 0;
    while(scanf("%d%d",&n,&m) == 2)
    {
        if(n == 0 && m == 0)break;
        iCase++;
        memset(F,-1,sizeof(F));
        tail = pool;
        null = tail++;
        null->size = 0; null->ch[0] = null->ch[1] = null->fa = null
           ;
        null->key = 0;
        for(int i = 1;i <= n;i++)scanf("%d",&a[i]);
        for(int i = 0;i < m;i++)
        {
            scanf("%d%d",&edge[i].u,&edge[i].v);
            del[i] = false;
        }
        int Q = 0;
        while(1)
        {
            scanf("%s",&query[Q].op);
            if(query[Q].op[0] == 'E')break;
            if(query[Q].op[0] == 'D'){
                scanf("%d",&query[Q].u);
                query[Q].u--;
                del[query[Q].u] = true;
            }
            else if(query[Q].op[0] == 'Q'){
                scanf("%d%d",&query[Q].u,&query[Q].v);
            }
            else{
                scanf("%d%d",&query[Q].u,&query[Q].v);
                y[Q] = a[query[Q].u];
                a[query[Q].u] = query[Q].v;
            }
            Q++;
        }
        for(int i = 1;i <= n;i++){
            node[i] = tail++;
            node[i]->clear();
            node[i]->key = a[i];
            root[i] = node[i];
        }
        for(int i = 0;i < m;i++)
            if(!del[i]){
                int u = edge[i].u;
                int v = edge[i].v;
                int t1 = find(u);
                int t2 = find(v);
                if(t1 == t2)continue;
                F[t2] = t1;
                merge(root[t1],root[t2]);
            }
        vector<int>ans;
        for(int i = Q-1;i >= 0;i--){
            if(query[i].op[0] == 'D'){
                int u = edge[query[i].u].u;
                int v = edge[query[i].u].v;
                int t1 = find(u);
                int t2 = find(v);
                if(t1 == t2)continue;
                F[t2] = t1;
                merge(root[t1],root[t2]);
            }
            else if(query[i].op[0] == 'Q'){
                int u = query[i].u;
                int k = query[i].v;
                u = find(u);
                if(k <= 0 || k > root[u]->size){
                    ans.push_back(0);
                }
                else{
                    k = root[u]->size - k + 1;
                    Node* p = get_kth(root[u],k);
                    ans.push_back(p->key);
                }
            }
            else{
                int u = query[i].u;
                int t1 = find(u);
                Node* p = node[u];
                erase(root[t1],p);
                p->clear();
                p->key = y[i];
                a[u] = y[i];
                insert(root[t1],p);
            }
        }
        double ret = 0;
        int sz = ans.size();
        for(int i = 0;i < sz;i++)ret += ans[i];
        if(sz)ret /= sz;
        printf("Case␣%d:␣%.6lf\n",iCase,ret);
    }
    return 0;
}
```

3.5 动态树

3.5.1 SPOJQTREE

给定一棵 n 个结点的树，树的边上有权。有两种操作：
1. 修改一条边上的权值。
2. 查询两个结点 x 和 y 之间的最短路径中经过的最大的边的权值。
其中$n <= 10^4$

```
// http://www.spoj.com/problems/QTREE/
const int MAXN = 10010;
struct Node *null;
struct Node{
    Node *fa,*ch[2];
    int Max,key;
    inline void push_up(){
        if(this == null)return;
        Max = max(key,max(ch[0]->Max,ch[1]->Max));
    }
    inline void setc(Node *p,int d){
        ch[d] = p;
        p->fa = this;
    }
    inline bool d(){
        return  fa->ch[1] == this;
    }
    inline bool isroot() {
        return fa == null || fa->ch[0] != this && fa->ch[1] != this
           ;
    }
    inline void rot(){
        Node *f = fa,*ff = fa->fa;
        int c = d(), cc = fa->d();
        f->setc(ch[!c],c);
        this->setc(f,!c);
        if(ff->ch[cc] == f)ff->setc(this,cc);
        else this->fa = ff;
        f->push_up();
    }
    inline Node* splay(){
        while(!isroot()){
            if(!fa->isroot())
                d()==fa->d() ? fa->rot() : rot();
            rot();
        }
        push_up();
        return this;
    }
    inline Node* access(){
        for(Node *p = this,*q = null; p != null; q = p, p = p->fa){
            p->splay()->setc(q,1);
            p->push_up();
        }
        return splay();
    }
};
Node pool[MAXN],*tail;
Node *node[MAXN];
void init(int n){
    tail = pool;
    null = tail++;
    null->fa = null->ch[0] = null->ch[1] = null;
    null->Max = null->key = 0;
    for(int i = 1;i <= n;i++){
        node[i] = tail++;
        node[i]->fa = node[i]->ch[0] = node[i]->ch[1] = null;
        node[i]->Max = node[i]->key = 0;
    }
}
struct Edge{
    int to,next;
    int w,id;
}edge[MAXN*2];
int head[MAXN],tot;
inline int addedge(int u,int v,int w,int id){
    edge[tot].to = v;
    edge[tot].w = w;
    edge[tot].id = id;
    edge[tot].next = head[u];
    head[u] = tot++;
}
Node *ee[MAXN];
bool vis[MAXN];
void bfs(int n){
    for(int i = 1;i <= n;i++)vis[i] = false;
    queue<int>q;
    q.push(1);
    vis[1] = true;
    while(!q.empty()){
        int u = q.front();
        q.pop();
        for(int i = head[u];i != -1;i = edge[i].next){
            int v = edge[i].to;
            if(vis[v])continue;
            vis[v] = true;
            q.push(v);
            ee[edge[i].id] = node[v];
            node[v]->key = edge[i].w;
            node[v]->push_up();
            node[v]->fa = node[u];
        }
    }
}
inline int ask(Node *x,Node *y){
    x->access();
    for(x = null; y != null; x = y, y = y->fa){
        y->splay();
        if(y->fa == null)return max(y->ch[1]->Max,x->Max);
        y->setc(x,1);
        y->push_up();
    }
}
int main()
{
    int T;
    scanf("%d",&T);
    int n;
    while(T--){
        scanf("%d",&n);
        for(int i = 1;i <= n;i++)head[i] = -1;
        tot = 0;
        init(n);
        int u,v,w;
        for(int i = 1;i < n;i++){
            scanf("%d%d%d",&u,&v,&w);
            addedge(u,v,w,i);
            addedge(v,u,w,i);
        }
        bfs(n);
        char op[20];
        int x,y;
        while(scanf("%s",op) == 1){
            if(strcmp(op,"DONE") == 0)break;
            scanf("%d%d",&x,&y);
            if(op[0] == 'Q'){
                printf("%d\n",ask(node[x],node[y]));
            }
            else {
                ee[x]->splay()->key = y;
                ee[x]->push_up();
            }
        }
    }
    return 0;
}
```

3.5.2 SPOJQTREE2

给定一棵 n 个结点的树，树的边上有权。有两种操作：
1. 查询两个结点 x 和 y 之间的最短路径长度。
2. 查询从 x 到 y 的最短路径的第 K 条边的长度。
其中 $n <= 10^4$

```
// http://www.spoj.com/problems/QTREE2/
const int MAXN = 10010;
struct Node *null;
struct Node{
    Node *fa,*ch[2];
    int sum,val;
    int size;
    int id;
    void clear(){
        fa = ch[0] = ch[1] = null;
        sum = val = 0;
        size = 1;
    }
    inline void push_up(){
        if(this == null)return;
        sum = val + ch[0]->sum + ch[1]->sum;
        size = ch[0]->size + ch[1]->size + 1;
    }
    inline void setc(Node *p,int d){
        ch[d] = p;
        p->fa = this;
    }
    inline bool d(){
        return fa->ch[1] == this;
    }
    inline bool isroot(){
        return fa == null || fa->ch[0] != this && fa->ch[1] != this
            ;
    }
    inline void rot(){
        Node *f = fa, *ff = fa->fa;
        int c = d(), cc = fa->d();
        f->setc(ch[!c],c);
        this->setc(f,!c);
        if(ff->ch[cc] == f)ff->setc(this,cc);
        else this->fa = ff;
        f->push_up();
    }
    inline Node* splay(){
        while(!isroot()){
            if(!fa->isroot())
                d()==fa->d() ? fa->rot() : rot();
            rot();
        }
        push_up();
        return this;
    }
    inline Node* access(){
        for(Node *p = this,*q = null; p != null; q = p, p = p->fa){
            p->splay()->setc(q,1);
            p->push_up();
        }
        return splay();
    }
};
Node pool[MAXN],*tail;
Node *node[MAXN];
void init(int n){
    tail = pool;
    null = tail++;
    null->fa = null->ch[0] = null->ch[1] = null;
    null->size = null->sum = null->val = 0;
    for(int i = 1;i <= n;i++){
        node[i] = tail++;
        node[i]->id = i;
        node[i]->clear();
    }
}
struct Edge{
    int to,next;
    int w;
}edge[MAXN*2];
int head[MAXN],tot;
inline void addedge(int u,int v,int w){
    edge[tot].to = v;
    edge[tot].w = w;
    edge[tot].next = head[u];
    head[u] = tot++;
}
void dfs(int u,int pre){
    for(int i = head[u];i != -1;i = edge[i].next){
        int v = edge[i].to;
        if(v == pre)continue;
        dfs(v,u);
        node[v]->val = edge[i].w;
        node[v]->push_up();
        node[v]->fa = node[u];
    }
}
//查询 x->y 的距离
inline int query_sum(Node *x,Node *y){
    x->access();
    for(x = null; y != null; x = y, y = y->fa){
        y->splay();
        if(y->fa == null)
            return y->ch[1]->sum + x->sum;
        y->setc(x,1);
        y->push_up();
    }
}
//在 splay 中得到第 k 个点
Node* get_kth(Node* r,int k){
    Node *x = r;
    while(x->ch[0]->size+1 != k){
        if(k < x->ch[0]->size+1)x = x->ch[0];
        else {
            k -= x->ch[0]->size+1;
            x = x->ch[1];
        }
    }
    return x;
}
//查询 x->y 路径上的第 k 个点
inline int query_kth(Node *x,Node *y,int k){
    x->access();
    for(x = null; y != null; x = y, y = y->fa){
        y->splay();
        if(y->fa == null){
            if(y->ch[1]->size+1 == k)return y->id;
            else if(y->ch[1]->size+1 > k)
                return get_kth(y->ch[1],y->ch[1]->size+1-k)->id;
            else return get_kth(x,k-(y->ch[1]->size+1))->id;
        }
        y->setc(x,1);
        y->push_up();
    }
}
int main()
```

```
{
    int T,n;
    scanf("%d",&T);
    while(T--){
        scanf("%d",&n);
        for(int i = 1;i <= n;i++)head[i] = -1;
        tot = 0;
        init(n);
        int u,v,w;
        for(int i = 1;i < n;i++){
            scanf("%d%d%d",&u,&v,&w);
            addedge(u,v,w);
            addedge(v,u,w);
        }
        dfs(1,1);
        char op[20];
        while(scanf("%s",op) == 1){
            if(strcmp(op,"DONE") == 0)break;
            if(op[0] == 'D'){
                scanf("%d%d",&u,&v);
                printf("%d\n",query_sum(node[u],node[v]));
            }
            else {
                int k; scanf("%d%d%d",&u,&v,&k);
                printf("%d\n",query_kth(node[u],node[v],k));
            }
        }
    }
    return 0;
}
```

3.5.3 SPOJQTREE4

给定一棵 n 个结点的树，树的边上有权，每个结点有黑白两色，初始时所有的结点都是白的。有两种操作：
1. 对一个结点执行反色操作（白变黑，黑变白）
2. 查询树中距离最远的两个白点的距离。
其中$n <= 10^5$，查询数目不超过10^5.

```
//http://www.spoj.com/problems/QTREE4/
const int MAXN = 100010;
const int INF = 0x3f3f3f3f;
struct Node *null;
struct Node{
    Node *fa,*ch[2];
    multiset<int>st0,st1;//st0 是链，st1 是路径
    int dd,d0;//d0 是该点对应边的长度，dd 是重链长度
    int w0;//白点值为 0，黑点值为 -INF
    int ls,rs,ms;
    inline void clear(){
        fa = ch[0] = ch[1] = null;
        st0.clear(); st1.clear();
        st0.insert(-INF);
        st0.insert(-INF);
        st1.insert(-INF);
        w0 = 0; dd = d0 = 0;
        ls = rs = ms = -INF;
    }
    inline void push_up(){
        if(this == null)return;
        dd = d0 + ch[0]->dd + ch[1]->dd;
        int m0 = max(w0,*st0.rbegin()), ml = max(m0,ch[0]->rs+d0),
           mr = max(m0,ch[1]->ls);
        ls = max(ch[0]->ls,ch[0]->dd + d0 + mr);
        rs = max(ch[1]->rs,ch[1]->dd + ml);
        multiset<int>::reverse_iterator it = st0.rbegin();
        ++it;
        int t0 = max((*st0.rbegin()) + (*it) , *st1.rbegin());
        if(w0 == 0)
            t0 = max(t0,max(0,*st0.rbegin()));
        ms = max(max(max(ml+ch[1]->ls,mr+d0+ch[0]->rs),max(ch[0]->
           ms,ch[1]->ms)),t0);
    }
    inline void setc(Node *p,int d){
        ch[d] = p;
        p->fa = this;
    }
    inline bool d(){
        return fa->ch[1] == this;
    }
    inline bool isroot(){
        return fa == null || fa->ch[0] != this && fa->ch[1] != this
           ;
    }
    inline void rot(){
        Node *f = fa, *ff = fa->fa;
        int c = d(), cc = fa->d();
        f->setc(ch[!c],c);
        this->setc(f,!c);
        if(ff->ch[cc] == f)ff->setc(this,cc);
        else this->fa = ff;
        f->push_up();
    }
    inline Node* splay(){
        while(!isroot()){
            if(!fa->isroot())
                d()==fa->d() ? fa->rot() : rot();
            rot();
        }
        push_up();
        return this;
    }
    inline Node* access(){
        for(Node *p = this,*q = null; p != null; q = p, p = p->fa){
            p->splay();
            if(p->ch[1] != null){
                p->st0.insert(p->ch[1]->ls);
                p->st1.insert(p->ch[1]->ms);
            }
            if(q != null){
                p->st0.erase(p->st0.find(q->ls));
                p->st1.erase(p->st1.find(q->ms));
            }
            p->setc(q,1);
            p->push_up();
        }
        return splay();
    }
};
Node pool[MAXN],*tail;
Node *node[MAXN];
inline void init(int n){
    tail = pool;
    null = tail++;
    null->fa = null->ch[0] = null->ch[1] = null;
    null->st0.clear(); null->st1.clear();
    null->ls = null->rs = null->ms = -INF;
    null->w0 = -INF;
    null->d0 = null->dd = 0;
    for(int i = 1;i <= n;i++){
        node[i] = tail++;
        node[i]->clear();
    }
}
struct Edge{
    int to,next,w;
}edge[MAXN*2];
int head[MAXN],tot;
inline void addedge(int u,int v,int w){
    edge[tot].to = v;
    edge[tot].w = w;
    edge[tot].next = head[u];
    head[u] = tot++;
}
inline void dfs(int u,int pre){
    for(int i = head[u];i != -1;i = edge[i].next){
        int v = edge[i].to;
        if(v == pre)continue;
        node[v]->fa = node[u];
        node[v]->d0 = edge[i].w;
        dfs(v,u);
        node[u]->st0.insert(node[v]->ls);
        node[u]->st1.insert(node[v]->ms);
    }
    node[u]->push_up();
}
template <class T>
inline bool scan_d(T &ret) {
   char c; int sgn;
   if(c=getchar(),c==EOF) return 0;
   while(c!='-'&&(c<'0'||c>'9')) c=getchar();
   sgn=(c=='-')?-1:1;
   ret=(c=='-')?0:(c-'0');
   while(c=getchar(),c>='0'&&c<='9') ret=ret*10+(c-'0');
   ret*=sgn;
   return 1;
}
int main()
{
    int n;
    while(scanf("%d",&n) == 1){
        for(int i = 1;i <= n;i++)head[i] = -1;
        tot = 0;
        init(n);
        int u,v,w;
        for(int i = 1;i < n;i++){
            scan_d(u); scan_d(v);scan_d(w);
            addedge(u,v,w);
            addedge(v,u,w);
        }
        dfs(1,1);
        int ans = node[1]->ms;
        int Q;
        char op[10];
        scanf("%d",&Q);
        while(Q--){
            scanf("%s",op);
            if(op[0] == 'C'){
                scan_d(u);
                node[u]->access();
                node[u]->splay();
                if(node[u]->w0 == 0)node[u]->w0 = -INF;
                else node[u]->w0 = 0;
                node[u]->push_up();
                ans = node[u]->ms;
            }
            else{
                if(ans < 0)puts("They␣have␣disappeared.");
                else printf("%d\n",ans);
            }
        }
    }
    return 0;
}
```

3.5.4 SPOJQTREE5

给定一棵 n 个结点的树，边权均为 1。每个结点有黑白两色，初始时所有结点都是黑的。两种查询操作：
1. 对一个结点执行反色操作（白变黑，黑变白）
2. 查询距离某个特定结点 i 最远的白点的距离。
其中$<= 10^5$，查询数目不超过10^5.

```cpp
//http://www.spoj.com/problems/QTREE5/
const int MAXN = 100010;
const int INF = 0x3f3f3f3f;
struct Node *null;
struct Node{
    Node *fa,*ch[2];
    multiset<int>st;
    int dd,d0;
    int w0;
    int ls,rs;
    inline void clear(){
        fa = ch[0] = ch[1] = null;
        st.clear();
        st.insert(INF);
        w0 = INF; dd = d0 = 0;
        ls = rs = INF;
    }
    inline void push_up(){
        if(this == null)return;
        dd = d0 + ch[0]->dd + ch[1]->dd;
        int m0 = min(w0,*st.begin()), ml = min(m0,ch[0]->rs+d0), mr
            = min(m0,ch[1]->ls);
        ls = min(ch[0]->ls,ch[0]->dd + d0 + mr);
        rs = min(ch[1]->rs,ch[1]->dd + ml);
    }
    inline void setc(Node *p,int d){
        ch[d] = p;
        p->fa = this;
    }
    inline bool d(){
        return fa->ch[1] == this;
    }
    inline bool isroot(){
        return fa == null || fa->ch[0] != this && fa->ch[1] != this
           ;
    }
    inline void rot(){
        Node *f = fa, *ff = fa->fa;
        int c = d(), cc = fa->d();
        f->setc(ch[!c],c);
        this->setc(f,!c);
        if(ff->ch[cc] == f)ff->setc(this,cc);
        else this->fa = ff;
        f->push_up();
    }
    inline Node* splay(){
        while(!isroot()){
            if(!fa->isroot())
                d()==fa->d() ? fa->rot() : rot();
            rot();
        }
        push_up();
        return this;
    }
    inline Node* access(){
        for(Node *p = this,*q = null; p != null; q = p, p = p->fa){
            p->splay();
            if(p->ch[1] != null){
                p->st.insert(p->ch[1]->ls);
            }
            if(q != null){
                p->st.erase(p->st.find(q->ls));
            }
            p->setc(q,1);
            p->push_up();
        }
        return splay();
    }
};
Node pool[MAXN],*tail;
Node *node[MAXN];
inline void init(int n){
    tail = pool;
    null = tail++;
    null->fa = null->ch[0] = null->ch[1] = null;
    null->st.clear();
    null->ls = null->rs = INF;
    null->w0 = INF;
    null->dd = null->d0 = 0;
    for(int i = 1;i <= n;i++){
        node[i] = tail++;
        node[i]->clear();
    }
}
struct Edge{
    int to,next;
}edge[MAXN*2];
int head[MAXN],tot;
inline void addedge(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
inline void dfs(int u,int pre){
    for(int i = head[u];i != -1;i = edge[i].next){
        int v = edge[i].to;
        if(v == pre)continue;
        node[v]->fa = node[u];
        node[v]->d0 = 1;
        dfs(v,u);
        node[u]->st.insert(node[v]->ls);
    }
    node[u]->push_up();
}
int main()
{
    int n;
    while(scanf("%d",&n) == 1){
        init(n);
        for(int i = 1;i <= n;i++)head[i] = -1;
        tot = 0;
        int u,v;
        for(int i = 1;i < n;i++){
            scanf("%d%d",&u,&v);
            addedge(u,v);
            addedge(v,u);
        }
        dfs(1,1);
        int Q;
        scanf("%d",&Q);
        int op;
        while(Q--){
            scanf("%d%d",&op,&v);
            if(op == 0){
                node[v]->access();
                node[v]->splay();
                if(node[v]->w0 == 0)node[v]->w0 = INF;
                else node[v]->w0 = 0;
                node[v]->push_up();
            }
            else {
                node[v]->access();
                node[v]->splay();
                if(node[v]->rs < INF)printf("%d\n",node[v]->rs);
                else printf("-1\n");
            }
        }
    }
    return 0;
}
```

3.5.5 SPOJQTREE6

给定一棵 n 个结点的树，每个结点有黑白两色，初始时所有结点都是黑的。你被要求支持：
1. 对一个结点执行反色操作（白变黑，黑变白）

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

```
//http://www.spoj.com/problems/QTREE6/
const int MAXN = 100010;
struct Node *null;
struct Node{
    Node *fa,*ch[2];
    int co;//0 is black, 1 is white
    int lco,rco;
    int ls,rs;
    int s[2];
    int sum[2];//the sum of black and white
    inline void clear(){
        fa = ch[0] = ch[1] = null;
        co = lco = rco = 0;
        ls = rs = 1;
        s[0] = s[1] = 0;
        sum[0] = 1;  sum[1] = 0;
    }
    inline void push_up(){
        if(this == null)return;
        if(ch[0] != null)lco = ch[0]->lco;
        else lco = co;
        if(ch[1] != null)rco = ch[1]->rco;
        else rco = co;
        sum[0] = ch[0]->sum[0] + ch[1]->sum[0] + (co == 0);
        sum[1] = ch[0]->sum[1] + ch[1]->sum[1] + (co == 1);
        int ml = 1 + s[co] + (co==ch[0]->rco?ch[0]->rs:0);
        int mr = 1 + s[co] + (co==ch[1]->lco?ch[1]->ls:0);
        ls = ch[0]->ls;
        if(lco == co && ch[0]->sum[!co] == 0)ls += mr;
        rs = ch[1]->rs;
        if(rco == co && ch[1]->sum[!co] == 0)rs += ml;
    }
    inline void setc(Node *p,int d){
        ch[d] = p;
        p->fa = this;
    }
    inline bool d(){
        return fa->ch[1] == this;
    }
    inline bool isroot(){
        return fa == null || fa->ch[0] != this && fa->ch[1] != this
           ;
    }
    inline void rot(){
        Node *f = fa, *ff = fa->fa;
        int c = d(), cc = fa->d();
        f->setc(ch[!c],c);
        this->setc(f,!c);
        if(ff->ch[cc] == f)ff->setc(this,cc);
        else this->fa = ff;
        f->push_up();
    }
    inline Node* splay(){
        while(!isroot()){
            if(!fa->isroot())
                d()==fa->d() ? fa->rot() : rot();
            rot();
        }
        push_up();
        return this;
    }
    inline Node* access(){
        for(Node *p = this,*q = null; p != null; q = p, p = p->fa){
            p->splay();
            if(p->ch[1] != null)
                p->s[p->ch[1]->lco] += p->ch[1]->ls;
            if(q != null)
                p->s[q->lco] -= q->ls;
            p->setc(q,1);
            p->push_up();
        }
        return splay();
    }
};
Node pool[MAXN],*tail;
Node *node[MAXN];
void init(int n){
    tail = pool;
    null = tail++;
    null->fa = null->ch[0] = null->ch[1] = null;
    null->s[0] = null->s[1] = 0;
    null->ls = null->rs = 0;
    null->sum[0] = null->sum[1] = 0;
    null->co = null->lco = null->rco = 0;
    for(int i = 1;i <= n;i++){
        node[i] = tail++;
        node[i]->clear();
    }
}
struct Edge{
    int to,next;
}edge[MAXN*2];
int head[MAXN],tot;
inline void addedge(int u,int v){
    edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;
}
void dfs(int u,int pre){
    for(int i = head[u];i != -1;i = edge[i].next){
        int v = edge[i].to;
        if(v == pre)continue;
        node[v]->fa = node[u];
        dfs(v,u);
        node[u]->s[node[v]->lco] += node[v]->ls;
    }
    node[u]->push_up();
}
int main()
{
    int n;
    while(scanf("%d",&n) == 1){
        init(n);
        for(int i = 1;i <= n;i++)head[i] = -1;
        tot = 0;
        int u,v;
        for(int i = 1;i < n;i++){
            scanf("%d%d",&u,&v);
            addedge(u,v);
            addedge(v,u);
        }
        dfs(1,1);
        int Q;
        int op;
        scanf("%d",&Q);
        while(Q--){
            scanf("%d%d",&op,&u);
            if(op == 0){
                node[u]->access();
                node[u]->splay();
                printf("%d\n",node[u]->rs);
            }
            else{
                node[u]->access();
                node[u]->splay();
                node[u]->co ^= 1;
                node[u]->push_up();
            }
        }
        return 0;
    }
    return 0;
}
```

3.5.6 SPOJQTREE7

给定一棵 n 个结点的树，每个结点有黑白两色和权值。三种操作：
1. 对一个结点执行反色操作（白变黑，黑变白）
2. 询问与 u 相连的点中点权的最大值。两个结点 u,v 相连当且仅当 u,v 路径上所有点的颜色相同。
3. 改变一个点的点权。
其中$n <= 10^5$，查询数目不超过10^5.

```
//http://www.spoj.com/problems/QTREE7/
const int MAXN = 100010;
const int INF = 0x3f3f3f3f;
struct Node *null;
struct Node{
    Node *fa,*ch[2];
    int co;
    int lco,rco;
    int ls,rs;
    int w0;
    multiset<int>st[2];
    int sum[2];
    inline void clear(int _co = 0, int _w0 = 0){
        fa = ch[0] = ch[1] = null;
        co = lco = rco = _co;
        w0 = _w0;
        ls = rs = _w0;
        st[0].clear(); st[1].clear();
        st[0].insert(-INF); st[1].insert(-INF);
        sum[0] = sum[1] = 0; sum[_co]++;
    }
    inline void push_up(){
        if(this == null)return;
        if(ch[0] != null)lco = ch[0]->lco;
        else lco = co;
        if(ch[1] != null)rco = ch[1]->rco;
        else rco = co;
        sum[0] = ch[0]->sum[0] + ch[1]->sum[0] + (co == 0);
        sum[1] = ch[0]->sum[1] + ch[1]->sum[1] + (co == 1);
        int ml = max(w0,max(*st[co].rbegin(),co==ch[0]->rco?ch[0]->
           rs:-INF));
        int mr = max(w0,max(*st[co].rbegin(),co==ch[1]->lco?ch[1]->
           ls:-INF));
        ls = ch[0]->ls;
        if(lco == co && ch[0]->sum[!co] == 0)ls = max(ls,mr);
        rs = ch[1]->rs;
        if(rco == co && ch[1]->sum[!co] == 0)rs = max(rs,ml);
    }
    inline void setc(Node *p,int d){
        ch[d] = p;
        p->fa = this;
    }
    inline bool d(){
        return fa->ch[1] == this;
    }
    inline bool isroot(){
        return fa == null || fa->ch[0] != this && fa->ch[1] != this
           ;
    }
    inline void rot(){
        Node *f = fa, *ff = fa->fa;
        int c = d(), cc = fa->d();
        f->setc(ch[!c],c);
        this->setc(f,!c);
        if(ff->ch[cc] == f)ff->setc(this,cc);
        else this->fa = ff;
        f->push_up();
    }
    inline Node* splay(){
        while(!isroot()){
            if(!fa->isroot())
                d()==fa->d() ? fa->rot() : rot();
            rot();
        }
        push_up();
        return this;
    }
    inline Node* access(){
        for(Node *p = this,*q = null; p != null; q = p, p = p->fa){
            p->splay();
            if(p->ch[1] != null)
                p->st[p->ch[1]->lco].insert(p->ch[1]->ls);
            if(q != null)
                p->st[q->lco].erase(p->st[q->lco].find(q->ls));
            p->setc(q,1);
            p->push_up();
        }
        return splay();
    }
};
Node pool[MAXN],*tail;
Node *node[MAXN];
int color[MAXN],val[MAXN];
void init(int n){
    tail = pool;
    null = tail++;
    null->fa = null->ch[0] = null->ch[1] = null;
    null->st[0].clear(); null->st[1].clear();
    null->ls = null->rs = -INF;
    null->sum[0] = null->sum[1] = 0;
    null->co = null->lco = null->rco = 0;
    for(int i = 1;i <= n;i++){
        node[i] = tail++;
        node[i]->clear(color[i],val[i]);
    }
}
struct Edge{
    int to,next;
}edge[MAXN*2];
int head[MAXN],tot;
inline void addedge(int u,int v){
    edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;
}
void dfs(int u,int pre){
    for(int i = head[u];i != -1;i = edge[i].next){
        int v = edge[i].to;
        if(v == pre)continue;
        node[v]->fa = node[u];
        dfs(v,u);
        node[u]->st[node[v]->lco].insert(node[v]->ls);
    }
    node[u]->push_up();
}
int main()
{
    int n;
    while(scanf("%d",&n) == 1){
        for(int i = 1;i <= n;i++)head[i] = -1;
        tot = 0;
        int u,v;
        for(int i = 1;i < n;i++){
            scanf("%d%d",&u,&v);
            addedge(u,v);
            addedge(v,u);
        }
        for(int i = 1;i <= n;i++)scanf("%d",&color[i]);
        for(int i = 1;i <= n;i++)scanf("%d",&val[i]);
        init(n);
        dfs(1,1);
        int Q;
        int w,op;
        scanf("%d",&Q);
        while(Q--){
            scanf("%d",&op);
            if(op == 0){
                scanf("%d",&u);
                node[u]->access(); node[u]->splay();
                printf("%d\n",node[u]->rs);
            }
            else if(op == 1){
                scanf("%d",&u);
                node[u]->access(); node[u]->splay();
                node[u]->co ^= 1;
                node[u]->push_up();
            }
            else {
                scanf("%d%d",&u,&w);
                node[u]->access(); node[u]->splay();
                node[u]->w0 = w;
                node[u]->push_up();
            }
        }
    }
    return 0;
}
```

3.5.7 HDU4010

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

```
const int MAXN = 300010;
const int INF = 0x3f3f3f3f;
struct Node *null;
struct Node{
    Node *fa,*ch[2];
    int Max,val;
    int rev;//旋转标记
    int add;
    inline void clear(int _val){
        fa = ch[0] = ch[1] = null;
        val = Max = _val;
        rev = 0;
        add = 0;
    }
    inline void push_up(){
        Max = max(val,max(ch[0]->Max,ch[1]->Max));
    }
    inline void setc(Node *p,int d){
        ch[d] = p;
        p->fa = this;
    }
    inline bool d(){
        return fa->ch[1] == this;
    }
    inline bool isroot(){
        return fa == null || fa->ch[0] != this && fa->ch[1] != this
           ;
    }
    //翻转
    inline void flip(){
        if(this == null)return;
        swap(ch[0],ch[1]);
        rev ^= 1;
    }
    inline void update_add(int w){
        if(this == null)return;
        val += w;
        add += w;
        Max += w;
    }
    inline void push_down(){
        if(rev){
            ch[0]->flip(); ch[1]->flip(); rev = 0;
        }
        if(add){
            ch[0]->update_add(add); ch[1]->update_add(add);
            add = 0;
        }
    }
    //直接标记下放
    inline void go(){
        if(!isroot())fa->go();
        push_down();
    }
    inline void rot(){
        Node *f = fa, *ff = fa->fa;
        int c = d(), cc = fa->d();
        f->setc(ch[!c],c);
        this->setc(f,!c);
        if(ff->ch[cc] == f)ff->setc(this,cc);
        else this->fa = ff;
        f->push_up();
    }
    inline Node* splay(){
        go();
        while(!isroot()){
            if(!fa->isroot())
                d()==fa->d() ? fa->rot() : rot();
            rot();
        }
        push_up();
        return this;
    }
    inline Node* access(){
        for(Node *p = this,*q = null; p != null; q = p, p = p->fa){
            p->splay()->setc(q,1);
            p->push_up();
        }
        return splay();
    }
    //找该点的根
    inline Node* find_root(){
        Node *x;
        for(x = access(); x->push_down(), x->ch[0] != null; x = x->
           ch[0]);
        return x;
    }
    //变为树根 (换根操作)
    void make_root(){
        access()->flip();
    }
    //切断该点和父亲结点的边
    void cut(){
        access();
        ch[0]->fa = null;
        ch[0] = null;
        push_up();
    }
    //切断该点以 x 为根时，该点和父亲结点的根
    //要求这个点和 x 在同一颗树而且不能相同
    //x 变为所在树的树根
    void cut(Node* x){
        if(this == x || find_root() != x->find_root())
            puts("-1");
        else {
            x->make_root();
            cut();
        }
    }
    //该点连接到 x
    //假如是有虚边信息的，需要先 x->access() 再连接
    void link(Node *x){
        if(find_root() == x->find_root())
            puts("-1");
        else {
            make_root(); fa = x;
        }
    }
};
Node pool[MAXN],*tail;
Node *node[MAXN];
struct Edge{
    int to,next;
}edge[MAXN*2];
int head[MAXN],tot;
inline void addedge(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
void dfs(int u,int pre){
    for(int i = head[u];i != -1;i = edge[i].next){
        int v = edge[i].to;
        if(v == pre)continue;
        node[v]->fa = node[u];
        dfs(v,u);
    }
}
void ADD(Node *x,Node *y,int w){
    x->access();
    for(x = null; y != null; x = y, y = y->fa){
        y->splay();
        if(y->fa == null){
            y->ch[1]->update_add(w);
            x->update_add(w);
            y->val += w;
            y->push_up();
            return;
        }
        y->setc(x,1);
        y->push_up();
    }
}
int ask(Node *x,Node *y){
    x->access();
    for(x = null; y != null; x = y, y = y->fa){
        y->splay();
        if(y->fa == null)
            return max(y->val,max(y->ch[1]->Max,x->Max));
        y->setc(x,1);
        y->push_up();
    }
}
int main()
{
    int n;
    while(scanf("%d",&n) == 1){
        for(int i = 1;i <= n;i++)head[i] = -1;
        tot = 0;
        int u,v;
        for(int i = 1;i < n;i++){
            scanf("%d%d",&u,&v);
            addedge(u,v);
            addedge(v,u);
        }
        tail = pool;
        null = tail++;
        null->clear(-INF);
        for(int i = 1;i <= n;i++){
            node[i] = tail++;
            scanf("%d",&v);
            node[i]->clear(v);
        }
        dfs(1,1);
        int m,op;
        int x,y,w;
        scanf("%d",&m);
        while(m--){
            scanf("%d",&op);
            if(op == 1){
                scanf("%d%d",&x,&y);
                node[x]->link(node[y]);
            }
            else if(op == 2){
                scanf("%d%d",&x,&y);
                node[y]->cut(node[x]);
            }
            else if(op == 3){
                scanf("%d%d%d",&w,&x,&y);
                if(node[x]->find_root() != node[y]->find_root())
                    printf("-1\n");
                else ADD(node[x],node[y],w);
            }
            else{
                scanf("%d%d",&x,&y);
                if(node[x]->find_root() != node[y]->find_root())
                    printf("-1\n");
                else printf("%d\n",ask(node[x],node[y]));
            }
        }
        printf("\n");
    }
    return 0;
}
```

3.6 主席树

3.6.1 查询区间多少个不同的数

查询区间有多少个不同的数（SPOJ DQUERY）

```
/*
 * 给出一个序列，查询区间内有多少个不相同的数
 */
const int MAXN = 30010;
const int M = MAXN * 100;
int n,q,tot;
int a[MAXN];
int T[MAXN],lson[M],rson[M],c[M];
int build(int l,int r){
    int root = tot++;
    c[root] = 0;
    if(l != r){
        int mid = (l+r)>>1;
        lson[root] = build(l,mid);
        rson[root] = build(mid+1,r);
    }
    return root;
}
int update(int root,int pos,int val){
    int newroot = tot++, tmp = newroot;
    c[newroot] = c[root] + val;
    int l = 1, r = n;
    while(l < r){
        int mid = (l+r)>>1;
        if(pos <= mid){
            lson[newroot] = tot++; rson[newroot] = rson[root];
            newroot = lson[newroot]; root = lson[root];
            r = mid;
        }
        else{
            rson[newroot] = tot++; lson[newroot] = lson[root];
            newroot = rson[newroot]; root = rson[root];
            l = mid+1;
        }
        c[newroot] = c[root] + val;
    }
    return tmp;
}
int query(int root,int pos){
    int ret = 0;
    int l = 1, r = n;
    while(pos < r){
        int mid = (l+r)>>1;
        if(pos <= mid){
            r = mid;
            root = lson[root];
        }
        else{
            ret += c[lson[root]];
            root = rson[root];
            l = mid+1;
        }
    }
    return ret + c[root];
}
int main(){
    while(scanf("%d",&n) == 1){
        tot = 0;
        for(int i = 1;i <= n;i++)
            scanf("%d",&a[i]);
        T[n+1] = build(1,n);
        map<int,int>mp;
        for(int i = n;i>= 1;i--){
            if(mp.find(a[i]) == mp.end()){
                T[i] = update(T[i+1],i,1);
            }
            else{
                int tmp = update(T[i+1],mp[a[i]],-1);
                T[i] = update(tmp,i,1);
            }
            mp[a[i]] = i;
        }
        scanf("%d",&q);
        while(q--){
            int l,r;
            scanf("%d%d",&l,&r);
            printf("%d\n",query(T[l],r));
        }
    }
    return 0;
}
```

3.6.2 静态区间第 k 大

POJ2104

```
const int MAXN = 100010;
const int M = MAXN * 30;
int n,q,m,tot;
int a[MAXN], t[MAXN];
int T[MAXN], lson[M], rson[M], c[M];
void Init_hash(){
    for(int i = 1; i <= n;i++)
        t[i] = a[i];
    sort(t+1,t+1+n);
    m = unique(t+1,t+1+n)-t-1;
}
int build(int l,int r){
    int root = tot++;
    c[root] = 0;
    if(l != r){
        int mid = (l+r)>>1;
        lson[root] = build(l,mid);
        rson[root] = build(mid+1,r);
    }
    return root;
}
int hash(int x){
    return lower_bound(t+1,t+1+m,x) - t;
}
int update(int root,int pos,int val){
    int newroot = tot++, tmp = newroot;
    c[newroot] = c[root] + val;
    int l = 1, r = m;
    while(l < r){
        int mid = (l+r)>>1;
        if(pos <= mid){
            lson[newroot] = tot++; rson[newroot] = rson[root];
            newroot = lson[newroot]; root = lson[root];
            r = mid;
        }
        else{
            rson[newroot] = tot++; lson[newroot] = lson[root];
            newroot = rson[newroot]; root = rson[root];
            l = mid+1;
        }
        c[newroot] = c[root] + val;
    }
    return tmp;
}
int query(int left_root,int right_root,int k){
    int l = 1, r = m;
    while( l < r){
        int mid = (l+r)>>1;
        if(c[lson[left_root]]-c[lson[right_root]] >= k ){
            r = mid;
            left_root = lson[left_root];
            right_root = lson[right_root];
        }
        else{
            l = mid + 1;
            k -= c[lson[left_root]] - c[lson[right_root]];
            left_root = rson[left_root];
            right_root = rson[right_root];
        }
    }
    return l;
}
int main(){
    while(scanf("%d%d",&n,&q) == 2){
        tot = 0;
        for(int i = 1;i <= n;i++)
            scanf("%d",&a[i]);
        Init_hash();
        T[n+1] = build(1,m);
        for(int i = n;i ;i--){
            int pos = hash(a[i]);
            T[i] = update(T[i+1],pos,1);
        }
        while(q--){
            int l,r,k;
            scanf("%d%d%d",&l,&r,&k);
            printf("%d\n",t[query(T[l],T[r+1],k)]);
        }
    }
    return 0;
}
```

3.6.3 树上路径点权第 k 大

树上路径点权第 k 大（SPOJ COT）
LCA + 主席树

```
//主席树部分 *****************
const int MAXN = 200010;
const int M = MAXN * 40;
```

```
int n,q,m,TOT;
int a[MAXN], t[MAXN];
int T[MAXN], lson[M], rson[M], c[M];
void Init_hash(){
    for(int i = 1; i <= n;i++)
        t[i] = a[i];
    sort(t+1,t+1+n);
    m = unique(t+1,t+n+1)-t-1;
}
int build(int l,int r){
    int root = TOT++;
    c[root] = 0;
    if(l != r){
        int mid = (l+r)>>1;
        lson[root] = build(l,mid);
        rson[root] = build(mid+1,r);
    }
    return root;
}
int hash(int x){
    return lower_bound(t+1,t+1+m,x) - t;
}
int update(int root,int pos,int val){
    int newroot = TOT++, tmp = newroot;
    c[newroot] = c[root] + val;
    int l = 1, r = m;
    while( l < r){
        int mid = (l+r)>>1;
        if(pos <= mid){
            lson[newroot] = TOT++; rson[newroot] = rson[root];
            newroot = lson[newroot]; root = lson[root];
            r = mid;
        }
        else{
            rson[newroot] = TOT++; lson[newroot] = lson[root];
            newroot = rson[newroot]; root = rson[root];
            l = mid+1;
        }
        c[newroot] = c[root] + val;
    }
    return tmp;
}
int query(int left_root,int right_root,int LCA,int k){
    int lca_root = T[LCA];
    int pos = hash(a[LCA]);
    int l = 1, r = m;
    while(l < r){
        int mid = (l+r)>>1;
        int tmp = c[lson[left_root]] + c[lson[right_root]] - 2*c[
           lson[lca_root]] + (pos >= l && pos <= mid);
        if(tmp >= k){
            left_root = lson[left_root];
            right_root = lson[right_root];
            lca_root = lson[lca_root];
            r = mid;
        }
        else{
            k -= tmp;
            left_root = rson[left_root];
            right_root = rson[right_root];
            lca_root = rson[lca_root];
            l = mid + 1;
        }
    }
    return l;
}

//LCA 部分
int rmq[2*MAXN];//rmq 数组，就是欧拉序列对应的深度序列
struct ST{
    int mm[2*MAXN];
    int dp[2*MAXN][20];//最小值对应的下标
    void init(int n){
        mm[0] = -1;
        for(int i = 1;i <= n;i++){
            mm[i] = ((i&(i-1)) == 0)?mm[i-1]+1:mm[i-1];
            dp[i][0] = i;
        }
        for(int j = 1; j <= mm[n];j++)
            for(int i = 1; i + (1<<j) - 1 <= n; i++)
                dp[i][j] = rmq[dp[i][j-1]] < rmq[dp[i+(1<<(j-1))][j
                   -1]]?dp[i][j-1]:dp[i+(1<<(j-1))][j-1];
    }
    //查询 [a,b] 之间最小值的下标
    int query(int a,int b){
        if(a > b)swap(a,b);
        int k = mm[b-a+1];
        return rmq[dp[a][k]] <= rmq[dp[b-(1<<k)+1][k]]?dp[a][k]:dp[
           b-(1<<k)+1][k];
    }
};
//边的结构体定义
struct Edge{
    int to,next;
};
Edge edge[MAXN*2];
int tot,head[MAXN];

int F[MAXN*2];//欧拉序列，就是 dfs 遍历的顺序，长度为 2*n-1, 下标从 1 开始
int P[MAXN];//P[i] 表示点 i 在 F 中第一次出现的位置
int cnt;

```

```
ST st;
void init(){
    tot = 0;
    memset(head,-1,sizeof(head));
}
//加边，无向边需要加两次
void addedge(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
void dfs(int u,int pre,int dep){
    F[++cnt] = u;
    rmq[cnt] = dep;
    P[u] = cnt;
    for(int i = head[u];i != -1;i = edge[i].next){
        int v = edge[i].to;
        if(v == pre)continue;
        dfs(v,u,dep+1);
        F[++cnt] = u;
        rmq[cnt] = dep;
    }
}
//查询 LCA 前的初始化
void LCA_init(int root,int node_num){
    cnt = 0;
    dfs(root,root,0);
    st.init(2*node_num-1);
}
//查询 u,v 的 lca 编号
int query_lca(int u,int v){
    return F[st.query(P[u],P[v])];
}
void dfs_build(int u,int pre){
    int pos = hash(a[u]);
    T[u] = update(T[pre],pos,1);
    for(int i = head[u]; i != -1;i = edge[i].next){
        int v = edge[i].to;
        if(v == pre)continue;
        dfs_build(v,u);
    }
}
int main(){
    while(scanf("%d%d",&n,&q) == 2){
        for(int i = 1;i <= n;i++)
            scanf("%d",&a[i]);
        Init_hash();
        init();
        TOT = 0;
        int u,v;
        for(int i = 1;i < n;i++){
            scanf("%d%d",&u,&v);
            addedge(u,v);
            addedge(v,u);
        }
        LCA_init(1,n);
        T[n+1] = build(1,m);
        dfs_build(1,n+1);
        int k;
        while(q--){
            scanf("%d%d%d",&u,&v,&k);
            printf("%d\n",t[query(T[u],T[v],query_lca(u,v),k)]);
        }
        return 0;
    }
    return 0;
}
```

3.6.4 动态第 k 大

树状数组套主席树 ZOJ 2112

```
const int MAXN = 60010;
const int M = 2500010;
int n,q,m,tot;
int a[MAXN], t[MAXN];
int T[MAXN], lson[M], rson[M],c[M];
int S[MAXN];
struct Query{
    int kind;
    int l,r,k;
}query[10010];

void Init_hash(int k){
    sort(t,t+k);
    m = unique(t,t+k) - t;
}
int hash(int x){
    return lower_bound(t,t+m,x)-t;
}
int build(int l,int r){
    int root = tot++;
    c[root] = 0;
    if(l != r){
        int mid = (l+r)/2;
        lson[root] = build(l,mid);
        rson[root] = build(mid+1,r);
    }
    return root;
}

int Insert(int root,int pos,int val){
    int newroot = tot++, tmp = newroot;
    int l = 0, r = m-1;
    c[newroot] = c[root] + val;
    while(l < r){
        int mid = (l+r)>>1;
        if(pos <= mid){
            lson[newroot] = tot++; rson[newroot] = rson[root];
            newroot = lson[newroot]; root = lson[root];
            r = mid;
        }
        else{
            rson[newroot] = tot++; lson[newroot] = lson[root];
            newroot = rson[newroot]; root = rson[root];
            l = mid+1;
        }
        c[newroot] = c[root] + val;
    }
    return tmp;
}

int lowbit(int x){
    return x&(-x);
}
int use[MAXN];
void add(int x,int pos,int val){
    while(x <= n){
        S[x] = Insert(S[x],pos,val);
        x += lowbit(x);
    }
}
int sum(int x){
    int ret = 0;
    while(x > 0){
        ret += c[lson[use[x]]];
        x -= lowbit(x);
    }
    return ret;
}
int Query(int left,int right,int k){
    int left_root = T[left-1];
    int right_root = T[right];
    int l = 0, r = m-1;
    for(int i = left-1;i;i -= lowbit(i)) use[i] = S[i];
    for(int i = right;i ;i -= lowbit(i)) use[i] = S[i];
    while(l < r){
        int mid = (l+r)/2;
        int tmp = sum(right) - sum(left-1) + c[lson[right_root]] -
           c[lson[left_root]];
        if(tmp >= k){
            r = mid;
            for(int i = left-1; i ;i -= lowbit(i))
                use[i] = lson[use[i]];
            for(int i = right; i; i -= lowbit(i))
                use[i] = lson[use[i]];
            left_root = lson[left_root];
            right_root = lson[right_root];
        }
        else{
            l = mid+1;
            k -= tmp;
            for(int i = left-1; i;i -= lowbit(i))
                use[i] = rson[use[i]];
            for(int i = right;i ;i -= lowbit(i))
                use[i] = rson[use[i]];
            left_root = rson[left_root];
            right_root = rson[right_root];
        }
    }
    return l;
}
void Modify(int x,int p,int d){
    while(x <= n){
        S[x] = Insert(S[x],p,d);
        x += lowbit(x);
    }
}

int main(){
    int Tcase;
    scanf("%d",&Tcase);
    while(Tcase--){
        scanf("%d%d",&n,&q);
        tot = 0;
        m = 0;
        for(int i = 1;i <= n;i++){
            scanf("%d",&a[i]);
            t[m++] = a[i];
        }
        char op[10];
        for(int i = 0;i < q;i++){
            scanf("%s",op);
            if(op[0] == 'Q'){
                query[i].kind = 0;
                scanf("%d%d%d",&query[i].l,&query[i].r,&query[i].k)
                   ;
            }
            else{
                query[i].kind = 1;
                scanf("%d%d",&query[i].l,&query[i].r);
                t[m++] = query[i].r;
            }
        }
        Init_hash(m);
        T[0] = build(0,m-1);
        for(int i = 1;i <= n;i++)
            T[i] = Insert(T[i-1],hash(a[i]),1);
        for(int i = 1;i <= n;i++)
            S[i] = T[0];
        for(int i = 0;i < q;i++){
            if(query[i].kind == 0)
                printf("%d\n",t[Query(query[i].l,query[i].r,query[i
                   ].k)]);
            else{
                Modify(query[i].l,hash(a[query[i].l]),-1);
                Modify(query[i].l,hash(query[i].r),1);
                a[query[i].l] = query[i].r;
            }
        }
    }
    return 0;
}
```

3.7 Treap

ZOJ3765

```
long long gcd(long long a,long long b){
    if(b == 0)return a;
    else return gcd(b,a%b);
}
const int MAXN = 300010;
int num[MAXN],st[MAXN];
struct Treap{
    int tot1;
    int s[MAXN],tot2;//内存池和容量
    int ch[MAXN][2];
    int key[MAXN],size[MAXN];
    int sum0[MAXN],sum1[MAXN];
    int status[MAXN];
    void Init(){
        tot1 = tot2 = 0;
        size[0] = 0;
        ch[0][0] = ch[0][1] = 0;
        sum0[0] = sum1[0] = 0;
    }
    bool random(double p){
        return (double)rand() / RAND_MAX < p;
    }
    int newnode(int val,int _status){
        int r;
        if(tot2)r = s[tot2--];
        else r = ++tot1;
        size[r] = 1;
        key[r] = val;
        status[r] = _status;
        ch[r][0] = ch[r][1] = 0;
        sum0[r] = sum1[r] = 0;//需要push_up
        return r;
    }
    void del(int r){
        if(!r)return;
        s[++tot2] = r;
        del(ch[r][0]);
        del(ch[r][1]);
    }
    void push_up(int r){
        int lson = ch[r][0], rson = ch[r][1];
        size[r] = size[lson] + size[rson] + 1;
        sum0[r] = gcd(sum0[lson],sum0[rson]);
        sum1[r] = gcd(sum1[lson],sum1[rson]);
        if(status[r] == 0)
            sum0[r] = gcd(sum0[r],key[r]);
        else sum1[r] = gcd(sum1[r],key[r]);
    }
    void merge(int &p,int x,int y){
        if(!x || !y)
            p = x|y;
        else if(random((double)size[x]/(size[x]+size[y]))){
            merge(ch[x][1],ch[x][1],y);
            push_up(p=x);
        }
        else{
            merge(ch[y][0],x,ch[y][0]);
            push_up(p=y);
        }
    }
    void split(int p,int &x,int &y,int k){
        if(!k){
            x = 0; y = p;
            return;
        }
        if(size[ch[p][0]] >= k){
            y = p;
            split(ch[p][0],x,ch[y][0],k);
            push_up(y);
        }
        else{
            x = p;
            split(ch[p][1],ch[x][1],y,k - size[ch[p][0]] - 1);
            push_up(x);
        }
    }
    void build(int &p,int l,int r){
        if(l > r)return;
        int mid = (l + r)/2;
        p = newnode(num[mid],st[mid]);
```

```cpp
        build(ch[p][0],l,mid-1);
        build(ch[p][1],mid+1,r);
        push_up(p);
    }
    void debug(int root){
        if(root == 0)return;
        printf("%d␣␣␣左儿子：%d␣␣右儿子:␣%d␣␣size␣=␣%d␣␣key␣=␣%d\n",
           root,ch[root][0],ch[root][1],size[root],key[root]);
        debug(ch[root][0]);
        debug(ch[root][1]);
    }
};
Treap T;
char op[10];
int main(){
    int n,q;
    while(scanf("%d%d",&n,&q) == 2){
        int root = 0;
        T.Init();
        for(int i = 1;i <= n;i++)
            scanf("%d%d",&num[i],&st[i]);
        T.build(root,1,n);
        while(q--){
            scanf("%s",op);
            if(op[0] == 'Q'){
                int l,r,s;
                scanf("%d%d%d",&l,&r,&s);
                int x,y,z;
                T.split(root,x,z,r);
                T.split(x,x,y,l-1);
                if(s == 0)
                    printf("%d\n",T.sum0[y] == 0? -1:T.sum0[y]);
                else
                    printf("%d\n",T.sum1[y] == 0?-1:T.sum1[y]);
                T.merge(x,x,y);
                T.merge(root,x,z);
            }
            else if(op[0] == 'I'){
                int v,s,loc;
                scanf("%d%d%d",&loc,&v,&s);
                int x,y;
                T.split(root,x,y,loc);
                T.merge(x,x,T.newnode(v,s));
                T.merge(root,x,y);
            }
            else if(op[0] == 'D'){
                int loc;
                scanf("%d",&loc);
                int x,y,z;
                T.split(root,x,z,loc);
                T.split(x,x,y,loc-1);
                T.del(y);
                T.merge(root,x,z);
            }
            else if(op[0] == 'R'){
                int loc;
                scanf("%d",&loc);
                int x,y,z;
                T.split(root,x,z,loc);
                T.split(x,x,y,loc-1);
                T.status[y] = 1-T.status[y];
                T.push_up(y);
                T.merge(x,x,y);
                T.merge(root,x,z);
            }
            else{
                int loc,v;
                scanf("%d%d",&loc,&v);
                int x,y,z;
                T.split(root,x,z,loc);
                T.split(x,x,y,loc-1);
                T.key[y] = v;
                T.push_up(y);
                T.merge(x,x,y);
                T.merge(root,x,z);
            }
        }
    }
    return 0;
}
```

3.8 KD 树

3.8.1 HDU4347 K 近邻

模板题，求出最近的 K 个点。

```
const int MAXN = 50010;
const int DIM = 10;
inline double sqr(double x){return x*x;}
namespace KDTree{
    int K;//维数
    struct Point{
        int x[DIM];
        double distance(const Point &b)const{
            double ret = 0;
            for(int i = 0;i < K;i++)
                ret += sqr(x[i]-b.x[i]);
            return ret;
        }
        void input(){
            for(int i = 0;i < K;i++)scanf("%d",&x[i]);
        }
        void output(){
            for(int i = 0;i < K;i++)
                printf("%d%c",x[i],i < K-1?'␣':'\n');
        }
    };
    struct qnode{
        Point p;
        double dis;
        qnode(){}
        qnode(Point _p,double _dis){
            p = _p; dis = _dis;
        }
        bool operator <(const qnode &b)const{
            return dis < b.dis;
        }
    };
    priority_queue<qnode>q;
    struct cmpx{
        int div;
        cmpx(const int &_div){div = _div;}
        bool operator()(const Point &a,const Point &b){
            for(int i = 0;i < K;i++)
                if(a.x[(div+i)%K] != b.x[(div+i)%K])
                    return a.x[(div+i)%K] < b.x[(div+i)%K];
            return true;
        }
    };
    bool cmp(const Point &a,const Point &b,int div){
        cmpx cp = cmpx(div);
        return cp(a,b);
    }
    struct Node{
        Point e;
        Node *lc,*rc;
        int div;
    }pool[MAXN],*tail,*root;
    void init(){
        tail = pool;
    }
    Node* build(Point *a,int l,int r,int div){
        if(l >= r)return NULL;
        Node *p = tail++;
        p->div = div;
        int mid = (l+r)/2;
        nth_element(a+l,a+mid,a+r,cmpx(div));
        p->e = a[mid];
        p->lc = build(a,l,mid,(div+1)%K);
        p->rc = build(a,mid+1,r,(div+1)%K);
        return p;
    }
    void search(Point p,Node *x,int div,int m){
        if(!x)return;
        if(cmp(p,x->e,div)){
            search(p,x->lc,(div+1)%K,m);
            if(q.size() < m){
                q.push(qnode(x->e,p.distance(x->e)));
                search(p,x->rc,(div+1)%K,m);
            }
            else {
                if(p.distance(x->e) < q.top().dis){
                    q.pop();
                    q.push(qnode(x->e,p.distance(x->e)));
                }
                if(sqr(x->e.x[div]-p.x[div]) < q.top().dis)
                    search(p,x->rc,(div+1)%K,m);
            }
        }
        else {
            search(p,x->rc,(div+1)%K,m);
            if(q.size() < m){
                q.push(qnode(x->e,p.distance(x->e)));
                search(p,x->lc,(div+1)%K,m);
            }
            else {
                if(p.distance(x->e) < q.top().dis){
                    q.pop();
                    q.push(qnode(x->e,p.distance(x->e)));
                }
                if(sqr(x->e.x[div]-p.x[div]) < q.top().dis)
                    search(p,x->lc,(div+1)%K,m);
            }
        }
    }
    void search(Point p,int m){
        while(!q.empty())q.pop();
        search(p,root,0,m);
    }
};
KDTree::Point p[MAXN];
int main()
{
    int n,k;
    while(scanf("%d%d",&n,&k) == 2){
        KDTree::K = k;
        for(int i = 0;i < n;i++)p[i].input();
        KDTree::init();
        KDTree::root = KDTree::build(p,0,n,0);
        int Q;
        scanf("%d",&Q);
        KDTree::Point o;
        while(Q--){
            o.input();
            int m;
            scanf("%d",&m);
            KDTree::search(o,m);
            printf("the␣closest␣%d␣points␣are:\n",m);
            int cnt = 0;
            while(!KDTree::q.empty()){
                p[cnt++] = KDTree::q.top().p;
                KDTree::q.pop();
            }
            for(int i = 0;i < m;i++)p[m−1−i].output();
        }
    }
    return 0;
}
```

3.8.2 CF44G

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

```
const int MAXN = 100010;
const int INF = 0x3f3f3f3f;
struct Point{
    int x,y,id;
    bool operator ==(const Point &b)const{
        return x == b.x && y == b.y && id == b.id;
    }
};
struct Node{
    Point e;
    Node *lc,*rc;
    bool div;
    int sub,cur;
    int size;
    bool exist;
    void push_up(){
        size = lc−>size + rc−>size + exist;
        sub = min(cur,min(lc−>sub,rc−>sub));
    }
}pool[MAXN],*root,*tail,*null;
inline bool cmpX(const Point &a,const Point &b){return a.x < b.x ||
    (a.x == b.x && a.y < b.y) || (a.x == b.x && a.y == b.y && a.id
   < b.id);}
inline bool cmpY(const Point &a,const Point &b){return a.y < b.y ||
    (a.y == b.y && a.x < b.x) || (a.y == b.y && a.x == b.x && a.id
   < b.id);}
inline bool cmp(const Point &a,const Point &b,bool div){return div?
   cmpY(a,b):cmpX(a,b);}
Node* build(Point *a,int l,int r,bool div){
    if(l >= r)return null;
    Node *p = tail++;
    p->div = div;
    int mid = (l+r)/2;
    nth_element(a+l,a+mid,a+r,div?cmpY:cmpX);
    p->e = a[mid];
    p->lc = build(a,l,mid,!div);
    p->rc = build(a,mid+1,r,!div);
    p->exist = 1;
    p->cur = p->e.id;
    p->push_up();
    return p;
}
void remove(Node *p,Point o){
    if(p->e == o){
        p->exist = 0;
        p->cur = INF;
        p->size--;
    }
    else {
        if(cmp(p->e,o,p->div))remove(p->rc,o);
        else remove(p->lc,o);
    }
    p->push_up();
}
int getMin(Node *p,int xl,int xr,int yl,int yr,int minx,int maxx,
   int miny,int maxy){
    if(p == null || p->size == 0)return INF;
    if(xl <= minx && xr >= maxx && yl <= miny && yr >= maxy)return
       p->sub;
    if(xl > maxx || xr < minx || yl > maxy || yr < miny)return INF;
    int ret = INF;
    if(p->e.x >= xl && p->e.x <= xr && p->e.y >= yl && p->e.y <= yr
       )
        ret = min(ret,p->cur);
    if(p->div){
        if(yl <= p->e.y)
            ret = min(ret,getMin(p->lc,xl,xr,yl,min(yr,p->e.y),minx
               ,maxx,miny,min(maxy,p->e.y)));
        if(yr >= p->e.y)
            ret = min(ret,getMin(p->rc,xl,xr,max(yl,p->e.y),yr,minx
               ,maxx,max(miny,p->e.y),maxy));
    }
    else {
        if(xl <= p->e.x)
            ret = min(ret,getMin(p->lc,xl,min(xr,p->e.x),yl,yr,minx
               ,min(maxx,p->e.x),miny,maxy));
        if(xr >= p->e.x)
            ret = min(ret,getMin(p->rc,max(xl,p->e.x),xr,yl,yr,max(
               minx,p->e.x),maxx,miny,maxy));
    }
    return ret;
}
Point pp[MAXN],pp2[MAXN];
struct REC{
    int xl,xr,yl,yr,z;
    int id;
    void input(){
        scanf("%d%d%d%d%d",&xl,&xr,&yl,&yr,&z);
    }
    bool operator <(const REC &b)const{
        return z < b.z;
    }
}rec[MAXN];
int ans[MAXN];
int main()
{
    int n,m;
    while(scanf("%d",&n) == 1){
        for(int i = 0;i < n;i++){
            rec[i].input();
            rec[i].id = i+1;
        }
        sort(rec,rec+n);
        scanf("%d",&m);
        for(int i = 0;i < m;i++){
            scanf("%d%d",&pp[i].x,&pp[i].y);
            pp[i].id = i;
            pp2[i] = pp[i];//备份
        }
        tail = pool;
        null = tail++;
        null->size = 0;
        null->sub = null->cur = INF;
        null->lc = null->rc = null;
        root = build(pp,0,m,0);
        memset(ans,0,sizeof(ans));
        for(int i = 0;i < n;i++){
            int tmp = getMin(root,rec[i].xl,rec[i].xr,rec[i].yl,rec
               [i].yr,-INF,INF,-INF,INF);
            if(tmp == INF)continue;
            ans[tmp] = rec[i].id;
            remove(root,pp2[tmp]);
        }
        for(int i = 0;i < m;i++)printf("%d\n",ans[i]);
    }
    return 0;
}
```

3.8.3 HDU4742

三维 LIS。即每个点有个三维坐标，两个点能放在一前一后当且仅当 $xi < xj, yi < yj, zi < zj$，求最长的序列，并该条件下的方案数。

```
const int MAXN = 100010;
const int MOD = 1<<30;
const int INF = 0x7fffffff;//这个一定要够大
struct Node{
    pair<int,int>e,sub,cur;
    bool div;
    Node *lc,*rc;
};
Node pool[MAXN],*tail;
Node *root;
bool cmpX(const pair<int,int> &a,const pair<int,int> &b){return a.first < b.first || (a.first ==  b.first && a.second < b.second);}
bool cmpY(const pair<int,int> &a,const pair<int,int> &b){return a.second < b.second || (a.second == b.second && a.first < b.first);}
bool cmp(const pair<int,int> &a,const pair<int,int> &b,bool div){return div?cmpY(a,b):cmpX(a,b);}
Node* build(pair<int,int> *a,int l,int r,bool div){
    if(l >= r)return NULL;
    Node *p = tail++;
    p->div = div;
    int mid = (l+r)/2;
    nth_element(a+l,a+mid,a+r,div?cmpY:cmpX);
    p->e = a[mid];
    p->cur = p->sub = make_pair(0,0);
    p->lc = build(a,l,mid,!div);
    p->rc = build(a,mid+1,r,!div);
    return p;
}
inline void update(pair<int,int> &a,pair<int,int> b){
    if(a.first < b.first)a = b;
    else if(a.first == b.first){
        a.second += b.second;
        if(a.second >= MOD)a.second -= MOD;
    }
}
void add(Node *p,pair<int,int> e,pair<int,int> v){
    update(p->sub,v);
    if(e == p->e){
        update(p->cur,v);
        return;
    }
    else {
        if(cmp(p->e,e,p->div))add(p->rc,e,v);
        else add(p->lc,e,v);
    }
```

```
}
pair<int,int>ans;
//查询最大值
void get(Node *p,pair<int,int>e,int maxx,int maxy){
    if(!p)return;
    if(p->sub.first < ans.first)return;
    if(maxx <= e.first && maxy <= e.second)
        update(ans,p->sub);
    else {
        if(p->e.first <= e.first && p->e.second <= e.second)update(
           ans,p->cur);
        if(p->div){
            if(p->e.second <= e.second)get(p->rc,e,maxx,maxy);
            get(p->lc,e,maxx,min(maxy,p->e.second));
        }
        else {
            if(p->e.first <= e.first)get(p->rc,e,maxx,maxy);
            get(p->lc,e,min(maxx,p->e.first),maxy);
        }
    }
}
struct TNode{
    int x,y,z;
    void input(){
        scanf("%d%d%d",&x,&y,&z);
    }
    bool operator < (const TNode &b)const{
        if(x != b.x)return x < b.x;
        else if(y != b.y)return y < b.y;
        else return z < b.z;
    }
}node[MAXN];
pair<int,int>p[MAXN];
pair<int,int>dp[MAXN];
int main()
{
    int T;
    int n;
    scanf("%d",&T);
    while(T--){
        scanf("%d",&n);
        int cnt = 0;
        for(int i = 0;i < n;i++){
            node[i].input();
            p[cnt++] = make_pair(node[i].y,node[i].z);
        }
        sort(node,node+n);
        sort(p,p+cnt);
        cnt = unique(p,p+cnt)-p;
        tail = pool;
        root = build(p,0,cnt,0);
        for(int i = 0;i < n;i++)dp[i] = make_pair(1,1);
        for(int i = 0;i < n;i++){
            ans = make_pair(0,0);
            get(root,make_pair(node[i].y,node[i].z),INF,INF);
            ans.first++;
            update(dp[i],ans);
            add(root,make_pair(node[i].y,node[i].z),dp[i]);
        }
        printf("%d␣%d\n",root->sub.first,root->sub.second);
    }
    return 0;
}
```

3.9 替罪羊树 (ScapeGoat Tree)

3.9.1 CF455D

http://codeforces.com/contest/455/problem/D
题意：给了一个序列，1 操作把一个区间的末尾的数插入到头部，2 操作是询问一个区间里面等于某个数的个数。
使用替罪羊树，里面套一个 map 来统计区间的个数。

```
const int MAXN = 200010;
const double alpha = 0.75;
struct Node{
    Node *ch[2];
    int size,key,nodeCount;
    bool exist;
    map<int,int>mp;
    bool isBad(){
        return ch[0]->nodeCount > alpha*nodeCount+5 || ch[1]->
           nodeCount > alpha*nodeCount + 5;
    }
    void push_up(){
        size = exist + ch[0]->size + ch[1]->size;
        nodeCount = 1 + ch[0]->nodeCount + ch[1]->nodeCount;
        mp.clear();
        if(exist)mp[key]++;
        for(map<int,int>::iterator it = ch[0]->mp.begin();it != ch
           [0]->mp.end();it++)
            mp[(*it).first] += (*it).second;
        for(map<int,int>::iterator it = ch[1]->mp.begin();it != ch
           [1]->mp.end();it++)
            mp[(*it).first] += (*it).second;
    }
};
struct ScapeGoatTree{
    Node pool[MAXN];
    Node *tail,*root,*null;
    Node *bc[MAXN];//内存回收
    int bc_top;
    void init(){
        tail = pool;
        null = tail++;
        null->ch[0] = null->ch[1] = null;
        null->size = null->key = null->nodeCount = 0;
        null->mp.clear();
        root = null;
        bc_top = 0;
    }
    inline Node *newNode(int key){
        Node *p;
        if(bc_top)p = bc[--bc_top];
        else p = tail++;
        p->ch[0] = p->ch[1] = null;
        p->size = p->nodeCount = 1;
        p->key = key;
        p->exist = true;
        p->mp.clear();
        p->mp[key] = 1;
        return p;
    }
    Node *buildTree(int *a,int l,int r){
        if(l >= r)return null;
        int mid = (l+r)>>1;
        Node *p = newNode(a[mid]);
        p->ch[0] = buildTree(a,l,mid);
        p->ch[1] = buildTree(a,mid+1,r);
        p->push_up();
        return p;
    }
    inline void Travel(Node *p,vector<Node *>&v){
        if(p == null)return;
        Travel(p->ch[0],v);
        if(p->exist)v.push_back(p);
        else bc[bc_top++] = p;
        Travel(p->ch[1],v);
    }
    inline Node *divide(vector<Node *>&v,int l,int r){
        if(l >= r)return null;
        int mid = (l+r)/2;
        Node *p = v[mid];
        p->ch[0] = divide(v,l,mid);
        p->ch[1] = divide(v,mid+1,r);
        p->push_up();
        return p;
    }
    //重构，注意 p 要引用
    inline void rebuild(Node *&p){
        vector<Node *>v;
        Travel(p,v);
        p = divide(v,0,v.size());
    }
    //删除第 id 个元素，返回第 id 个元素的值
    inline int erase(Node *p,int id){
        if(p->exist && id == p->ch[0]->size + 1){
            p->exist = 0;
            p->mp[p->key]--;
            p->size--;
            return p->key;
        }
        p->size--;
        int res;
        if(p->ch[0]->size >= id)
            res = erase(p->ch[0],id);
        else res = erase(p->ch[1],id - p->ch[0]->size - p->exist);
        p->mp[res]--;
        return res;
    }
    //删除一定的点以后重构
    void check_erase(){
        if(root->size < 0.5*root->nodeCount)
            rebuild(root);
    }
    Node **insert(Node *&p,int id,int val){
        if(p == null){
            p = newNode(val);
            return &null;
        }
        else {
            p->size++;
            p->nodeCount++;
            p->mp[val]++;
            Node ** res;
            if(id <= p->ch[0]->size+p->exist)
                res = insert(p->ch[0],id,val);
            else res = insert(p->ch[1],id-p->ch[0]->size-p->exist,
               val);
            if(p->isBad())res = &p;
            return res;
        }
    }
    //在第 id 个位置插入数 val
    void insert(int id,int val){
        Node **p = insert(root,id,val);
        if(*p != null)rebuild(*p);
    }
    //查询 [l,r] 之间值为 val 的数的个数
    int query(Node *p,int l,int r,int val){
        if(p == null)return 0;
        if(l <= 1 && p->size <= r)
            return p->mp.count(val)?p->mp[val]:0;
        else {
            int ans = 0;
            if(l <= p->ch[0]->size)
                ans += query(p->ch[0],l,r,val);
            if(r > p->ch[0]->size+p->exist)
                ans += query(p->ch[1],l - p->ch[0]->size - p->exist
                    , r - p->ch[0]->size - p->exist,val);
            if(p->exist && p->key == val && l <= p->ch[0]->size+1
               && r >= p->ch[0]->size+1)
                ans++;
            return ans;
        }
    }
}tree;
int a[MAXN];
int main()
{
    int n;
    while(scanf("%d",&n) == 1){
        tree.init();
        for(int i = 0;i < n;i++)scanf("%d",&a[i]);
        tree.root = tree.buildTree(a,0,n);
        int  m;
        int op,l,r,k;
        scanf("%d",&m);
        int ans = 0;
        while(m--){
            scanf("%d",&op);
            if(op == 1){
                scanf("%d%d",&l,&r);
                l = ((l+ans-1)%n)+1;
                r = ((r+ans-1)%n)+1;
                if(l > r)swap(l,r);
                int v = tree.erase(tree.root,r);
                //tree.check_erase(); //有时候可以加上删除重构
                tree.insert(l,v);
            }
            else {
                scanf("%d%d%d",&l,&r,&k);
                l = ((l+ans-1)%n)+1;
                r = ((r+ans-1)%n)+1;
                k = ((k+ans-1)%n)+1;
                if(l > r)swap(l,r);
                ans = tree.query(tree.root,l,r,k);
                printf("%d\n",ans);
            }
        }
    }
    return 0;
}
```

3.10 动态 KD 树

动态 KD 树就是结合了 KD 树和替罪羊树。支持 KD 树的插入删除操作，用替罪羊树的思想来保存平衡。
UVALive6045
题意：给了二维平面上的 N 个整点（$N \le 50000$）。每次操作给了点 (x_i, y_i), 需要曼哈顿距离小于 E 的点进行一个变换。输出最后的点的坐标，保证变换次数不超过 50000.

```
const int MAXN = 100010;
const double alpha = 0.75;
struct Point{
    int x,y,id;
};
struct Node{
    Point e;
    int size,nodeCount;
    Node *lc,*rc;
    bool div;
    bool exist;
    bool isBad(){
        return lc->nodeCount > alpha*nodeCount+5 || rc->nodeCount >
           alpha*nodeCount+5;
    }
    inline void push_up(){
        size = exist + lc->size + rc->size;
        nodeCount = 1+lc->nodeCount+rc->nodeCount;
    }
};
Node pool[MAXN],*tail,*root,*null;
Node *bc[MAXN];
int bc_top;
void init(){
    tail = pool;
    null = tail++;
    null->lc = null->rc = null;
    null->size = null->nodeCount = 0;
    root = null;
    bc_top = 0;
}
Node *newNode(Point e){
    Node *p;
    if(bc_top)p = bc[--bc_top];
    else p = tail++;
    p->e = e;
    p->lc = p->rc = null;
    p->size = p->nodeCount = 1;
    p->exist = true;
    return p;
}
inline bool cmpX(const Point &a,const Point &b){
    return a.x < b.x || (a.x == b.x && a.y < b.y) || (a.x == b.x &&
        a.y == b.y && a.id < b.id);
```

```
}
inline bool cmpY(const Point &a,const Point &b){
    return a.y < b.y || (a.y == b.y && a.x < b.x) || (a.y == b.y &&
        a.x == b.x && a.id < b.id);
}
inline bool cmp(const Point &a,const Point &b,bool div){
    return div?cmpY(a,b):cmpX(a,b);
}
//注意 a 需要备份，否则就乱序
Node *build(Point *a,int l,int r,bool div){
    if(l >= r)return null;
    int mid = (l+r)/2;
    nth_element(a+l,a+mid,a+r,div?cmpY:cmpX);
    Node *p = newNode(a[mid]);
    p->div = div;
    p->lc = build(a,l,mid,!div);
    p->rc = build(a,mid+1,r,!div);
    p->push_up();
    return p;
}
void Travel(Node *p,vector<Point>&v){
    if(p == null)return;
    Travel(p->lc,v);
    if(p->exist)v.push_back(p->e);
    bc[bc_top++] = p;
    Travel(p->rc,v);
}
Node *divide(vector<Point>&v,int l,int r,bool div){
    if(l >= r)return null;
    int mid = (l+r)/2;
    nth_element(v.begin()+l,v.begin()+mid,v.begin()+r,div?cmpY:cmpX
       );
    Node *p = newNode(v[mid]);
    p->div = div;
    p->lc = divide(v,l,mid,!div);
    p->rc = divide(v,mid+1,r,!div);
    p->push_up();
    return p;
}
inline void rebuild(Node *&p){
    vector<Point>v;
    Travel(p,v);
    p = divide(v,0,v.size(),p->div);
}
Node **insert(Node *&p,Point a,bool div){
    if(p == null){
        p = newNode(a);
        p->div = div;
        return &null;
    }
    else {
        p->nodeCount++;
        p->size++;
        Node **res;
        if(cmp(a,p->e,div))
            res = insert(p->lc,a,!div);
        else res = insert(p->rc,a,!div);
        if(p->isBad())res = &p;
        return res;
    }
}
void insert(Point e){
    Node **p = insert(root,e,0);
    if(*p != null)rebuild(*p);
}
vector<int>vec;
void getvec(Node *p,int minx,int maxx,int miny,int maxy){
    if(p->size == 0)return;
    if(p->exist && minx <= p->e.x && p->e.x <= maxx && miny <= p->e
       .y && p->e.y <= maxy){
        vec.push_back(p->e.id);
        p->exist = 0;
        p->size--;
    }
    if(p->div? miny <= p->e.y : minx <= p->e.x)getvec(p->lc,minx,
       maxx,miny,maxy);
    if(p->div? maxy >= p->e.y : maxx >= p->e.x)getvec(p->rc,minx,
       maxx,miny,maxy);
    p->push_up();
}
Point p[MAXN],p2[MAXN];
Point p3[MAXN];
int main()
{
    int T;
    scanf("%d",&T);
    int iCase = 0;
    int N,Q,W,H;
    while(T--){
        iCase++;
        scanf("%d%d%d%d",&N,&Q,&W,&H);
        init();
        for(int i = 0;i < N;i++){
            scanf("%d%d",&p[i].x,&p[i].y);
            p[i].id = p2[i].id = i;
            p2[i].x = p[i].x+p[i].y;
            p2[i].y = p[i].x-p[i].y;
            p3[i] = p2[i];
        }
        root = build(p3,0,N,0);
        int X,Y,E,a,b,c,d,e,f;
        while(Q--){
            scanf("%d%d%d%d%d%d%d%d%d",&X,&Y,&E,&a,&b,&c,&d,&e,&f);
            vec.clear();
            int minx = X+Y-E;
            int maxx = X+Y+E;
            int miny = X-Y-E;
            int maxy = X-Y+E;
            getvec(root,minx,maxx,miny,maxy);
            int sz = vec.size();
            for(int i = 0;i < sz;i++){
                int id = vec[i];
                long long tx = p[id].x;
                long long ty = p[id].y;
                p[id].x = (tx*a+ty*b+(long long)(id+1)*c)%W;
                p[id].y = (tx*d+ty*e+(long long)(id+1)*f)%H;
                p2[id].x = p[id].x+p[id].y;
                p2[id].y = p[id].x-p[id].y;
                insert(p2[id]);
            }
        }
        printf("Case␣#%d:\n",iCase);
        for(int i = 0;i < N;i++)
            printf("%d␣%d\n",p[i].x,p[i].y);
    }
    return 0;
}
```

3.11 树套树

3.11.1 替罪羊树套 splay

BZOJ 3065: 带插入区间 K 小值
带插入、修改的区间 k 小值在线查询。
1. Q x y k: 询问从左至右第 x 只跳蚤到从左至右第 y 只跳蚤中，弹跳力第 k 小的跳蚤的弹跳力是多少。$(1 <= x <= y <= m, 1 <= k <= y - x + 1)$
2. M x val: 将从左至右第 x 只跳蚤的弹跳力改为 val。$(1 <= x <= m)$
3. I x val: 在从左至右第 x 只跳蚤的前面插入一只弹跳力为 val 的跳蚤。即插入后从左至右第 x 只跳蚤是我刚插入的跳蚤。$(1 <= x <= m + 1)$

```
const int MAXN = 70010;
namespace Splay{
    struct Node *null;
    struct Node{
        Node *ch[2],*fa;
        int size,key,cnt;
        inline void setc(Node *p,int d){
            ch[d] = p;
            p->fa = this;
        }
        inline bool d(){
            return fa->ch[1] == this;
        }
        inline void push_up(){
            size = ch[0]->size + ch[1]->size + cnt;
        }
        void clear(int _key){
            size = cnt = 1;
            key = _key;
            ch[0] = ch[1] = fa = null;
        }
        inline bool isroot(){
            return fa == null || this != fa->ch[0] && this != fa->
               ch[1];
        }
    };
    Node pool[MAXN*20],*tail;
    Node *bc[MAXN*20];
    int bc_top;//内存回收
    void init(){
        tail = pool;
        bc_top = 0;
        null = tail++;
        null->size = null->cnt = 0;
        null->ch[0] = null->ch[1] = null->fa = null;
    }
    inline void rotate(Node *x){
        Node *f = x->fa, *ff = x->fa->fa;
        int c = x->d(), cc = f->d();
        f->setc(x->ch[!c],c);
        x->setc(f,!c);
        if(ff->ch[cc] == f)ff->setc(x,cc);
        else x->fa = ff;
        f->push_up();
    }
    inline void splay(Node* &root,Node* x,Node* goal){
        while(x->fa != goal){
            if(x->fa->fa == goal)rotate(x);
            else {
                bool f = x->fa->d();
                x->d() == f ? rotate(x->fa) : rotate(x);
                rotate(x);
            }
        }
        x->push_up();
        if(goal == null)root = x;
    }
    //找到 r 子树里面的最左边那个
    Node* get_left(Node* r){
        Node* x = r;
        while(x->ch[0] != null)x = x->ch[0];
        return x;
    }
    //在 root 的树中删掉 x
    void erase(Node* &root,Node* x){
        splay(root,x,null);
        Node* t = root;
        if(t->ch[1] != null){
            root = t->ch[1];
            splay(root,get_left(t->ch[1]),null);
            root->setc(t->ch[0],0);
        }
        else root = root->ch[0];
        bc[bc_top++] = x;
        root->fa = null;
        if(root != null)root->push_up();
    }
    Node* newNode(int key){
        Node* p;
        if(bc_top)p = bc[--bc_top];
        else p = tail++;
        p->clear(key);
        return p;
    }
    //插入一个值 key
    void insert(Node* &root,int key){
        if(root == null){
            root = newNode(key);
            return;
        }
        Node* now = root;
        Node* pre = root->fa;
        while(now != null){
            if(now->key == key){
                now->cnt++;
                splay(root,now,null);
                return;
            }
            pre = now;
            now = now->ch[key >= now->key];
        }
        Node *x = newNode(key);
        pre->setc(x,key >= pre->key);
        splay(root,x,null);
    }
    //删除一个值 key
    void erase(Node* &root,int key){
        Node* now = root;
        while(now->key != key){
            now = now->ch[key >= now->key];
        }
        now->cnt--;
        if(now->cnt == 0)erase(root,now);
        else splay(root,now,null);
    }
    void Travel(Node* r){
        if(r == null)return;
        Travel(r->ch[0]);
        bc[bc_top++] = r;
        Travel(r->ch[1]);
    }
    void CLEAR(Node* &root){
        Travel(root);
        root = null;
    }
    //查询小于等于 val 的个数
    int query(Node *root,int val){
        int ans = 0;
        Node *x = root;
        while(x != null){
            if(val < x->key)x = x->ch[0];
            else{
                ans += x->ch[0]->size + x->cnt;
                x = x->ch[1];
            }
        }
        return ans;
    }
};
namespace ScapeGoatTree{
    const double alpha = 0.75;
    struct Node{
        Node *ch[2];
        int size,nodeCount,key;
        Splay::Node *root;
        bool isBad(){
            return ch[0]->nodeCount > alpha*nodeCount+5 || ch[1]->
              nodeCount > alpha*nodeCount+5;
        }
        void push_up(){
            size = 1+ch[0]->size+ch[1]->size;
            nodeCount = 1+ch[0]->nodeCount+ch[1]->nodeCount;
        }
    };
    Node pool[MAXN];
    Node *tail,*root,*null;
    Node *bc[MAXN];
    int bc_top;
    void init(){
        tail = pool;
        null = tail++;
        null->ch[0] = null->ch[1] = null;
        null->size = null->nodeCount = 0;
        null->root = Splay::null;
        bc_top = 0;
    }
    inline Node* newNode(int key){
        Node *p;
        if(bc_top)p = bc[--bc_top];
        else p = tail++;
        p->key = key;
        p->ch[0] = p->ch[1] = null;
        p->size = p->nodeCount = 1;
        p->root = Splay::null;
        return p;
    }
    Node *buildTree(int *a,int l,int r){
        if(l >= r)return null;
        int mid = (l+r)/2;
        Node *p = newNode(a[mid]);
        for(int i = l;i < r;i++)
            Splay::insert(p->root,a[i]);
        p->ch[0] = buildTree(a,l,mid);
        p->ch[1] = buildTree(a,mid+1,r);
        p->push_up();
        return p;
    }
    void Travel(Node *p,vector<int>&v){
        if(p == null)return;
        Travel(p->ch[0],v);
        v.push_back(p->key);
        Splay::CLEAR(p->root);
        bc[bc_top++] = p;
        Travel(p->ch[1],v);
    }
    Node *divide(vector<int>&v,int l,int r){
        if(l == r)return null;
        int mid = (l+r)/2;
        Node *p = newNode(v[mid]);
        for(int i = l;i < r;i++)
            Splay::insert(p->root,v[i]);
        p->ch[0] = divide(v,l,mid);
        p->ch[1] = divide(v,mid+1,r);
        p->push_up();
        return p;
    }
    inline void rebuild(Node *&p){
        vector<int>v;
        Travel(p,v);
        p = divide(v,0,v.size());
    }
    //将第 id 个值修改为 val
    int Modify(Node *p,int id,int val){
        if(id == p->ch[0]->size+1){
            int v = p->key;
            Splay::erase(p->root,v);
            Splay::insert(p->root,val);
            p->key = val;
            return v;
        }
        int res;
        if(p->ch[0]->size >= id)
            res = Modify(p->ch[0],id,val);
        else res = Modify(p->ch[1],id-p->ch[0]->size-1,val);
        Splay::erase(p->root,res);
        Splay::insert(p->root,val);
        return res;
    }
    Node **insert(Node *&p,int id,int val){
        if(p == null){
            p = newNode(val);
            Splay::insert(p->root,val);
            return &null;
        }
        else {
            p->size++;
            p->nodeCount++;
            Splay::insert(p->root,val);
            Node ** res;
            if(id <= p->ch[0]->size+1)
                res = insert(p->ch[0],id,val);
            else res = insert(p->ch[1],id-p->ch[0]->size-1,val);
            if(p->isBad())res = &p;
            return res;
        }
    }
    void insert(int id,int val){
        Node **p = insert(root,id,val);
        if(*p != null)rebuild(*p);
    }
    //查询在 [l,r] 区间，值小于等于 val 的个数
    int query(Node *p,int l,int r,int val){
        if(p == null)return 0;
        if(l <= 1 && p->size <= r)
            return Splay::query(p->root,val);
        else {
            int ans = 0;
            if(l <= p->ch[0]->size)
                ans += query(p->ch[0],l,r,val);
            if(r > p->ch[0]->size+1)
                ans += query(p->ch[1],l-p->ch[0]->size-1,r-p->ch
                   [0]->size-1,val);
            if(p->key <= val && l <= p->ch[0]->size+1 && p->ch[0]->
               size+1 <= r)
                ans++;
            return ans;
        }
    }
    int query(int L,int R,int k){
        int ans;
        int l = 0, r = 100000;
        while(l <= r){
            int mid = (l+r)/2;
            if(query(root,L,R,mid) >= k){
                ans = mid;
                r = mid-1;
            }
            else l = mid+1;
        }
        return ans;
    }
};
int a[MAXN];
int main()
{
    int n;
    while(scanf("%d",&n) == 1){
        Splay::init();
        ScapeGoatTree::init();
        for(int i = 0;i < n;i++)scanf("%d",&a[i]);
        ScapeGoatTree::root = ScapeGoatTree::buildTree(a,0,n);
        int m;
        char op[10];
        scanf("%d",&m);
        int ans = 0;
        while(m--){
            scanf("%s",op);
            if(op[0] == 'Q'){
                int x,y,k;
                scanf("%d%d%d",&x,&y,&k);
                x ^= ans; y ^= ans; k ^= ans;
                ans = ScapeGoatTree::query(x,y,k);
                printf("%d\n",ans);
            }
            else if(op[0] == 'M'){
                int x,val;
                scanf("%d%d",&x,&val);
                x ^= ans; val ^= ans;
                ScapeGoatTree::Modify(ScapeGoatTree::root,x,val);
            }
            else if(op[0] == 'I'){
                int x,val;
                scanf("%d%d",&x,&val);
                x ^= ans; val ^= ans;
                ScapeGoatTree::insert(x,val);
            }
        }
    }
    return 0;
}
```

4 图论

4.1 最短路

4.1.1 Dijkstra 单源最短路

权值必须是非负

```
/*
 * 单源最短路径，Dijkstra 算法，邻接矩阵形式，复杂度为O(n^2)
 * 求出源 beg 到所有点的最短路径，传入图的顶点数，和邻接矩阵 cost[][]
 * 返回各点的最短路径 lowcost[], 路径 pre[].pre[i] 记录 beg 到 i 路径上的
   父结点，pre[beg]=-1
 * 可更改路径权类型，但是权值必须为非负
 */
const int MAXN=1010;
#define typec int
const typec INF=0x3f3f3f3f;//防止后面溢出，这个不能太大
bool vis[MAXN];
int pre[MAXN];
void Dijkstra(typec cost[][MAXN],typec lowcost[],int n,int beg){
    for(int i=0;i<n;i++){
        lowcost[i]=INF;vis[i]=false;pre[i]=-1;
    }
    lowcost[beg]=0;
    for(int j=0;j<n;j++){
        int k=-1;
        int Min=INF;
        for(int i=0;i<n;i++)
            if(!vis[i]&&lowcost[i]<Min){
                Min=lowcost[i];
                k=i;
            }
        if(k==-1)break;
        vis[k]=true;
        for(int i=0;i<n;i++)
            if(!vis[i]&&lowcost[k]+cost[k][i]<lowcost[i]){
                lowcost[i]=lowcost[k]+cost[k][i];
                pre[i]=k;
            }
    }
}
```

4.1.2 Dijkstra 算法 + 堆优化

使用优先队列优化，复杂度 O (E log E)

```
/*
 * 使用优先队列优化 Dijkstra 算法
 * 复杂度 O(ElogE)
 * 注意对 vector<Edge>E[MAXN] 进行初始化后加边
 */
const int INF=0x3f3f3f3f;
const int MAXN=1000010;
struct qnode{
    int v;
    int c;
    qnode(int _v=0,int _c=0):v(_v),c(_c){}
    bool operator <(const qnode &r)const{
        return c>r.c;
    }
};
struct Edge{
    int v,cost;
    Edge(int _v=0,int _cost=0):v(_v),cost(_cost){}
};
vector<Edge>E[MAXN];
bool vis[MAXN];
int dist[MAXN];
//点的编号从 1 开始
void Dijkstra(int n,int start){
    memset(vis,false,sizeof(vis));
    for(int i=1;i<=n;i++)dist[i]=INF;
    priority_queue<qnode>que;
    while(!que.empty())que.pop();
    dist[start]=0;
    que.push(qnode(start,0));
    qnode tmp;
    while(!que.empty()){
        tmp=que.top();
        que.pop();
        int u=tmp.v;
        if(vis[u])continue;
        vis[u]=true;
        for(int i=0;i<E[u].size();i++){
            int v=E[tmp.v][i].v;
            int cost=E[u][i].cost;
            if(!vis[v]&&dist[v]>dist[u]+cost){
                dist[v]=dist[u]+cost;
                que.push(qnode(v,dist[v]));
            }
        }
    }
}
void addedge(int u,int v,int w){
    E[u].push_back(Edge(v,w));
}
```

4.1.3 单源最短路 bellman_ford 算法

```
/*
 * 单源最短路 bellman_ford 算法，复杂度 O(VE)
 * 可以处理负边权图。
 * 可以判断是否存在负环回路。返回 true，当且仅当图中不包含从源点可达的负权回路
 * vector<Edge>E; 先 E.clear() 初始化，然后加入所有边
 * 点的编号从 1 开始 (从 0 开始简单修改就可以了)
 */
const int INF=0x3f3f3f3f;
const int MAXN=550;
int dist[MAXN];
struct Edge{
    int u,v;
    int cost;
    Edge(int _u=0,int _v=0,int _cost=0):u(_u),v(_v),cost(_cost){}
};
vector<Edge>E;
//点的编号从 1 开始
bool bellman_ford(int start,int n){
    for(int i=1;i<=n;i++)dist[i]=INF;
    dist[start]=0;
    //最多做 n-1 次
    for(int i=1;i<n;i++){
        bool flag=false;
        for(int j=0;j<E.size();j++){
            int u=E[j].u;
            int v=E[j].v;
            int cost=E[j].cost;
            if(dist[v]>dist[u]+cost){
                dist[v]=dist[u]+cost;
                flag=true;
            }
        }
        if(!flag)return true;//没有负环回路
    }
    for(int j=0;j<E.size();j++)
        if(dist[E[j].v]>dist[E[j].u]+E[j].cost)
            return false;//有负环回路
    return true;//没有负环回路
}
```

4.1.4 单源最短路 SPFA

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

```
void addedge(int u,int v,int w){
    E[u].push_back(Edge(v,w));
}
bool vis[MAXN];//在队列标志
int cnt[MAXN];//每个点的入队列次数
int dist[MAXN];
bool SPFA(int start,int n){
    memset(vis,false,sizeof(vis));
    for(int i=1;i<=n;i++)dist[i]=INF;
    vis[start]=true;
    dist[start]=0;
    queue<int>que;
    while(!que.empty())que.pop();
    que.push(start);
    memset(cnt,0,sizeof(cnt));
    cnt[start]=1;
    while(!que.empty()){
        int u=que.front();
        que.pop();
        vis[u]=false;
        for(int i=0;i<E[u].size();i++){
            int v=E[u][i].v;
            if(dist[v]>dist[u]+E[u][i].cost){
                dist[v]=dist[u]+E[u][i].cost;
                if(!vis[v]){
                    vis[v]=true;
                    que.push(v);
                    if(++cnt[v]>n)return false;
                    //cnt[i] 为入队列次数，用来判定是否存在负环回路
                }
            }
        }
    }
    return true;
}
```

4.2 最小生成树

4.2.1 Prim 算法

```
/*
 * Prim 求 MST
 * 耗费矩阵 cost[][]，标号从 0 开始，0∼n-1
 * 返回最小生成树的权值，返回 -1 表示原图不连通
 */
const int INF=0x3f3f3f3f;
const int MAXN=110;
bool vis[MAXN];
int lowc[MAXN];
//点是 0 n-1
int Prim(int cost[][MAXN],int n){
    int ans=0;
    memset(vis,false,sizeof(vis));
    vis[0]=true;
    for(int i=1;i<n;i++)lowc[i]=cost[0][i];
    for(int i=1;i<n;i++){
        int minc=INF;
        int p=-1;
        for(int j=0;j<n;j++)
            if(!vis[j]&&minc>lowc[j]){
                minc=lowc[j];
                p=j;
            }
        if(minc==INF)return -1;//原图不连通
        ans+=minc;
        vis[p]=true;
        for(int j=0;j<n;j++)
            if(!vis[j]&&lowc[j]>cost[p][j])
                lowc[j]=cost[p][j];
    }
    return ans;
}
```

4.2.2 Kruskal 算法

```
/*
 * Kruskal 算法求 MST
 */
const int MAXN=110;//最大点数
const int MAXM=10000;//最大边数
int F[MAXN];//并查集使用
struct Edge{
    int u,v,w;
}edge[MAXM];//存储边的信息，包括起点/终点/权值
int tol;//边数，加边前赋值为 0
void addedge(int u,int v,int w){
    edge[tol].u=u;
    edge[tol].v=v;
    edge[tol++].w=w;
}
//排序函数，讲边按照权值从小到大排序
bool cmp(Edge a,Edge b){
    return a.w<b.w;
}
int find(int x){
    if(F[x]==-1)return x;
    else return F[x]=find(F[x]);
}
//传入点数，返回最小生成树的权值，如果不连通返回 -1
int Kruskal(int n){
    memset(F,-1,sizeof(F));
    sort(edge,edge+tol,cmp);
    int cnt=0;//计算加入的边数
    int ans=0;
    for(int i=0;i<tol;i++){
        int u=edge[i].u;
        int v=edge[i].v;
        int w=edge[i].w;
        int t1=find(u);
        int t2=find(v);
        if(t1!=t2){
            ans+=w;
            F[t1]=t2;
            cnt++;
        }
        if(cnt==n-1)break;
    }
    if(cnt<n-1)return -1;//不连通
    else return ans;
}
```

4.3 次小生成树

```
/*
 * 次小生成树
 * 求最小生成树时，用数组 Max[i][j] 来表示 MST 中 i 到 j 最大边权
 * 求完后，直接枚举所有不在 MST 中的边，替换掉最大边权的边，更新答案
 * 点的编号从 0 开始
 */
const int MAXN=110;
const int INF=0x3f3f3f3f;
bool vis[MAXN];
int lowc[MAXN];
int pre[MAXN];
int Max[MAXN][MAXN];//Max[i][j] 表示在最小生成树中从 i 到 j 的路径中的最
   大边权
bool used[MAXN][MAXN];
int Prim(int cost[][MAXN],int n){
    int ans=0;
    memset(vis,false,sizeof(vis));
    memset(Max,0,sizeof(Max));
    memset(used,false,sizeof(used));
    vis[0]=true;
    pre[0]=-1;
    for(int i=1;i<n;i++){
        lowc[i]=cost[0][i];
        pre[i]=0;
    }
    lowc[0]=0;
    for(int i=1;i<n;i++){
        int minc=INF;
        int p=-1;
        for(int j=0;j<n;j++)
            if(!vis[j]&&minc>lowc[j]){
                minc=lowc[j];
                p=j;
            }
        if(minc==INF)return -1;
        ans+=minc;
        vis[p]=true;
        used[p][pre[p]]=used[pre[p]][p]=true;
        for(int j=0;j<n;j++){
            if(vis[j] && j != p)Max[j][p]=Max[p][j]=max(Max[j][pre[
               p]],lowc[p]);
            if(!vis[j]&&lowc[j]>cost[p][j]){
                lowc[j]=cost[p][j];
                pre[j]=p;
            }
        }
    }
    return ans;
}
```

4.4 有向图的强连通分量

4.4.1 Tarjan

```
/*
 * Tarjan 算法
 * 复杂度 O(N+M)
 */
const int MAXN = 20010;//点数
const int MAXM = 50010;//边数
struct Edge{
    int to,next;
}edge[MAXM];
int head[MAXN],tot;
int Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN];//Belong 数组的值是
   1 ~ scc
int Index,top;
int scc;//强连通分量的个数
bool Instack[MAXN];
int num[MAXN];//各个强连通分量包含点的个数，数组编号 1 ~ scc
//num 数组不一定需要，结合实际情况

void addedge(int u,int v){
    edge[tot].to = v;edge[tot].next = head[u];head[u] = tot++;
}
void Tarjan(int u){
    int v;
    Low[u] = DFN[u] = ++Index;
    Stack[top++] = u;
    Instack[u] = true;
    for(int i = head[u];i != -1;i = edge[i].next){
        v = edge[i].to;
        if( !DFN[v] ){
            Tarjan(v);
            if( Low[u] > Low[v] )Low[u] = Low[v];
        }
        else if(Instack[v] && Low[u] > DFN[v])
            Low[u] = DFN[v];
    }
    if(Low[u] == DFN[u]){
        scc++;
        do{
            v = Stack[--top];
            Instack[v] = false;
            Belong[v] = scc;
            num[scc]++;
        }
        while( v != u);
    }
}
void solve(int N){
    memset(DFN,0,sizeof(DFN));
    memset(Instack,false,sizeof(Instack));
    memset(num,0,sizeof(num));
    Index = scc = top = 0;
    for(int i = 1;i <= N;i++)
        if(!DFN[i])
            Tarjan(i);
}
void init(){
    tot = 0;
    memset(head,-1,sizeof(head));
}
```

4.4.2 Kosaraju

```
/*
 * Kosaraju 算法，复杂度 O(N+M)
 */
const int MAXN = 20010;
const int MAXM = 50010;
struct Edge{
    int to,next;
}edge1[MAXM],edge2[MAXM];
//edge1 是原图 G，edge2 是逆图 GT
int head1[MAXN],head2[MAXN];
bool mark1[MAXN],mark2[MAXN];
int tot1,tot2;
int cnt1,cnt2;
int st[MAXN];//对原图进行 dfs，点的结束时间从小到大排序
int Belong[MAXN];//每个点属于哪个连通分量 (0~cnt2-1)
int num;//中间变量，用来数某个连通分量中点的个数
int setNum[MAXN];//强连通分量中点的个数，编号 0~cnt2-1
void addedge(int u,int v){
    edge1[tot1].to = v;edge1[tot1].next = head1[u];head1[u] = tot1
       ++;
    edge2[tot2].to = u;edge2[tot2].next = head2[v];head2[v] = tot2
       ++;
}
void DFS1(int u){
    mark1[u] = true;
    for(int i = head1[u];i != −1;i = edge1[i].next)
        if(!mark1[edge1[i].to])
            DFS1(edge1[i].to);
    st[cnt1++] = u;
}
void DFS2(int u){
    mark2[u] = true;
    num++;
    Belong[u] = cnt2;
    for(int i = head2[u];i != −1;i = edge2[i].next)
        if(!mark2[edge2[i].to])
            DFS2(edge2[i].to);
}
//点的编号从 1 开始
void solve(int n){
    memset(mark1,false,sizeof(mark1));
    memset(mark2,false,sizeof(mark2));
    cnt1 = cnt2 = 0;
    for(int i = 1;i <= n;i++)
        if(!mark1[i])
            DFS1(i);
    for(int i = cnt1−1;i >= 0; i−−)
        if(!mark2[st[i]]){
            num = 0;
            DFS2(st[i]);
            setNum[cnt2++] = num;
        }
}
```

4.5 图的割点、桥和双连通分支的基本概念

[点连通度与边连通度] 在一个无向连通图中，如果有一个顶点集合，删除这个顶点集合，以及这个集合中所有顶点相关联的边以后，原图变成多个连通块，就称这个点集为割点集合。一个图的点连通度的定义为，最小割点集合中的顶点数。
类似的，如果有一个边集合，删除这个边集合以后，原图变成多个连通块，就称这个点集为割边集合。一个图的边连通度的定义为，最小割边集合中的边数。
[双连通图、割点与桥]
如果一个无向连通图的点连通度大于 1，则称该图是点双连通的 (point biconnected)，简称双连通或重连通。一个图有割点，当且仅当这个图的点连通度为 1，则割点集合的唯一元素被称为割点 (cut point)，又叫关节点 (articulation point)。
如果一个无向连通图的边连通度大于 1，则称该图是边双连通的 (edge biconnected)，简称双连通或重连通。一个图有桥，当且仅当这个图的边连通度为 1，则割边集合的唯一元素被称为桥 (bridge)，又叫关节边 (articulation edge)。
可以看出，点双连通与边双连通都可以简称为双连通，它们之间是有着某种联系的，下文中提到的双连通，均既可指点双连通，又可指边双连通。
[双连通分支]
在图 G 的所有子图 G' 中，如果 G' 是双连通的，则称 G' 为双连通子图。如果一个双连

通子图 G' 它不是任何一个双连通子图的真子集，则 G' 为极大双连通子图。双连通分支 (biconnected component)，或重连通分支，就是图的极大双连通子图。特殊的，点双连通分支又叫做块。[求割点与桥]
该算法是 R.Tarjan 发明的。对图深度优先搜索，定义 DFS(u) 为 u 在搜索树（以下简称为树）中被遍历到的次序号。定义 Low(u) 为 u 或 u 的子树中能通过非父子边追溯到的最早的节点，即 DFS 序号最小的节点。根据定义，则有：
Low(u)=Min DFS(u) DFS(v) (u,v) 为后向边 (返祖边) 等价于 DFS(v)<DFS(u) 且 v 不为 u 的父亲节点 Low(v) (u,v) 为树枝边 (父子边) 一个顶点 u 是割点，当且仅当满足 (1) 或 (2) (1) u 为树根，且 u 有多于一个子树。(2) u 不为树根，且满足存在 (u,v) 为树枝边 (或称父子边，即 u 为 v 在搜索树中的父亲)，使得 DFS(u)<=Low(v)。
一条无向边 (u,v) 是桥，当且仅当 (u,v) 为树枝边，且满足 DFS(u)<Low(v)。
[求双连通分支]
下面要分开讨论点双连通分支与边双连通分支的求法。
对于点双连通分支，实际上在求割点的过程中就能顺便把每个点双连通分支求出。建立一个栈，存储当前双连通分支，在搜索图时，每找到一条树枝边或后向边 (非横叉边)，就把这条边加入栈中。如果遇到某时满足 DFS(u)<=Low(v)，说明 u 是一个割点，同时把边从栈顶一个个取出，直到遇到了边 (u,v)，取出的这些边与其关联的点，组成一个点双连通分支。割点可以属于多个点双连通分支，其余点和每条边只属于且属于一个点双连通分支。
对于边双连通分支，求法更为简单。只需在求出所有的桥以后，把桥边删除，原图变成了多个连通块，则每个连通块就是一个边双连通分支。桥不属于任何一个边双连通分支，其余的边和每个顶点都属于且只属于一个边双连通分支。
[构造双连通图]
一个有桥的连通图，如何把它通过加边变成边双连通图？方法为首先求出所有的桥，然后删除这些桥边，剩下的每个连通块都是一个双连通子图。把每个双连通子图收缩为一个顶点，再把桥边加回来，最后的这个图一定是一棵树，边连通度为 1。
统计出树中度为 1 的节点的个数，即为叶节点的个数，记为 leaf。则至少在树上添加 (leaf+1)/2 条边，就能使树达到边二连通，所以至少添加的边数就是 (leaf+1)/2。具体方法为，首先把两个最近公共祖先最远的两个叶节点之间连接一条边，这样可以把这两个点到祖先的路径上所有点收缩到一起，因为一个形成的环一定是双连通的。然后再找两个最近公共祖先最远的两个叶节点，这样一对一对找完，恰好是 (leaf+1)/2 次，把所有点收缩到了一起。

4.6 割点与桥

4.6.1 模板

```
/*
*  求无向图的割点和桥
*  可以找出割点和桥，求删掉每个点后增加的连通块。
*  需要注意重边的处理，可以先用矩阵存，再转邻接表，或者进行判重
*/
const int MAXN = 10010;
const int MAXM = 100010;
struct Edge{
    int to,next;
    bool cut;//是否为桥的标记
}edge[MAXM];
int head[MAXN],tot;
int Low[MAXN],DFN[MAXN],Stack[MAXN];
int Index,top;
bool Instack[MAXN];
bool cut[MAXN];
int add_block[MAXN];//删除一个点后增加的连通块
int bridge;
void addedge(int u,int v){
    edge[tot].to = v;edge[tot].next = head[u];edge[tot].cut = false
       ;
    head[u] = tot++;
}
void Tarjan(int u,int pre){
    int v;
    Low[u] = DFN[u] = ++Index;
    Stack[top++] = u;
    Instack[u] = true;
    int son = 0;
    int pre_cnt = 0; //处理重边，如果不需要可以去掉
    for(int i = head[u];i != −1;i = edge[i].next){
        v = edge[i].to;
        if(v == pre && pre_cnt == 0){pre_cnt++;continue;}
        if( !DFN[v] ){
            son++;
            Tarjan(v,u);
            if(Low[u] > Low[v])Low[u] = Low[v];
            //桥
            //一条无向边 (u,v) 是桥，当且仅当 (u,v) 为树枝边，且满足
               DFS(u)<Low(v)。
            if(Low[v] > DFN[u]){
                bridge++;
                edge[i].cut = true;
                edge[i^1].cut = true;
            }
            //割点
            //一个顶点 u 是割点，当且仅当满足 (1) 或 (2) (1) u 为树根，且
               u 有多于一个子树。
            //(2) u 不为树根，且满足存在 (u,v) 为树枝边 (或称父子边，
            //即 u 为 v 在搜索树中的父亲)，使得 DFS(u)<=Low(v)
            if(u != pre && Low[v] >= DFN[u]){//不是树根
                cut[u] = true;
                add_block[u]++;
            }
        }
        else if( Low[u] > DFN[v])
             Low[u] = DFN[v];
    }
    //树根，分支数大于 1
    if(u == pre && son > 1)cut[u] = true;
    if(u == pre)add_block[u] = son − 1;
    Instack[u] = false;
    top−−;
}
```

4.6.2 调用

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

```
void solve(int N){
    memset(DFN,0,sizeof(DFN));
    memset(Instack,false,sizeof(Instack));
    memset(add_block,0,sizeof(add_block));
    memset(cut,false,sizeof(cut));
    Index = top = 0;
    bridge = 0;
    for(int i = 1;i <= N;i++)
        if( !DFN[i] )
            Tarjan(i,i);
    printf("%d critical links\n",bridge);
    vector<pair<int,int> >ans;
    for(int u = 1;u <= N;u++)
        for(int i = head[u];i != -1;i = edge[i].next)
            if(edge[i].cut && edge[i].to > u)
            {
                ans.push_back(make_pair(u,edge[i].to));
            }
    sort(ans.begin(),ans.end());
    //按顺序输出桥
    for(int i = 0;i < ans.size();i++)
        printf("%d - %d\n",ans[i].first-1,ans[i].second-1);
    printf("\n");
}
void init(){
    tot = 0;
    memset(head,-1,sizeof(head));
}
//处理重边
map<int,int>mapit;
inline bool isHash(int u,int v){
    if(mapit[u*MAXN+v])return true;
    if(mapit[v*MAXN+u])return true;
    mapit[u*MAXN+v] = mapit[v*MAXN+u] = 1;
    return false;
}
int main(){
    int n;
    while(scanf("%d",&n) == 1){
        init();
        int u;
        int k;
        int v;
        //mapit.clear();
        for(int i = 1;i <= n;i++){
            scanf("%d (%d)",&u,&k);
            u++;
            //这样加边，要保证正边和反边是相邻的，建无向图
            while(k--){
                scanf("%d",&v);
                v++;
                if(v <= u)continue;
                //if(isHash(u,v))continue;
                addedge(u,v);
                addedge(v,u);
            }
        }
        solve(n);
    }
    return 0;
}
```

2）POJ 2117 求删除一个点后，图中最多有多少个连通块

```
void solve(int N){
    memset(DFN,0,sizeof(DFN));
    memset(Instack,0,sizeof(Instack));
    memset(add_block,0,sizeof(add_block));
    memset(cut,false,sizeof(cut));
    Index = top = 0;
    int cnt = 0;//原来的连通块数
    for(int i = 1;i <= N;i++)
       if( !DFN[i] ){
           Tarjan(i,i);//找割点调用必须是 Tarjan(i,i)
           cnt++;
       }
    int ans = 0;
    for(int i = 1;i <= N;i++)
       ans = max(ans,cnt+add_block[i]);
    printf("%d\n",ans);
}
void init(){
    tot = 0;
    memset(head,-1,sizeof(head));
}
int main(){
    int n,m;
    int u,v;
    while(scanf("%d%d",&n,&m)==2){
        if(n==0 && m == 0)break;
        init();
        while(m--){
            scanf("%d%d",&u,&v);
            u++;v++;
            addedge(u,v);
            addedge(v,u);
        }
        solve(n);
    }
    return 0;
}
```

4.7 边双连通分支

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

```
const int MAXN = 5010;//点数
const int MAXM = 20010;//边数，因为是无向图，所以这个值要 *2
struct Edge{
    int to,next;
    bool cut;//是否是桥标记
}edge[MAXM];
int head[MAXN],tot;
int Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN];//Belong 数组的值是
   1 ~ block
int Index,top;
int block;//边双连通块数
bool Instack[MAXN];
int bridge;//桥的数目
void addedge(int u,int v){
    edge[tot].to = v;edge[tot].next = head[u];edge[tot].cut=false;
    head[u] = tot++;
}
void Tarjan(int u,int pre){
    int v;
    Low[u] = DFN[u] = ++Index;
    Stack[top++] = u;
    Instack[u] = true;
    int pre_cnt = 0;
    for(int i = head[u];i != -1;i = edge[i].next){
        v = edge[i].to;
        if(v == pre && pre_cnt == 0){pre_cnt++; continue;}
        if( !DFN[v] ){
            Tarjan(v,u);
            if( Low[u] > Low[v] )Low[u] = Low[v];
            if(Low[v] > DFN[u]){
                bridge++;
                edge[i].cut = true;
                edge[i^1].cut = true;
            }
        }
        else if( Instack[v] && Low[u] > DFN[v] )
            Low[u] = DFN[v];
    }
    if(Low[u] == DFN[u]){
        block++;
        do{
            v = Stack[--top];
            Instack[v] = false;
            Belong[v] = block;
        }
        while( v!=u );
    }
}
void init(){
    tot = 0;
    memset(head,-1,sizeof(head));
}
int du[MAXN];//缩点后形成树，每个点的度数
void solve(int n){
    memset(DFN,0,sizeof(DFN));
    memset(Instack,false,sizeof(Instack));
    Index = top = block = 0;
    Tarjan(1,0);
    int ans = 0;
    memset(du,0,sizeof(du));
    for(int i = 1;i <= n;i++)
       for(int j = head[i];j != -1;j = edge[j].next)
          if(edge[j].cut)
             du[Belong[i]]++;
    for(int i = 1;i <= block;i++)
       if(du[i]==1)
          ans++;
    //找叶子结点的个数 ans，构造边双连通图需要加边 (ans+1)/2
    printf("%d\n",(ans+1)/2);
}
int main(){
    int n,m;
    int u,v;
    while(scanf("%d%d",&n,&m)==2){
        init();
        while(m--){
            scanf("%d%d",&u,&v);
            addedge(u,v);
            addedge(v,u);
        }
        solve(n);
    }
    return 0;
}
```

4.8 点双连通分支

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

POJ 2942

奇圈，二分图判断的染色法，求点双连通分支

```
/*
POJ 2942 Knights of the Round Table
```

```
亚瑟王要在圆桌上召开骑士会议，为了不引发骑士之间的冲突，
并且能够让会议的议题有令人满意的结果，每次开会前都必须对出席会议的骑士有如下要求：
1、相互憎恨的两个骑士不能坐在直接相邻的 2 个位置；
2、出席会议的骑士数必须是奇数，这是为了让投票表决议题时都能有结果。

注意：1、所给出的憎恨关系一定是双向的，不存在单向憎恨关系。
2、由于是圆桌会议，则每个出席的骑士身边必定刚好有 2 个骑士。
即每个骑士的座位两边都必定各有一个骑士。
3、一个骑士无法开会，就是说至少有 3 个骑士才可能开会。

首先根据给出的互相憎恨的图中得到补图。
然后就相当于找出不能形成奇圈的点。
利用下面两个定理：
（1）如果一个双连通分量内的某些顶点在一个奇圈中（即双连通分量含有奇圈），
那么这个双连通分量的其他顶点也在某个奇圈中；
（2）如果一个双连通分量含有奇圈，则他必定不是一个二分图。反过来也成立，这是一个充要条件。

所以本题的做法，就是对补图求点双连通分量。
然后对于求得的点双连通分量，使用染色法判断是不是二分图，不是二分图，这个双连通分量的点是可以存在的
*/

const int MAXN = 1010;
const int MAXM = 2000010;
struct Edge{
    int to,next;
}edge[MAXM];
int head[MAXN],tot;
int Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN];
int Index,top;
int block;//点双连通分量的个数
bool Instack[MAXN];
bool can[MAXN];
bool ok[MAXN];//标记
int tmp[MAXN];//暂时存储双连通分量中的点
int cc;//tmp 的计数
int color[MAXN];//染色
void addedge(int u,int v){
    edge[tot].to = v;edge[tot].next = head[u];head[u] = tot++;
}
//染色判断二分图
bool dfs(int u,int col){
    color[u] = col;
    for(int i = head[u];i != -1;i = edge[i].next){
        int v = edge[i].to;
        if( !ok[v] )continue;
        if(color[v] != -1){
            if(color[v]==col)return false;
            continue;
        }
        if(!dfs(v,!col))return false;
    }
    return true;
}
void Tarjan(int u,int pre){
    int v;
    Low[u] = DFN[u] = ++Index;
    Stack[top++] = u;
    Instack[u] = true;
    int pre_cnt = 0;
    for(int i = head[u];i != −1;i = edge[i].next){
        v = edge[i].to;
        if(v == pre && pre_cnt == 0){pre_cnt++; continue;}
        if( !DFN[v] ){
            Tarjan(v,u);
            if(Low[u] > Low[v])Low[u] = Low[v];
            if( Low[v] >= DFN[u]){
                block++;
                int vn;
                cc = 0;
                memset(ok,false,sizeof(ok));
                do{
                    vn = Stack[−−top];
                    Belong[vn] = block;
                    Instack[vn] = false;
                    ok[vn] = true;
                    tmp[cc++] = vn;
                }
                while( vn!=v );
                ok[u] = 1;
                memset(color,−1,sizeof(color));
                if( !dfs(u,0) ){
                    can[u] = true;
                    while(cc−−)can[tmp[cc]]=true;
                }
            }
        }
        else if(Instack[v] && Low[u] > DFN[v])
           Low[u] = DFN[v];
    }
}
void solve(int n){
    memset(DFN,0,sizeof(DFN));
    memset(Instack,false,sizeof(Instack));
    Index = block = top = 0;
    memset(can,false,sizeof(can));
    for(int i = 1;i <= n;i++)
       if(!DFN[i])
          Tarjan(i,−1);
    int ans = n;
    for(int i = 1;i <= n;i++)
        if(can[i])
          ans--;
    printf("%d\n",ans);
}
void init(){
    tot = 0;
    memset(head,-1,sizeof(head));
}
int g[MAXN][MAXN];
int main(){
    int n,m;
    int u,v;
    while(scanf("%d%d",&n,&m)==2){
        if(n==0 && m==0)break;
        init();
        memset(g,0,sizeof(g));
        while(m--){
            scanf("%d%d",&u,&v);
            g[u][v]=g[v][u]=1;
        }
        for(int i = 1;i <= n;i++)
          for(int j = 1;j <= n;j++)
             if(i != j && g[i][j]==0)
                addedge(i,j);
        solve(n);
    }
    return 0;
}
```

4.9 最小树形图

```
/*
 * 最小树形图
 * int 型
 * 复杂度 O(NM)
 * 点从 0 开始
 */
const int INF = 0x3f3f3f3f;
const int MAXN = 1010;
const int MAXM = 40010;
struct Edge{
    int u,v,cost;
};
Edge edge[MAXM];
int pre[MAXN],id[MAXN],visit[MAXN],in[MAXN];
int zhuliu(int root,int n,int m,Edge edge[]){
    int res = 0,u,v;
    while(1){
        for(int i = 0;i < n;i++)
            in[i] = INF;
        for(int i = 0;i < m;i++)
            if(edge[i].u != edge[i].v && edge[i].cost < in[edge[i].
               v]){
                pre[edge[i].v] = edge[i].u;
                in[edge[i].v] = edge[i].cost;
            }
        for(int i = 0;i < n;i++)
            if(i != root && in[i] == INF)
                return −1;//不存在最小树形图
        int tn = 0;
        memset(id,−1,sizeof(id));
        memset(visit,−1,sizeof(visit));
        in[root] = 0;
        for(int i = 0;i < n;i++){
            res += in[i];
            v = i;
            while( visit[v] != i && id[v] == −1 && v != root){
                visit[v] = i;
                v = pre[v];
            }
            if( v != root && id[v] == −1 ){
                for(int u = pre[v]; u != v ;u = pre[u])
                    id[u] = tn;
                id[v] = tn++;
            }
        }
        if(tn == 0)break;//没有有向环
        for(int i = 0;i < n;i++)
            if(id[i] == −1)
                id[i] = tn++;
        for(int i = 0;i < m;){
            v = edge[i].v;
            edge[i].u = id[edge[i].u];
            edge[i].v = id[edge[i].v];
            if(edge[i].u != edge[i].v)
                edge[i++].cost −= in[v];
            else
                swap(edge[i],edge[−−m]);
        }
        n = tn;
        root = id[root];
    }
    return res;
}
int g[MAXN][MAXN];
int main(){
    int n,m;
    int iCase = 0;
    int T;
    scanf("%d",&T);
    while( T−− ){
        iCase ++;
        scanf("%d%d",&n,&m);
        for(int i = 0;i < n;i++)
            for(int j = 0;j < n;j++)
                g[i][j] = INF;
        int u,v,cost;
        while(m--){
            scanf("%d%d%d",&u,&v,&cost);
            if(u == v)continue;
            g[u][v] = min(g[u][v],cost);
        }
        int L = 0;
        for(int i = 0;i < n;i++)
            for(int j = 0;j < n;j++)
                if(g[i][j] < INF){
                    edge[L].u = i;
                    edge[L].v = j;
                    edge[L++].cost = g[i][j];
                }
        int ans = zhuliu(0,n,L,edge);
        printf("Case␣#%d:␣",iCase);
        if(ans == -1)printf("Possums!\n");
        else printf("%d\n",ans);
    }
    return 0;
}
```

4.10 二分图匹配

4.10.1 邻接矩阵（匈牙利算法）

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

4.10.2 邻接表（匈牙利算法）

```
/*
 * 匈牙利算法邻接表形式
 * 使用前用 init() 进行初始化，给 uN 赋值
 * 加边使用函数 addedge(u,v)
 *
 */
const int MAXN = 5010;//点数的最大值
const int MAXM = 50010;//边数的最大值
struct Edge{
    int to,next;
}edge[MAXM];
int head[MAXN],tot;
void init(){
    tot = 0;
    memset(head,-1,sizeof(head));
}
void addedge(int u,int v){
    edge[tot].to = v; edge[tot].next = head[u];
    head[u] = tot++;
}
int linker[MAXN];
bool used[MAXN];
int uN;
bool dfs(int u){
    for(int i = head[u]; i != -1 ;i = edge[i].next){
        int v = edge[i].to;
        if(!used[v]){
            used[v] = true;
            if(linker[v] == -1 || dfs(linker[v])){
                linker[v] = u;
                return true;
            }
        }
    }
    return false;
}
int hungary(){
    int res = 0;
    memset(linker,-1,sizeof(linker));
    //点的编号 0~uN-1
    for(int u = 0; u < uN;u++){
        memset(used,false,sizeof(used));
        if(dfs(u))res++;
    }
    return res;
}
```

4.10.3 Hopcroft-Karp 算法

```
/* *******************************
 * 二分图匹配（Hopcroft-Karp 算法）
 * 复杂度 O(sqrt(n)*E)
 * 邻接表存图，vector 实现
 * vector 先初始化，然后假如边
 * uN 为左端的顶点数，使用前赋值 (点编号 0 开始)
 */
const int MAXN = 3000;
const int INF = 0x3f3f3f3f;
vector<int>G[MAXN];
int uN;
int Mx[MAXN],My[MAXN];
int dx[MAXN],dy[MAXN];
int dis;
bool used[MAXN];
bool SearchP(){
    queue<int>Q;
    dis = INF;
    memset(dx,-1,sizeof(dx));
    memset(dy,-1,sizeof(dy));
    for(int i = 0 ; i < uN; i++)
        if(Mx[i] == -1){
            Q.push(i);
            dx[i] = 0;
        }
    while(!Q.empty()){
        int u = Q.front();
        Q.pop();
        if(dx[u] > dis)break;
        int sz = G[u].size();
        for(int i = 0;i < sz;i++){
            int v = G[u][i];
            if(dy[v] == -1){
                dy[v] = dx[u] + 1;
                if(My[v] == -1)dis = dy[v];
                else{
                    dx[My[v]] = dy[v] + 1;
```

```cpp
                    Q.push(My[v]);
                }
            }
        }
    }
    return dis != INF;
}
bool DFS(int u){
    int sz = G[u].size();
    for(int i = 0;i < sz;i++){
        int v = G[u][i];
        if(!used[v] && dy[v] == dx[u] + 1){
            used[v] = true;
            if(My[v] != -1 && dy[v] == dis)continue;
            if(My[v] == -1 || DFS(My[v])){
                My[v] = u;
                Mx[u] = v;
                return true;
            }
        }
    }
    return false;
}
int MaxMatch(){
    int res = 0;
    memset(Mx,-1,sizeof(Mx));
    memset(My,-1,sizeof(My));
    while(SearchP()){
        memset(used,false,sizeof(used));
        for(int i = 0;i < uN;i++)
            if(Mx[i] == -1 && DFS(i))
                res++;
    }
    return res;
}
```

4.11 二分图多重匹配

```cpp
const int MAXN = 1010;
const int MAXM = 510;
int uN,vN;
int g[MAXN][MAXM];
int linker[MAXM][MAXN];
bool used[MAXM];
int num[MAXM];//右边最大的匹配数
bool dfs(int u){
    for(int v = 0; v < vN;v++)
        if(g[u][v] && !used[v]){
            used[v] = true;
            if(linker[v][0] < num[v]){
                linker[v][++linker[v][0]] = u;
                return true;
            }
            for(int i = 1;i <= num[v];i++)
                if(dfs(linker[v][i])){
                    linker[v][i] = u;
                    return true;
                }
        }
    return false;
}
int hungary(){
    int res = 0;
    for(int i = 0;i < vN;i++)
        linker[i][0] = 0;
    for(int u = 0; u < uN; u++){
        memset(used,false,sizeof(used));
        if(dfs(u))res++;
    }
    return res;
}
```

4.12 二分图最大权匹配（KM 算法）

```
/*  KM 算法
 *   复杂度 O(nx*nx*ny)
 *  求最大权匹配
 *   若求最小权匹配，可将权值取相反数，结果取相反数
 *  点的编号从 0 开始
 */
const int N = 310;
const int INF = 0x3f3f3f3f;
int nx,ny;//两边的点数
int g[N][N];//二分图描述
int linker[N],lx[N],ly[N];//y 中各点匹配状态，x,y 中的点标号
int slack[N];
bool visx[N],visy[N];
bool DFS(int x){
    visx[x] = true;
    for(int y = 0; y < ny; y++){
        if(visy[y])continue;
        int tmp = lx[x] + ly[y] - g[x][y];
        if(tmp == 0){
            visy[y] = true;
            if(linker[y] == -1 || DFS(linker[y])){
                linker[y] = x;
                return true;
            }
        }
        else if(slack[y] > tmp)
            slack[y] = tmp;
    }
    return false;
}
```

```
int KM(){
    memset(linker,-1,sizeof(linker));
    memset(ly,0,sizeof(ly));
    for(int i = 0;i < nx;i++){
        lx[i] = -INF;
        for(int j = 0;j < ny;j++)
            if(g[i][j] > lx[i])
                lx[i] = g[i][j];
    }
    for(int x = 0;x < nx;x++){
        for(int i = 0;i < ny;i++)
            slack[i] = INF;
        while(true){
            memset(visx,false,sizeof(visx));
            memset(visy,false,sizeof(visy));
            if(DFS(x))break;
            int d = INF;
            for(int i = 0;i < ny;i++)
                if(!visy[i] && d > slack[i])
                    d = slack[i];
            for(int i = 0;i < nx;i++)
                if(visx[i])
                    lx[i] -= d;
            for(int i = 0;i < ny;i++){
                if(visy[i])ly[i] += d;
                else slack[i] -= d;
            }
        }
    }
    int res = 0;
    for(int i = 0;i < ny;i++)
        if(linker[i] != -1)
            res += g[linker[i]][i];
    return res;
}
//HDU 2255
int main(){
    int n;
    while(scanf("%d",&n) == 1){
        for(int i = 0;i < n;i++)
            for(int j = 0;j < n;j++)
                scanf("%d",&g[i][j]);
        nx = ny = n;
        printf("%d\n",KM());
    }
    return 0;
}
```

4.13 一般图匹配带花树

URAL 1099

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

```
int N; //点的个数，点的编号从 1 到 N
bool Graph[MAXN][MAXN];
int Match[MAXN];
bool InQueue[MAXN],InPath[MAXN],InBlossom[MAXN];
int Head,Tail;
int Queue[MAXN];
int Start,Finish;
int NewBase;
int Father[MAXN],Base[MAXN];
int Count;//匹配数，匹配对数是 Count/2
void CreateGraph(){
    int u,v;
    memset(Graph,false,sizeof(Graph));
    scanf("%d",&N);
    while(scanf("%d%d",&u,&v) == 2){
        Graph[u][v] = Graph[v][u] = true;
    }
}
void Push(int u){
    Queue[Tail] = u;
    Tail++;
    InQueue[u] = true;
}
int Pop(){
    int res = Queue[Head];
    Head++;
    return res;
}
int FindCommonAncestor(int u,int v){
    memset(InPath,false,sizeof(InPath));
    while(true){
        u = Base[u];
        InPath[u] = true;
        if(u == Start) break;
        u = Father[Match[u]];
    }
    while(true){
        v = Base[v];
        if(InPath[v])break;
        v = Father[Match[v]];
    }
    return v;
}
void ResetTrace(int u){
    int v;
    while(Base[u] != NewBase){
        v = Match[u];
        InBlossom[Base[u]] = InBlossom[Base[v]] = true;
        u = Father[v];
        if(Base[u] != NewBase) Father[u] = v;
    }
```

```
}
void BloosomContract(int u,int v){
    NewBase = FindCommonAncestor(u,v);
    memset(InBlossom,false,sizeof(InBlossom));
    ResetTrace(u);
    ResetTrace(v);
    if(Base[u] != NewBase) Father[u] = v;
    if(Base[v] != NewBase) Father[v] = u;
    for(int tu = 1; tu <= N; tu++)
        if(InBlossom[Base[tu]]){
            Base[tu] = NewBase;
            if(!InQueue[tu]) Push(tu);
        }
}
void FindAugmentingPath(){
    memset(InQueue,false,sizeof(InQueue));
    memset(Father,0,sizeof(Father));
    for(int i = 1;i <= N;i++)
        Base[i] = i;
    Head = Tail = 1;
    Push(Start);
    Finish = 0;
    while(Head < Tail){
        int u = Pop();
        for(int v = 1; v <= N; v++)
            if(Graph[u][v] && (Base[u] != Base[v]) && (Match[u] !=
               v)){
                if((v == Start) || ((Match[v] > 0) && Father[Match[
                   v]] > 0))
                    BloosomContract(u,v);
                else if(Father[v] == 0){
                    Father[v] = u;
                    if(Match[v] > 0)
                        Push(Match[v]);
                    else{
                        Finish = v;
                        return;
                    }
                }
            }
    }
}
void AugmentPath(){
    int u,v,w;
    u = Finish;
    while(u > 0){
        v = Father[u];
        w = Match[v];
        Match[v] = u;
        Match[u] = v;
        u = w;
```

```cpp
    }
}
void Edmonds(){
    memset(Match,0,sizeof(Match));
    for(int u = 1; u <= N; u++)
        if(Match[u] == 0){
            Start = u;
            FindAugmentingPath();
            if(Finish > 0)AugmentPath();
        }
}
void PrintMatch(){
    Count = 0;
    for(int u = 1; u <= N;u++)
        if(Match[u] > 0)
            Count++;
    printf("%d\n",Count);
    for(int u = 1; u <= N; u++)
        if(u < Match[u])
            printf("%d␣%d\n",u,Match[u]);
}
int main(){
    CreateGraph();//建图
    Edmonds();//进行匹配
    PrintMatch();//输出匹配数和匹配
    return 0;
}
```

4.14 一般图最大加权匹配

```cpp
//一般图的最大加权匹配模板
//注意 G 的初始化, 需要偶数个点, 刚好可以形成 n/2 个匹配
//如果要求最小权匹配, 可以取相反数, 或者稍加修改就可以了
//点的编号从 0 开始的
const int MAXN = 110;
const int INF = 0x3f3f3f3f;
int G[MAXN][MAXN];
int cnt_node;//点的个数
int dis[MAXN];
int match[MAXN];
bool vis[MAXN];
int sta[MAXN],top;//堆栈
bool dfs(int u){
    sta[top++] = u;
    if(vis[u])return true;
    vis[u] = true;
    for(int i = 0;i < cnt_node;i++)
        if(i != u && i != match[u] && !vis[i]){
            int t = match[i];
            if(dis[t] < dis[u] + G[u][i] - G[i][t]){
                dis[t] = dis[u] + G[u][i] - G[i][t];
                if(dfs(t))return true;
            }
        }
    top--;
    vis[u] = false;
    return false;
}
int P[MAXN];
//返回最大匹配权值
int get_Match(int N){
    cnt_node = N;
    for(int i = 0;i < cnt_node;i++)P[i] = i;
    for(int i = 0;i < cnt_node;i += 2){
        match[i] = i+1;
        match[i+1] = i;
    }
    int cnt = 0;
    while(1){
        memset(dis,0,sizeof(dis));
        memset(vis,false,sizeof(vis));
        top = 0;
        bool update = false;
        for(int i = 0;i < cnt_node;i++)
            if(dfs(P[i])){
                update = true;
                int tmp = match[sta[top-1]];
                int j = top-2;
                while(sta[j] != sta[top-1]){
                    match[tmp] = sta[j];
                    swap(tmp,match[sta[j]]);
                    j--;
                }
                match[tmp] = sta[j];
                match[sta[j]] = tmp;
                break;
            }
        if(!update){
            cnt++;
            if(cnt >= 3)break;
            random_shuffle(P,P+cnt_node);
        }
    }
    int ans = 0;
    for(int i = 0;i < cnt_node;i++){
        int v = match[i];
        if(i < v)ans += G[i][v];
    }
    return ans;
}
```

4.15 生成树计数

Matrix-Tree 定理 (Kirchhoff 矩阵 -树定理)
1、G 的度数矩阵 D[G] 是一个 n*n 的矩阵，并且满足：当 i j 时,dij=0；当 i=j 时，dij 等于 vi 的度数。
2、G 的邻接矩阵 A[G] 也是一个 n*n 的矩阵，并且满足：如果 vi、vj 之间有边直接相连，则 aij=1，否则为 0。
我们定义 G 的 Kirchhoff 矩阵 (也称为拉普拉斯算子)C[G] 为 C[G]=D[G]-A[G]，则 Matrix-Tree 定理可以描述为：
G 的所有不同的生成树的个数等于其 Kirchhoff 矩阵 C[G] 任何一个 n-1 阶主子式的行列式的绝对值。
所谓 n-1 阶主子式，就是对于 r(1 r n)，将 C[G] 的第 r 行、第 r 列同时去掉后得到的新矩阵，用 Cr[G] 表示。

//HDU 4305
//求生成树计数部分代码，计数对 10007 取模

```
const int MOD = 10007;
int INV[MOD];
//求 ax = 1( mod m) 的 x 值，就是逆元 (0<a<m)
long long inv(long long a,long long m){
    if(a == 1)return 1;
    return inv(m%a,m)*(m-m/a)%m;
}
struct Matrix{
    int mat[330][330];
    void init(){
        memset(mat,0,sizeof(mat));
    }
    //求行列式的值模上，需要使用逆元MOD
    int det(int n){
        for(int i = 0;i < n;i++)
            for(int j = 0;j < n;j++)
                mat[i][j] = (mat[i][j]%MOD+MOD)%MOD;
        int res = 1;
        for(int i = 0;i < n;i++){
            for(int j = i;j < n;j++)
                if(mat[j][i]!=0){
                    for(int k = i;k < n;k++)
                        swap(mat[i][k],mat[j][k]);
                    if(i != j)
                        res = (-res+MOD)%MOD;
                    break;
                }
            if(mat[i][i] == 0){
                res = -1;//不存在也就是行列式值为(0)
                break;
            }
            for(int j = i+1;j < n;j++){
                //int mut = (mat[j][i]*INV[mat[i][i]])%MOD打表逆元;//
                int mut = (mat[j][i]*inv(mat[i][i],MOD))%MOD;
                for(int k = i;k < n;k++)
                    mat[j][k] = (mat[j][k]-(mat[i][k]*mut)%MOD+MOD)
                        %MOD;
            }
            res = (res * mat[i][i])%MOD;
        }
        return res;
    }
};

        Matrix ret;
        ret.init();
        for(int i = 0;i < n;i++)
            for(int j = 0;j < n;j++)
                if(i != j && g[i][j]){
                    ret.mat[i][j] = -1;
                    ret.mat[i][i]++;
                }
        printf("%d\n",ret.det(n-1));
```

计算生成树个数，不取模，SPOJ 104

```
const double eps = 1e-8;
const int MAXN = 110;
int sgn(double x){
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
double b[MAXN][MAXN];
double det(double a[][MAXN],int n){
    int i, j, k, sign = 0;
    double ret = 1;
    for(i = 0;i < n;i++)
        for(j = 0;j < n;j++)
            b[i][j] = a[i][j];
    for(i = 0;i < n;i++){
        if(sgn(b[i][i]) == 0){
            for(j = i + 1; j < n;j++)
                if(sgn(b[j][i]) != 0)
                    break;
            if(j == n)return 0;
            for(k = i;k < n;k++)
                swap(b[i][k],b[j][k]);
            sign++;
        }
        ret *= b[i][i];
        for(k = i + 1;k < n;k++)
            b[i][k]/=b[i][i];
        for(j = i+1;j < n;j++)
            for(k = i+1;k < n;k++)
                b[j][k] -= b[j][i]*b[i][k];
    }
    if(sign & 1)ret = -ret;
    return ret;
}
double a[MAXN][MAXN];
int g[MAXN][MAXN];
int main(){
    int T;
    int n,m;
    int u,v;
    scanf("%d",&T);
    while(T--){
        scanf("%d%d",&n,&m);
        memset(g,0,sizeof(g));
        while(m--){
            scanf("%d%d",&u,&v);
            u--;v--;
            g[u][v] = g[v][u] = 1;
        }
        memset(a,0,sizeof(a));
        for(int i = 0;i < n;i++)
            for(int j = 0;j < n;j++)
                if(i != j && g[i][j]){
                    a[i][i]++;
                    a[i][j] = -1;
                }
        double ans = det(a,n-1);
        printf("%.0lf\n",ans);
    }
    return 0;
}
```

4.16 最大流

4.16.1 SAP 邻接矩阵形式

```
/*
 * SAP 算法（矩阵形式）
 * 结点编号从 0 开始
 */
const int MAXN=1100;
int maze[MAXN][MAXN];
int gap[MAXN],dis[MAXN],pre[MAXN],cur[MAXN];
int sap(int start,int end,int nodenum){
    memset(cur,0,sizeof(cur));
    memset(dis,0,sizeof(dis));
    memset(gap,0,sizeof(gap));
    int u=pre[start]=start,maxflow=0,aug=-1;
    gap[0]=nodenum;
    while(dis[start]<nodenum){
        loop:
          for(int v=cur[u];v<nodenum;v++)
```

```
            if(maze[u][v] && dis[u]==dis[v]+1){
                if(aug==-1 || aug>maze[u][v])aug=maze[u][v];
                pre[v]=u;
                u=cur[u]=v;
                if(v==end){
                    maxflow+=aug;
                    for(u=pre[u];v!=start;v=u,u=pre[u]){
                        maze[u][v]-=aug;
                        maze[v][u]+=aug;
                    }
                    aug=-1;
                }
                goto loop;
            }
            int mindis=nodenum-1;
            for(int v=0;v<nodenum;v++)
               if(maze[u][v]&&mindis>dis[v]){
                   cur[u]=v;
                   mindis=dis[v];
               }
            if((--gap[dis[u]])==0)break;
            gap[dis[u]=mindis+1]++;
            u=pre[u];
    }
    return maxflow;
}
```

4.16.2 SAP 邻接矩阵形式 2

保留原矩阵，可用于多次使用最大流

```
/*
 * SAP 邻接矩阵形式
 * 点的编号从 0 开始
 * 增加个 flow 数组，保留原矩阵 maze, 可用于多次使用最大流
 */
const int MAXN=1100;
int maze[MAXN][MAXN];
int gap[MAXN],dis[MAXN],pre[MAXN],cur[MAXN];
int flow[MAXN][MAXN];//存最大流的容量
int sap(int start,int end,int nodenum){
    memset(cur,0,sizeof(cur));
    memset(dis,0,sizeof(dis));
    memset(gap,0,sizeof(gap));
    memset(flow,0,sizeof(flow));
    int u=pre[start]=start,maxflow=0,aug=-1;
    gap[0]=nodenum;
    while(dis[start]<nodenum){
        loop:
          for(int v=cur[u];v<nodenum;v++)
            if(maze[u][v]-flow[u][v] && dis[u]==dis[v]+1){
                if(aug==-1 || aug>maze[u][v]-flow[u][v])aug=maze[u
                   ][v]-flow[u][v];
                pre[v]=u;
                u=cur[u]=v;
                if(v==end){
                    maxflow+=aug;
                    for(u=pre[u];v!=start;v=u,u=pre[u]){
                        flow[u][v]+=aug;
                        flow[v][u]-=aug;
                    }
                    aug=-1;
                }
                goto loop;
            }
            int mindis=nodenum-1;
            for(int v=0;v<nodenum;v++)
               if(maze[u][v]-flow[u][v]&&mindis>dis[v]){
                   cur[u]=v;
                   mindis=dis[v];
               }
            if((--gap[dis[u]])==0)break;
            gap[dis[u]=mindis+1]++;
            u=pre[u];
    }
    return maxflow;
}
```

4.16.3 ISAP 邻接表形式

```
const int MAXN = 100010;//点数的最大值
const int MAXM = 400010;//边数的最大值
const int INF = 0x3f3f3f3f;
struct Edge{
    int to,next,cap,flow;
}edge[MAXM];//注意是 MAXM
int tol;
int head[MAXN];
int gap[MAXN],dep[MAXN],pre[MAXN],cur[MAXN];
void init(){
    tol = 0;
    memset(head,-1,sizeof(head));
}
//加边，单向图三个参数，双向图四个参数
void addedge(int u,int v,int w,int rw=0){
    edge[tol].to = v;edge[tol].cap = w;edge[tol].next = head[u];
    edge[tol].flow = 0;head[u] = tol++;
    edge[tol].to = u;edge[tol].cap = rw;edge[tol].next = head[v];
    edge[tol].flow = 0;head[v]=tol++;
}
//输入参数：起点、终点、点的总数
//点的编号没有影响，只要输入点的总数
int sap(int start,int end,int N){
    memset(gap,0,sizeof(gap));
    memset(dep,0,sizeof(dep));
    memcpy(cur,head,sizeof(head));
    int u = start;
    pre[u] = -1;
    gap[0] = N;
    int ans = 0;
    while(dep[start] < N){
        if(u == end){
            int Min = INF;
            for(int i = pre[u];i != -1; i = pre[edge[i^1].to])
                if(Min > edge[i].cap - edge[i].flow)
                        Min = edge[i].cap - edge[i].flow;
            for(int i = pre[u];i != -1; i = pre[edge[i^1].to]){
                edge[i].flow += Min;
                edge[i^1].flow -= Min;
            }
            u = start;
            ans += Min;
            continue;
        }
        bool flag = false;
        int v;
        for(int i = cur[u]; i != -1;i = edge[i].next){
            v = edge[i].to;
            if(edge[i].cap - edge[i].flow && dep[v]+1 == dep[u])
            {
                flag = true;
                cur[u] = pre[v] = i;
                break;
            }
        }
        if(flag){
            u = v;
            continue;
        }
        int Min = N;
        for(int i = head[u]; i != -1;i = edge[i].next)
            if(edge[i].cap - edge[i].flow && dep[edge[i].to] < Min)
               {
                Min = dep[edge[i].to];
                cur[u] = i;
            }
        gap[dep[u]]--;
        if(!gap[dep[u]])return ans;
        dep[u] = Min+1;
        gap[dep[u]]++;
        if(u != start) u = edge[pre[u]^1].to;
    }
    return ans;
}
```

4.16.4 ISAP+bfs 初始化 + 栈优化

```
const int MAXN = 100010;//点数的最大值
const int MAXM = 400010;//边数的最大值
const int INF = 0x3f3f3f3f;
struct Edge{
    int to,next,cap,flow;
}edge[MAXM];//注意是 MAXM
int tol;
int head[MAXN];
int gap[MAXN],dep[MAXN],cur[MAXN];
void init(){
    tol = 0;
    memset(head,-1,sizeof(head));
}
void addedge(int u,int v,int w,int rw = 0){
    edge[tol].to = v; edge[tol].cap = w; edge[tol].flow = 0;
    edge[tol].next = head[u]; head[u] = tol++;
    edge[tol].to = u; edge[tol].cap = rw; edge[tol].flow = 0;
    edge[tol].next = head[v]; head[v] = tol++;
}
int Q[MAXN];
void BFS(int start,int end){
    memset(dep,-1,sizeof(dep));
    memset(gap,0,sizeof(gap));
    gap[0] = 1;
    int front = 0, rear = 0;
    dep[end] = 0;
    Q[rear++] = end;
    while(front != rear){
        int u = Q[front++];
        for(int i = head[u]; i != -1; i = edge[i].next){
            int v = edge[i].to;
            if(dep[v] != -1)continue;
            Q[rear++] = v;
            dep[v] = dep[u] + 1;
            gap[dep[v]]++;
        }
    }
}
int S[MAXN];
int sap(int start,int end,int N){
    BFS(start,end);
    memcpy(cur,head,sizeof(head));
    int top = 0;
    int u = start;
    int ans = 0;
    while(dep[start] < N){
        if(u == end){
            int Min = INF;
            int inser;
            for(int i = 0;i < top;i++)
                if(Min > edge[S[i]].cap - edge[S[i]].flow){
                    Min = edge[S[i]].cap - edge[S[i]].flow;
                    inser = i;
                }
            for(int i = 0;i < top;i++){
                edge[S[i]].flow += Min;
                edge[S[i]^1].flow -= Min;
            }
            ans += Min;
            top = inser;
            u = edge[S[top]^1].to;
            continue;
        }
        bool flag = false;
        int v;
        for(int i = cur[u]; i != -1; i = edge[i].next){
            v = edge[i].to;
            if(edge[i].cap - edge[i].flow && dep[v]+1 == dep[u]){
                flag = true;
                cur[u] = i;
                break;
            }
        }
        if(flag){
            S[top++] = cur[u];
            u = v;
            continue;
        }
        int Min = N;
        for(int i = head[u]; i != -1; i = edge[i].next)
            if(edge[i].cap - edge[i].flow && dep[edge[i].to] < Min)
               {
                Min = dep[edge[i].to];
                cur[u] = i;
            }
        gap[dep[u]]--;
        if(!gap[dep[u]])return ans;
        dep[u] = Min + 1;
        gap[dep[u]]++;
        if(u != start)u = edge[S[--top]^1].to;
    }
    return ans;
}
```

4.16.5 dinic

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

```
}edge[MAXM];//注意是 MAXM
int tol;
int head[MAXN];
void init(){
    tol = 2;
    memset(head,-1,sizeof(head));
}
void addedge(int u,int v,int w,int rw = 0){
    edge[tol].to = v; edge[tol].cap = w; edge[tol].flow = 0;
    edge[tol].next = head[u]; head[u] = tol++;
    edge[tol].to = u; edge[tol].cap = rw; edge[tol].flow = 0;
    edge[tol].next = head[v]; head[v] = tol++;
}
int Q[MAXN];
int dep[MAXN],cur[MAXN],sta[MAXN];
bool bfs(int s,int t,int n){
    int front = 0,tail = 0;
    memset(dep,-1,sizeof(dep[0])*(n+1));
    dep[s] = 0;
    Q[tail++] = s;
    while(front < tail){
        int u = Q[front++];
        for(int i = head[u]; i != -1;i = edge[i].next){
            int v = edge[i].to;
            if(edge[i].cap > edge[i].flow && dep[v] == -1){
                dep[v] = dep[u] + 1;
                if(v == t)return true;
                Q[tail++] = v;
            }
        }
    }
    return false;
}
int dinic(int s,int t,int n){
    int maxflow = 0;
    while(bfs(s,t,n)){
        for(int i = 0;i < n;i++)cur[i] = head[i];
        int u = s, tail = 0;
        while(cur[s] != -1){
            if(u == t){
                int tp = INF;
                for(int i = tail-1;i >= 0;i--)
                    tp = min(tp,edge[sta[i]].cap-edge[sta[i]].flow)
                      ;
                maxflow += tp;
                for(int i = tail-1;i >= 0;i--){
                    edge[sta[i]].flow += tp;
                    edge[sta[i]^1].flow -= tp;
                    if(edge[sta[i]].cap-edge[sta[i]].flow == 0)
                        tail = i;
                }
                u = edge[sta[tail]^1].to;
            }
            else if(cur[u] != -1 && edge[cur[u]].cap > edge[cur[u]].flow && dep[u] + 1 == dep[edge[cur[u]].to]){
                sta[tail++] = cur[u];
                u = edge[cur[u]].to;
            }
            else {
                while(u != s && cur[u] == -1)
                    u = edge[sta[--tail]^1].to;
                cur[u] = edge[cur[u]].next;
            }
        }
    }
    return maxflow;
}
```

4.16.6 最大流判断多解

```
//判断最大流多解就是在残留网络中找正环
bool vis[MAXN],no[MAXN];
int Stack[MAXN],top;
bool dfs(int u,int pre,bool flag){
    vis[u] = 1;
    Stack[top++] = u;
    for(int i = head[u];i != -1;i = edge[i].next)
    {
        int v = edge[i].to;
        if(edge[i].cap <= edge[i].flow)continue;
        if(v == pre)continue;
        if(!vis[v]){
            if(dfs(v,u,edge[i^1].flow < edge[i^1].cap))return true;
        }
        else if(!no[v])return true;
    }
    if(!flag){
        while(1){
            int v = Stack[--top];
            no[v] = true;
            if(v == u)break;
        }
    }
    return false;
}
//执行完最大流后可进行调用
        memset(vis,false,sizeof(vis));
        memset(no,false,sizeof(no));
        top = 0;
        bool flag = dfs(end,end,0);
```

4.17 最小费用最大流

4.17.1 SPFA 版费用流

最小费用最大流，求最大费用只需要取相反数，结果取相反数即可。
点的总数为 N，点的编号 0 ∼ N-1

```
const int MAXN = 10000;
const int MAXM = 100000;
const int INF = 0x3f3f3f3f;
struct Edge{
    int to,next,cap,flow,cost;
}edge[MAXM];
int head[MAXN],tol;
int pre[MAXN],dis[MAXN];
bool vis[MAXN];
int N;//节点总个数，节点编号从 0~N-1
void init(int n){
    N = n;
    tol = 0;
    memset(head,-1,sizeof(head));
}
void addedge(int u,int v,int cap,int cost){
    edge[tol].to = v;
    edge[tol].cap = cap;
    edge[tol].cost = cost;
    edge[tol].flow = 0;
    edge[tol].next = head[u];
    head[u] = tol++;
    edge[tol].to = u;
    edge[tol].cap = 0;
    edge[tol].cost = -cost;
    edge[tol].flow = 0;
    edge[tol].next = head[v];
    head[v] = tol++;
}
bool spfa(int s,int t){
    queue<int>q;
    for(int i = 0;i < N;i++){
        dis[i] = INF;
        vis[i] = false;
        pre[i] = -1;
    }
    dis[s] = 0;
    vis[s] = true;
    q.push(s);
    while(!q.empty()){
        int u = q.front();
        q.pop();
        vis[u] = false;
        for(int i = head[u]; i != -1;i = edge[i].next){
            int v = edge[i].to;
            if(edge[i].cap > edge[i].flow && dis[v] > dis[u] + edge
               [i].cost )
            {
                dis[v] = dis[u] + edge[i].cost;
                pre[v] = i;
                if(!vis[v]){
                    vis[v] = true;
                    q.push(v);
                }
            }
        }
    }
    if(pre[t] == -1)return false;
    else return true;
}
//返回的是最大流，cost 存的是最小费用
int minCostMaxflow(int s,int t,int &cost){
    int flow = 0;
    cost = 0;
    while(spfa(s,t)){
        int Min = INF;
        for(int i = pre[t];i != -1;i = pre[edge[i^1].to]){
            if(Min > edge[i].cap - edge[i].flow)
                Min = edge[i].cap - edge[i].flow;
        }
        for(int i = pre[t];i != -1;i = pre[edge[i^1].to]){
            edge[i].flow += Min;
            edge[i^1].flow -= Min;
            cost += edge[i].cost * Min;
        }
        flow += Min;
    }
    return flow;
}
```

4.17.2 zkw 费用流

对于二分图类型的比较高效

```
const int MAXN = 100;
const int MAXM = 20000;
const int INF = 0x3f3f3f3f;
struct Edge{
    int to,next,cap,flow,cost;
    Edge(int _to = 0,int _next = 0,int _cap = 0,int _flow = 0,int
       _cost = 0):
        to(_to),next(_next),cap(_cap),flow(_flow),cost(_cost){}
}edge[MAXM];
struct ZKW_MinCostMaxFlow{
    int head[MAXN],tot;
    int cur[MAXN];
    int dis[MAXN];
    bool vis[MAXN];
    int ss,tt,N;//源点、汇点和点的总个数（编号是 0~N-1），不需要额外赋值，
       调用会直接赋值
    int min_cost, max_flow;
    void init(){
        tot = 0;
        memset(head,−1,sizeof(head));
    }
    void addedge(int u,int v,int cap,int cost){
        edge[tot] = Edge(v,head[u],cap,0,cost);
        head[u] = tot++;
        edge[tot] = Edge(u,head[v],0,0,−cost);
        head[v] = tot++;
    }
    int aug(int u,int flow){
        if(u == tt)return flow;
        vis[u] = true;
        for(int i = cur[u];i != −1;i = edge[i].next){
            int v = edge[i].to;
            if(edge[i].cap > edge[i].flow && !vis[v] && dis[u] ==
               dis[v] + edge[i].cost){
                int tmp = aug(v,min(flow,edge[i].cap−edge[i].flow))
                    ;
                edge[i].flow += tmp;
                edge[i^1].flow −= tmp;
                cur[u] = i;
                if(tmp)return tmp;
            }
        }
        return 0;
    }
    bool modify_label(){
        int d = INF;
        for(int u = 0;u < N;u++)
            if(vis[u])
                for(int i = head[u];i != −1;i = edge[i].next){
                    int v = edge[i].to;
                    if(edge[i].cap>edge[i].flow && !vis[v])
                        d = min(d,dis[v]+edge[i].cost−dis[u]);
                }
        if(d == INF)return false;
        for(int i = 0;i < N;i++)
            if(vis[i]){
                vis[i] = false;
                dis[i] += d;

            }
        return true;
    }
    /*
     * 直接调用获取最小费用和最大流
     * 输入: start-源点, end-汇点, n-点的总个数（编号从 0 开始）
     * 返回值: pair<int,int> 第一个是最小费用, 第二个是最大流
     */
    pair<int,int> mincostmaxflow(int start,int end,int n){
        ss = start, tt = end, N = n;
        min_cost = max_flow = 0;
        for(int i = 0;i < n;i++)dis[i] = 0;
        while(1){
            for(int i = 0;i < n;i++)cur[i] = head[i];
            while(1){
                for(int i = 0;i < n;i++)vis[i] = false;
                int tmp = aug(ss,INF);
                if(tmp == 0)break;
                max_flow += tmp;
                min_cost += tmp*dis[ss];
            }
            if(!modify_label())break;
        }
        return make_pair(min_cost,max_flow);
    }
}solve;
```

4.18 2-SAT

4.18.1 染色法（可以得到字典序最小的解）

HDU 1814

```
const int MAXN = 20020;
const int MAXM = 100010;
struct Edge
{
    int to,next;
}edge[MAXM];
int head[MAXN],tot;
void init()
{
    tot = 0;
    memset(head,-1,sizeof(head));
}
void addedge(int u,int v)
{
    edge[tot].to = v;edge[tot].next = head[u];head[u] = tot++;
}
bool vis[MAXN];//染色标记, 为 true 表示选择
int S[MAXN],top;//栈
bool dfs(int u)
{
    if(vis[u^1])return false;
    if(vis[u])return true;
    vis[u] = true;
    S[top++] = u;
    for(int i = head[u];i != -1;i = edge[i].next)
        if(!dfs(edge[i].to))
            return false;
    return true;
}
bool Twosat(int n)
{
    memset(vis,false,sizeof(vis));
    for(int i = 0;i < n;i += 2)
    {
        if(vis[i] || vis[i^1])continue;
        top = 0;
        if(!dfs(i))
        {
            while(top)vis[S[--top]] = false;
            if(!dfs(i^1)) return false;
        }
    }
    return true;
}
int main()
{
    int n,m;
    int u,v;
    while(scanf("%d%d",&n,&m) == 2)
    {
        init();
        while(m--)
        {
            scanf("%d%d",&u,&v);
            u--;v--;
            addedge(u,v^1);
            addedge(v,u^1);
        }
        if(Twosat(2*n))
        {
            for(int i = 0;i < 2*n;i++)
                if(vis[i])
                   printf("%d\n",i+1);
        }
        else printf("NIE\n");
    }
    return 0;
}
```

4.18.2 强连通缩点法（拓扑排序只能得到任意解）

POJ 3648 Wedding

```
//******************************************
//2-SAT 强连通缩点
const int MAXN = 1010;
const int MAXM = 100010;
struct Edge
{
    int to,next;
}edge[MAXM];
int head[MAXN],tot;
void init()
{
    tot = 0;
    memset(head,-1,sizeof(head));
}
void addedge(int u,int v)
{
    edge[tot].to = v; edge[tot].next = head[u]; head[u] = tot++;
}
int Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN];//Belong 数组的值 1
   ~ scc
int Index,top;
int scc;
bool Instack[MAXN];
int num[MAXN];
void Tarjan(int u)
{
    int v;
    Low[u] = DFN[u] = ++Index;
    Stack[top++] = u;
    Instack[u] = true;
    for(int i = head[u];i != -1;i = edge[i].next)
    {
        v = edge[i].to;
        if( !DFN[v] )
        {
            Tarjan(v);
            if(Low[u] > Low[v])Low[u] = Low[v];
        }
        else if(Instack[v] && Low[u] > DFN[v])
            Low[u] = DFN[v];
    }
    if(Low[u] == DFN[u])
    {
        scc++;
        do
        {
            v = Stack[--top];
            Instack[v] = false;
            Belong[v] = scc;
            num[scc]++;
        }
        while(v != u);
    }
}
```

```
bool solvable(int n)//n 是总个数, 需要选择一半
{
    memset(DFN,0,sizeof(DFN));
    memset(Instack,false,sizeof(Instack));
    memset(num,0,sizeof(num));
    Index = scc = top = 0;
    for(int i = 0;i < n;i++)
        if(!DFN[i])
            Tarjan(i);
    for(int i = 0;i < n;i += 2)
    {
        if(Belong[i] == Belong[i^1])
            return false;
    }
    return true;
}
//**************************************************

//拓扑排序求任意一组解部分
queue<int>q1,q2;
vector<vector<int> > dag;//缩点后的逆向 DAG 图
char color[MAXN];//染色, 为'R' 是选择的
int indeg[MAXN];//入度
int cf[MAXN];
void solve(int n)
{
    dag.assign(scc+1,vector<int>());
    memset(indeg,0,sizeof(indeg));
    memset(color,0,sizeof(color));
    for(int u = 0;u < n;u++)
        for(int i = head[u];i != -1;i = edge[i].next)
        {
            int v = edge[i].to;
            if(Belong[u] != Belong[v])
            {
                dag[Belong[v]].push_back(Belong[u]);
                indeg[Belong[u]]++;
            }
        }
    for(int i = 0;i < n;i += 2)
    {
        cf[Belong[i]] = Belong[i^1];
        cf[Belong[i^1]] = Belong[i];
    }
    while(!q1.empty())q1.pop();
    while(!q2.empty())q2.pop();
    for(int i = 1;i <= scc;i++)
        if(indeg[i] == 0)
            q1.push(i);
    while(!q1.empty())
    {
        int u = q1.front();
        q1.pop();
        if(color[u] == 0)
        {
            color[u] = 'R';
            color[cf[u]] = 'B';
        }
        int sz = dag[u].size();
        for(int i = 0;i < sz;i++)
        {
            indeg[dag[u][i]]--;
            if(indeg[dag[u][i]] == 0)
                q1.push(dag[u][i]);
        }
    }
}

int change(char s[])
{
    int ret = 0;
    int i = 0;
    while(s[i]>='0' && s[i]<='9')
    {
        ret *= 10;
        ret += s[i]-'0';
        i++;
    }
    if(s[i] == 'w')return 2*ret;
    else return 2*ret+1;
}
int main()
{
    int n,m;
    char s1[10],s2[10];
    while(scanf("%d%d",&n,&m) == 2)
    {
        if(n == 0 && m == 0)break;
        init();
        while(m--)
        {
            scanf("%s%s",s1,s2);
            int u = change(s1);
            int v = change(s2);
            addedge(u^1,v);
            addedge(v^1,u);
        }
        addedge(1,0);
        if(solvable(2*n))
        {
            solve(2*n);
            for(int i = 1;i < n;i++)
            {
                //注意这一定是判断 color[Belong]
                if(color[Belong[2*i]] == 'R')printf("%dw",i);
                else printf("%dh",i);
                if(i < n-1)printf("␣");
                else printf("\n");
            }
        }
        else printf("bad␣luck\n");
    }
    return 0;
}
```

4.19 曼哈顿最小生成树

POJ 3241 求曼哈顿最小生成树上第 k 大的边

```
const int MAXN = 100010;
const int INF = 0x3f3f3f3f;
struct Point{
    int x,y,id;
}p[MAXN];
bool cmp(Point a,Point b){
    if(a.x != b.x) return a.x < b.x;
    else return a.y < b.y;
}
//树状数组，找 y-x 大于当前的，但是 y+x 最小的
struct BIT{
    int min_val,pos;
    void init()
    {
        min_val = INF;
        pos = -1;
    }
}bit[MAXN];
//所有有效边
struct Edge{
    int u,v,d;
}edge[MAXN<<2];
bool cmpedge(Edge a,Edge b){
    return a.d < b.d;
}
int tot;
int n;
int F[MAXN];
int find(int x){
    if(F[x] == -1) return x;
    else return F[x] = find(F[x]);
}
void addedge(int u,int v,int d){
    edge[tot].u = u;
    edge[tot].v = v;
    edge[tot++].d = d;
}
int lowbit(int x){
    return x&(-x);
}
void update(int i,int val,int pos){
    while(i > 0){
        if(val < bit[i].min_val){
            bit[i].min_val = val;
            bit[i].pos = pos;
        }
        i -= lowbit(i);
    }
}
//查询 [i,m] 的最小值位置
int ask(int i,int m){
    int min_val = INF,pos = -1;
    while(i <= m){
        if(bit[i].min_val < min_val){
            min_val = bit[i].min_val;
            pos = bit[i].pos;
        }
        i += lowbit(i);
    }
    return pos;
}
int dist(Point a,Point b){
    return abs(a.x - b.x) + abs(a.y - b.y);
}
void Manhattan_minimum_spanning_tree(int n,Point p[]){
    int a[MAXN],b[MAXN];
    tot = 0;
    for(int dir = 0; dir < 4;dir++){
        //4 种坐标变换
        if(dir == 1 || dir == 3){
            for(int i = 0;i < n;i++)
                swap(p[i].x,p[i].y);
        }
        else if(dir == 2){
            for(int i = 0;i < n;i++)
                p[i].x = -p[i].x;
        }
        sort(p,p+n,cmp);
        for(int i = 0;i < n;i++)
            a[i] = b[i] = p[i].y - p[i].x;
        sort(b,b+n);
        int m = unique(b,b+n) - b;
        for(int i = 1;i <= m;i++)
            bit[i].init();
        for(int i = n-1 ;i >= 0;i--){
            int pos = lower_bound(b,b+m,a[i]) - b + 1;
```

```
                int ans = ask(pos,m);
                if(ans != -1)
                    addedge(p[i].id,p[ans].id,dist(p[i],p[ans]));
                update(pos,p[i].x+p[i].y,i);
            }
        }
}
int solve(int k){
    Manhattan_minimum_spanning_tree(n,p);
    memset(F,-1,sizeof(F));
    sort(edge,edge+tot,cmpedge);
    for(int i = 0;i < tot;i++){
        int u = edge[i].u;
        int v = edge[i].v;
        int t1 = find(u), t2 = find(v);
        if(t1 != t2){
            F[t1] = t2;
            k--;
            if(k == 0)return edge[i].d;
        }
    }
}
int main()
{
    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);
    int k;
    while(scanf("%d%d",&n,&k)==2 && n){
        for(int i = 0;i < n;i++){
            scanf("%d%d",&p[i].x,&p[i].y);
            p[i].id = i;
        }
        printf("%d\n",solve(n-k));
    }
    return 0;
}
```

4.20 LCA

4.20.1 dfs+ST 在线算法

```
/*
 * LCA (POJ 1330)
 * 在线算法 DFS + ST
 */
const int MAXN = 10010;
int rmq[2*MAXN];//rmq 数组，就是欧拉序列对应的深度序列
struct ST{
    int mm[2*MAXN];
    int dp[2*MAXN][20];//最小值对应的下标
    void init(int n){
        mm[0] = -1;
        for(int i = 1;i <= n;i++){
            mm[i] = ((i&(i-1)) == 0)?mm[i-1]+1:mm[i-1];
            dp[i][0] = i;
        }
        for(int j = 1; j <= mm[n];j++)
            for(int i = 1; i + (1<<j) - 1 <= n; i++)
                dp[i][j] = rmq[dp[i][j-1]] < rmq[dp[i+(1<<(j-1))][j
                    -1]]?dp[i][j-1]:dp[i+(1<<(j-1))][j-1];
    }
    //查询 [a,b] 之间最小值的下标
    int query(int a,int b)
    {
        if(a > b)swap(a,b);
        int k = mm[b-a+1];
        return rmq[dp[a][k]] <= rmq[dp[b-(1<<k)+1][k]]?dp[a][k]:dp[
            b-(1<<k)+1][k];
    }
};
//边的结构体定义
struct Edge{
    int to,next;
};
Edge edge[MAXN*2];
int tot,head[MAXN];

int F[MAXN*2];//欧拉序列，就是 dfs 遍历的顺序，长度为 2*n-1, 下标从 1 开始
int P[MAXN];//P[i] 表示点 i 在 F 中第一次出现的位置
int cnt;
ST st;
void init(){
    tot = 0;
    memset(head,-1,sizeof(head));
}
//加边，无向边需要加两次
void addedge(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
void dfs(int u,int pre,int dep){
    F[++cnt] = u;
    rmq[cnt] = dep;
    P[u] = cnt;
    for(int i = head[u];i != -1;i = edge[i].next){
        int v = edge[i].to;
        if(v == pre)continue;
        dfs(v,u,dep+1);
        F[++cnt] = u;
        rmq[cnt] = dep;
    }
```

```
}
//查询 LCA 前的初始化
void LCA_init(int root,int node_num){
    cnt = 0;
    dfs(root,root,0);
    st.init(2*node_num-1);
}
//查询 u,v 的 lca 编号
int query_lca(int u,int v){
    return F[st.query(P[u],P[v])];
}
bool flag[MAXN];
int main(){
    int T;
    int N;
    int u,v;
    scanf("%d",&T);
    while(T--){
        scanf("%d",&N);
        init();
        memset(flag,false,sizeof(flag));
        for(int i = 1; i < N;i++){
            scanf("%d%d",&u,&v);
            addedge(u,v);
            addedge(v,u);
            flag[v] = true;
        }
        int root;
        for(int i = 1; i <= N;i++)
            if(!flag[i]){
                root = i;
                break;
            }
        LCA_init(root,N);
        scanf("%d%d",&u,&v);
        printf("%d\n",query_lca(u,v));
    }
    return 0;
}
```

4.20.2 离线 Tarjan 算法

```
/*
 * POJ 1470
 * 给出一颗有向树，Q 个查询
 * 输出查询结果中每个点出现次数
 */
/*
 * 离线算法，LCATarjan
 * 复杂度O(n+Q);
 */
const int MAXN = 1010;
const int MAXQ = 500010;//查询数的最大值

//并查集部分
int F[MAXN];//需要初始化为 -1
int find(int x){
    if(F[x] == −1)return x;
    return F[x] = find(F[x]);
}
void bing(int u,int v){
    int t1 = find(u);
    int t2 = find(v);
    if(t1 != t2)
        F[t1] = t2;
}
//************************
bool vis[MAXN];//访问标记
int ancestor[MAXN];//祖先
struct Edge{
    int to,next;
}edge[MAXN*2];
int head[MAXN],tot;
void addedge(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}

struct Query{
    int q,next;
    int index;//查询编号
}query[MAXQ*2];
int answer[MAXQ];//存储最后的查询结果，下标 0 Q-1
int h[MAXQ];
int tt;
int Q;

void add_query(int u,int v,int index){
    query[tt].q = v;
    query[tt].next = h[u];
    query[tt].index = index;
    h[u] = tt++;
    query[tt].q = u;
    query[tt].next = h[v];
    query[tt].index = index;
    h[v] = tt++;
}

void init(){
    tot = 0;
    memset(head,−1,sizeof(head));
    tt = 0;
    memset(h,-1,sizeof(h));
    memset(vis,false,sizeof(vis));
    memset(F,-1,sizeof(F));
    memset(ancestor,0,sizeof(ancestor));
}
void LCA(int u){
    ancestor[u] = u;
    vis[u] = true;
    for(int i = head[u];i != -1;i = edge[i].next){
        int v = edge[i].to;
        if(vis[v])continue;
        LCA(v);
        bing(u,v);
        ancestor[find(u)] = u;
    }
    for(int i = h[u];i != -1;i = query[i].next){
        int v = query[i].q;
        if(vis[v]){
            answer[query[i].index] = ancestor[find(v)];
        }
    }
}
bool flag[MAXN];
int Count_num[MAXN];
int main(){
    int n;
    int u,v,k;
    while(scanf("%d",&n) == 1){
        init();
        memset(flag,false,sizeof(flag));
        for(int i = 1;i <= n;i++){
            scanf("%d:(%d)",&u,&k);
            while(k--){
                scanf("%d",&v);
                flag[v] = true;
                addedge(u,v);
                addedge(v,u);
            }
        }
        scanf("%d",&Q);
        for(int i = 0;i < Q;i++){
            char ch;
            cin>>ch;
            scanf("%d␣%d)",&u,&v);
            add_query(u,v,i);
        }
        int root;
        for(int i = 1;i <= n;i++)
            if(!flag[i]){
                root = i;
                break;
            }
        LCA(root);
        memset(Count_num,0,sizeof(Count_num));
        for(int i = 0;i < Q;i++)
            Count_num[answer[i]]++;
        for(int i = 1;i <= n;i++)
            if(Count_num[i] > 0)
                printf("%d:%d\n",i,Count_num[i]);
    }
    return 0;
}
```

4.20.3 LCA 倍增法

```
/*
 * POJ 1330
 * LCA 在线算法
 */
const int MAXN = 10010;
const int DEG = 20;

struct Edge{
    int to,next;
}edge[MAXN*2];
int head[MAXN],tot;
void addedge(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
void init(){
    tot = 0;
    memset(head,-1,sizeof(head));
}
int fa[MAXN][DEG];//fa[i][j] 表示结点 i 的第2^j个祖先
int deg[MAXN];//深度数组

void BFS(int root){
    queue<int>que;
    deg[root] = 0;
    fa[root][0] = root;
    que.push(root);
    while(!que.empty()){
        int tmp = que.front();
        que.pop();
        for(int i = 1;i < DEG;i++)
            fa[tmp][i] = fa[fa[tmp][i-1]][i-1];
        for(int i = head[tmp]; i != -1;i = edge[i].next){
            int v = edge[i].to;
            if(v == fa[tmp][0])continue;
            deg[v] = deg[tmp] + 1;
            fa[v][0] = tmp;
```

```
                que.push(v);
            }

    }
}
int LCA(int u,int v){
    if(deg[u] > deg[v])swap(u,v);
    int hu = deg[u], hv = deg[v];
    int tu = u, tv = v;
    for(int det = hv-hu, i = 0; det ;det>>=1, i++)
        if(det&1)
            tv = fa[tv][i];
    if(tu == tv)return tu;
    for(int i = DEG-1; i >= 0; i--){
        if(fa[tu][i] == fa[tv][i])
            continue;
        tu = fa[tu][i];
        tv = fa[tv][i];
    }
    return fa[tu][0];
}
bool flag[MAXN];
int main(){
    int T;
    int n;
    int u,v;
    scanf("%d",&T);
    while(T--){
        scanf("%d",&n);
        init();
        memset(flag,false,sizeof(flag));
        for(int i = 1;i < n;i++){
            scanf("%d%d",&u,&v);
            addedge(u,v);
            addedge(v,u);
            flag[v] = true;
        }
        int root;
        for(int i = 1;i <= n;i++)
            if(!flag[i]){
                root = i;
                break;
            }
        BFS(root);
        scanf("%d%d",&u,&v);
        printf("%d\n",LCA(u,v));
    }
    return 0;
}
```

4.21 欧拉路

欧拉回路：每条边只经过一次，而且回到起点
欧拉路径：每条边只经过一次，不要求回到起点

欧拉回路判断：
无向图：连通（不考虑度为 0 的点），每个顶点度数都为偶数。
有向图：基图连通（把边当成无向边，同样不考虑度为 0 的点），每个顶点出度等于入度。
混合图（有无向边和有向边）：首先是基图连通（不考虑度为 0 的点），然后需要借助网络流判定。
首先给原图中的每条无向边随便指定一个方向（称为初始定向），将原图改为有向图 G'，然后的任务就是改变 G' 中某些边的方向（当然是无向边转化来的，原混合图中的有向边不能动）使其满足每个点的入度等于出度。
设 D[i] 为 G' 中 (点 i 的出度 - 点 i 的入度)。可以发现，在改变 G' 中边的方向的过程中，任何点的 D 值的奇偶性都不会发生改变（设将边 <i, j> 改为 <j, i>，则 i 入度加 1 出度减 1，j 入度减 1 出度加 1，两者之差加 2 或减 2，奇偶性不变）！而最终要求的是每个点的入度等于出度，即每个点的 D 值都为 0，是偶数，故可得：若初始定向得到的 G' 中任意一个点的 D 值是奇数，那么原图中一定不存在欧拉环！

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

欧拉路径的判断：
无向图：连通（不考虑度为 0 的点），每个顶点度数都为偶数或者仅有两个点的度数为偶数。
有向图：基图连通（把边当成无向边，同样不考虑度为 0 的点），每个顶点出度等于入度或者有且仅有一个点的出度比入度多 1，有且仅有一个点的出度比入度少 1，其余出度等于入度。
混合图：如果存在欧拉回路，一点存在欧拉路径了。否则如果有且仅有两个点的（出度 -入度）是奇数，那么给这个两个点加边，判断是否存在欧拉回路。

4.21.1 有向图

POJ 2337
给出 n 个小写字母组成的单词，要求将 n 个单词连接起来，使得前一个单词的最后一个字母和后一个单词的第一个字母相同。输出字典序最小的解。

```
struct Edge{
    int to,next;
    int index;
    bool flag;
}edge[2010];
int head[30],tot;
void init(){
    tot = 0;
    memset(head,−1,sizeof(head));
}
void addedge(int u,int v,int index){
    edge[tot].to = v;
    edge[tot].next = head[u];
    edge[tot].index = index;
    edge[tot].flag = false;
    head[u] = tot++;
}
string str[1010];
int in[30],out[30];
int cnt;
int ans[1010];
void dfs(int u){
    for(int i = head[u] ;i != -1;i = edge[i].next)
        if(!edge[i].flag)
        {
            edge[i].flag = true;
            dfs(edge[i].to);
            ans[cnt++] = edge[i].index;
        }
}
int main(){
    int T,n;
    scanf("%d",&T);
    while(T--){
        scanf("%d",&n);
        for(int i = 0;i < n;i++)
            cin>>str[i];
        sort(str,str+n);//要输出字典序最小的解，先按照字典序排序
        init();
        memset(in,0,sizeof(in));
        memset(out,0,sizeof(out));
        int start = 100;
        for(int i = n-1;i >= 0;i--)//字典序大的先加入
        {
            int u = str[i][0] - 'a';
            int v = str[i][str[i].length() - 1] - 'a';
            addedge(u,v,i);
            out[u]++;
            in[v]++;
            if(u < start)start = u;
            if(v < start)start = v;
        }
        int cc1 = 0, cc2 = 0;
        for(int i = 0;i < 26;i++){
            if(out[i] - in[i] == 1){
                cc1++;
                start = i;//如果有一个出度比入度大 1 的点，就从这个点出发，
                   否则从最小的点出发
            }
            else if(out[i] - in[i] == -1)
                cc2++;
            else if(out[i] - in[i] != 0)
                cc1 = 3;
        }
        if(! ( (cc1 == 0 && cc2 == 0) || (cc1 == 1 && cc2 == 1) )){
            printf("***\n");
            continue;
        }
        cnt = 0;
        dfs(start);
        if(cnt != n)//判断是否连通
        {
            printf("***\n");
            continue;
        }
        for(int i = cnt-1; i >= 0;i--){
            cout<<str[ans[i]];
            if(i > 0)printf(".");
            else printf("\n");
        }
    }
    return 0;
}
```

4.21.2 无向图

SGU 101

```
struct Edge{
    int to,next;
    int index;
    int dir;
    bool flag;
}edge[220];
int head[10],tot;
void init(){
    memset(head,-1,sizeof(head));
    tot = 0;
}
void addedge(int u,int v,int index){
    edge[tot].to = v;
    edge[tot].next = head[u];
    edge[tot].index = index;
    edge[tot].dir = 0;
    edge[tot].flag = false;
    head[u] = tot++;
    edge[tot].to = u;
    edge[tot].next = head[v];
    edge[tot].index = index;
    edge[tot].dir = 1;
    edge[tot].flag = false;
    head[v] = tot++;
}
int du[10];
vector<int>ans;
```

```
void dfs(int u){
    for(int i = head[u]; i != -1;i = edge[i].next)
        if(!edge[i].flag ){
            edge[i].flag = true;
            edge[i^1].flag = true;
            dfs(edge[i].to);
            ans.push_back(i);
        }
}
int main(){
    int n;
    while(scanf("%d",&n) == 1){
        init();
        int u,v;
        memset(du,0,sizeof(du));
        for(int i = 1;i <= n;i++){
            scanf("%d%d",&u,&v);
            addedge(u,v,i);
            du[u]++;
            du[v]++;
        }
        int s = -1;
        int cnt = 0;
        for(int i = 0;i <= 6;i++){
            if(du[i]&1) {cnt++; s = i;}
            if(du[i] > 0 && s == -1)
                s = i;
        }
        bool ff = true;
        if(cnt != 0 && cnt != 2){
            printf("No␣solution\n");
            continue;
        }
        ans.clear();
        dfs(s);
        if(ans.size() != n){
            printf("No␣solution\n");
            continue;
        }
        for(int i = 0;i < ans.size();i++){
            printf("%d␣",edge[ans[i]].index);
            if(edge[ans[i]].dir == 0)printf("-\n");
            else printf("+\n");
        }
    }
    return 0;
}
```

4.21.3 混合图

POJ 1637（本题保证了连通，故不需要判断连通，否则要判断连通）

```
const int MAXN = 210;
//最大流部分ISAP
const int MAXM = 20100;
const int INF = 0x3f3f3f3f;
struct Edge
{
    int to,next,cap,flow;
}edge[MAXM];
int tol;
int head[MAXN];
int gap[MAXN],dep[MAXN],pre[MAXN],cur[MAXN];
void init()
{
    tol = 0;
    memset(head,-1,sizeof(head));
}
void addedge(int u,int v,int w,int rw = 0)
{
    edge[tol].to = v;
    edge[tol].cap = w;
    edge[tol].next = head[u];
    edge[tol].flow = 0;
    head[u] = tol++;
    edge[tol].to = u;
    edge[tol].cap = rw;
    edge[tol].next = head[v];
    edge[tol].flow = 0;
    head[v] = tol++;
}
int sap(int start,int end,int N)
{
    memset(gap,0,sizeof(gap));
    memset(dep,0,sizeof(dep));
    memcpy(cur,head,sizeof(head));
    int u = start;
    pre[u] = -1;
    gap[0] = N;
    int ans = 0;
    while(dep[start] < N)
    {
        if(u == end)
        {
            int Min = INF;
            for(int i = pre[u]; i != -1;i = pre[edge[i^1].to])
                if(Min > edge[i].cap - edge[i].flow)
                    Min = edge[i].cap - edge[i].flow;
            for(int i = pre[u];i != -1;i = pre[edge[i^1].to])
            {
                edge[i].flow += Min;
                edge[i^1].flow -= Min;
            }
            u = start;
            ans += Min;
            continue;
        }
        bool flag = false;
        int v;
        for(int i = cur[u];i != -1;i = edge[i].next)
        {
            v = edge[i].to;
            if(edge[i].cap - edge[i].flow && dep[v] + 1 == dep[u])
            {
                flag = true;
                cur[u] = pre[v] = i;
                break;
            }
        }
        if(flag)
        {
            u = v;
            continue;
        }
        int Min = N;
        for(int i = head[u];i != -1;i = edge[i].next)
            if(edge[i].cap - edge[i].flow && dep[edge[i].to] < Min)
            {
                Min = dep[edge[i].to];
                cur[u] = i;
            }
        gap[dep[u]]--;
        if(!gap[dep[u]])return ans;
        dep[u] = Min+1;
        gap[dep[u]]++;
        if(u != start)u = edge[pre[u]^1].to;
    }
    return ans;
}
//the end of 最大流部分

int in[MAXN],out[MAXN];//每个点的出度和入度

int main()
{
    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);
    int T;
    int n,m;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d%d",&n,&m);
        init();
        int u,v,w;
        memset(in,0,sizeof(in));
        memset(out,0,sizeof(out));
        while(m--)
        {
            scanf("%d%d%d",&u,&v,&w);
            out[u]++; in[v]++;
            if(w == 0)//双向
                addedge(u,v,1);
        }
        bool flag = true;
        for(int i = 1;i <= n;i++)
        {
            if(out[i] - in[i] > 0)
                addedge(0,i,(out[i] - in[i])/2);
            else if(in[i] - out[i] > 0)
                addedge(i,n+1,(in[i] - out[i])/2);
            if((out[i] - in[i]) & 1)
                flag = false;
        }
        if(!flag)
        {
            printf("impossible\n");
            continue;
        }
        sap(0,n+1,n+2);
        for(int i = head[0]; i != -1;i = edge[i].next)
            if(edge[i].cap > 0 && edge[i].cap > edge[i].flow)
            {
                flag = false;
                break;
            }
        if(flag)printf("possible\n");
        else printf("impossible\n");
    }
    return 0;
}
```

4.22 树分治

4.22.1 点分治 -HDU5016

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

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

```
void init(){
    tot = 0;
    memset(head,-1,sizeof(head));
}
inline void addedge(int u,int v,int w){
    edge[tot].to = v;
    edge[tot].w = w;
    edge[tot].next = head[u];
    head[u] = tot++;
}
int size[MAXN],vis[MAXN],fa[MAXN],que[MAXN];
int TT;//时间戳
//找重心
inline int getroot(int u){
    int Min = MAXN, root = 0;
    int l,r;
    que[l = r = 1] = u;
    fa[u] = 0;
    for(;l <= r;l++)
        for(int i = head[que[l]]; i != -1;i = edge[i].next){
            int v = edge[i].to;
            if(v == fa[que[l]] || vis[v] == TT)continue;
            que[++r] = v;
            fa[v] = que[l];
        }
    for(l--;l;l--){
        int x = que[l], Max = 0;
        size[x] = 1;
        for(int i = head[x];i != -1;i = edge[i].next){
            int v = edge[i].to;
            if(v == fa[x] || vis[v] == TT)continue;
            Max = max(Max,size[v]);
            size[x] += size[v];
        }
        Max = max(Max,r - size[x]);
        if(Max < Min){
            Min = Max; root = x;
        }
    }
    return root;
}

int ans[MAXN];
pair<int,int>pp[MAXN];
pair<int,int>np[MAXN];
int dis[MAXN];
int type[MAXN];
inline void go(int u,int pre,int w,int tt){
    int l,r;
    que[l = r = 1] = u;
    fa[u] = pre; dis[u] = w;
    for(;l <= r;l++)
        for(int i = head[que[l]];i != -1;i = edge[i].next){
            int v = edge[i].to;
            if(v == fa[que[l]] || vis[v] == TT)continue;
            que[++r] = v;
            fa[v] = que[l];
            dis[v] = dis[que[l]] + edge[i].w;
        }
    int cnt = 0;
    for(int i = 1;i <= r;i++){
        int x = que[i];
        pp[cnt++] = make_pair(np[x].first-dis[x],np[x].second);
    }
    sort(pp,pp+cnt);
    for(int i = 1;i <= r;i++){
        int x = que[i];
        if(type[x])continue;
        int id = lower_bound(pp,pp+cnt,make_pair(dis[x],x)) - pp;
        ans[x] += (cnt-id)*tt;
    }
}
void solve(int u){
    int root = getroot(u);
    vis[root] = TT;
    go(root,0,0,1);
    for(int i = head[root];i != -1;i = edge[i].next){
        int v = edge[i].to;
        if(vis[v] == TT)continue;
        go(v,root,edge[i].w,-1);
    }
    for(int i = head[root];i != -1;i = edge[i].next){
        int v = edge[i].to;
        if(vis[v] == TT)continue;
        solve(v);
    }
}
bool ff[MAXN];
int main()
{
    int n;
    memset(vis,0,sizeof(vis));
    TT = 0;
    while(scanf("%d",&n) == 1){
        init();
        int u,v,w;
        for(int i = 1;i < n;i++){
            scanf("%d%d%d",&u,&v,&w);
            addedge(u,v,w);
            addedge(v,u,w);
        }
        for(int i = 1;i <= n;i++)scanf("%d",&type[i]);
        queue<int>q;
        for(int i = 1;i <= n;i++){
            if(type[i]){
                np[i] = make_pair(0,i);
                ff[i] = true;
                q.push(i);
            }
            else{
                np[i] = make_pair(INF,0);
                ff[i] = false;
            }
        }
        while(!q.empty()){
            u = q.front();
            q.pop();
            ff[u] = false;
            for(int i = head[u];i != -1;i = edge[i].next){
                v = edge[i].to;
                pair<int,int>tmp = make_pair(np[u].first+edge[i].w,
                   np[u].second);
                if(tmp < np[v]){
                    np[v] = tmp;
                    if(!ff[v]){
                        ff[v] = true;
                        q.push(v);
                    }
                }
            }
        }
        TT++;
        for(int i = 1;i <= n;i++)ans[i] = 0;
        solve(1);
        int ret = 0;
        for(int i = 1;i <= n;i++)ret = max(ret,ans[i]);
        printf("%d\n",ret);
    }
    return 0;
}
```

4.22.2 * 点分治 -HDU4918

HDU 4918
题意：给出一颗 n 个点的树，每个点有一个权值，有两种操作，一种是将某个点的权值修改为 v，另一种是查询距离点 u 不超过 d 的点的权值和。

```
const int MAXN = 100010;
const int MAXD = 40;
int cc[MAXN*MAXD];
int *cc_tail; //记得初始化 cc_tail = cc
//0-based BinaryIndexTree
struct BIT{
    int *c;
    int n;
    void init(int _n){
        n = _n;
        c = cc_tail;
        cc_tail = cc_tail + n;
        memset(c,0,sizeof(int)*n);
    }
    void add(int i,int val){
        while(i < n){
            c[i] += val;
            i += ~i & i+1;
        }
    }
    int sum(int i){
        i = min(i,n-1);
        int s = 0;
        while(i >= 0){
            s += c[i];
            i -= ~i & i+1;
        }
        return s;
    }
}bits[MAXN<<1];
struct Edge{
    int to,next;
}edge[MAXN*2];
int head[MAXN],tot;
void init(){
    tot = 0;
    memset(head,-1,sizeof(head));
}
inline void addedge(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
int size[MAXN],vis[MAXN],fa[MAXN],que[MAXN];
int TT;
inline int getroot(int u,int &tot){
    int Min = MAXN, root = 0;
    int l,r;
    que[l = r = 1] = u;
    fa[u] = 0;
    for(;l <= r;l++)
        for(int i = head[que[l]];i != -1;i = edge[i].next){
            int v = edge[i].to;
            if(v == fa[que[l]] || vis[v] == TT)continue;
            que[++r] = v;
            fa[v] = que[l];
        }
    tot = r;
    for(l--;l;l--){
        int x = que[l], Max = 0;
        size[x] = 1;
        for(int i = head[x];i != -1;i = edge[i].next){
            int v = edge[i].to;
            if(v == fa[x] || vis[v] == TT)continue;
            Max = max(Max,size[v]);
            size[x] += size[v];
        }
        Max = max(Max,r - size[x]);
        if(Max < Min){
            Min = Max, root = x;
        }
    }
    return root;
}
struct Node{
    int root,subtree,dis;
    Node(int _root = 0, int _subtree = 0,int _dis = 0){
        root = _root;
        subtree = _subtree;
        dis = _dis;
    }
};
vector<Node>vec[MAXN];
int id[MAXN];
int dist[MAXN];
int val[MAXN];
int cnt;
inline void go(int u,int root,int subtree){
    int l,r;
    que[l = r = 1] = u;
    fa[u] = 0; dist[u] = 1;
    for(; l <= r;l++){
        u = que[l];
        vec[u].push_back(Node(id[root],subtree,dist[u]));
        for(int i = head[u];i != -1;i = edge[i].next){
            int v = edge[i].to;
            if(v == fa[u] || vis[v] == TT)continue;
            que[++r] = v;
            fa[v] = u;
            dist[v] = dist[u] + 1;
        }
    }
    bits[subtree].init(r+1);
    for(int i = 1;i <= r;i++){
        u = que[i];
        bits[id[root]].add(dist[u],val[u]);
        bits[subtree].add(dist[u],val[u]);
    }
}
```

```
void solve(int u){
    int tot;
    int root = getroot(u,tot);
    vis[root] = TT;
    id[root] = cnt++;
    vec[root].push_back(Node(id[root],-1,0));
    bits[id[root]].init(tot);
    bits[id[root]].add(0,val[root]);
    for(int i = head[root];i != -1;i = edge[i].next){
        int v = edge[i].to;
        if(vis[v] == TT)continue;
        go(v,root,cnt);
        cnt++;
    }
    for(int i = head[root];i != -1;i = edge[i].next){
        int v = edge[i].to;
        if(vis[v] == TT)continue;
        solve(v);
    }
}
int main(){
    int n,m;
    memset(vis,0,sizeof(vis));
    TT = 0;
    while(scanf("%d%d",&n,&m) == 2){
        init();
        TT++;
        cc_tail = cc;
        cnt = 0;
        for(int i = 1;i <= n;i++)vec[i].clear();
        for(int i = 1;i <= n;i++)scanf("%d",&val[i]);
        int u,v;
        for(int i = 1;i < n;i++){
            scanf("%d%d",&u,&v);
            addedge(u,v);
            addedge(v,u);
        }
        solve(1);
        char op[10];
        int d;
        while(m--){
            scanf("%s%d%d",op,&u,&d);
            if(op[0] == '!'){
                int dv = d - val[u];
                int sz = vec[u].size();
                for(int i = 0;i < sz;i++){
                    Node tmp = vec[u][i];
                    bits[tmp.root].add(tmp.dis,dv);
                    if(tmp.subtree != -1)
                        bits[tmp.subtree].add(tmp.dis,dv);
                }
```

```
                    val[u] += dv;
                }
                else {
                    int ans = 0;
                    int sz = vec[u].size();
                    for(int i = 0;i < sz;i++){
                        Node tmp = vec[u][i];
                        ans += bits[tmp.root].sum(d-tmp.dis);
                        if(tmp.subtree != -1)
                            ans -= bits[tmp.subtree].sum(d-tmp.dis);
                    }
                    printf("%d\n",ans);
                }
            }
        }
    return 0;
}
```

4.22.3 链分治 -HDU5039

HDU 5039 一颗树，每条边的属性为 0 或 1，求有多少条路径经过奇数条属性为 1 的边。
一种是查询操作，一种是修改边的属性。
虽然这题有更加简单的方法，但是用来练习链分治还是不错的。

```
const int MAXN = 30010;
const int INF = 0x3f3f3f3f;
struct Edge{
    int to,next;
    int f;
}edge[MAXN*2];
int head[MAXN],tot;
void init(){
    tot = 0;
    memset(head,-1,sizeof(head));
}
void addedge(int u,int v,int f){
    edge[tot].to = v;
    edge[tot].next = head[u];
    edge[tot].f = f;
    head[u] = tot++;
}
long long ans;
int num0[MAXN],num1[MAXN];
long long tnum[MAXN];
struct Node{
    int l0,l1;
    int r0,r1;
    int cc;
    long long sum;
    Node gao(int u){
        l0 = r0 = num0[u];
        l1 = r1 = num1[u];
        sum = tnum[u];
        cc = 0;
        return *this;
    }
};
int pos[MAXN];
int val[MAXN];
int fa[MAXN];
int cnt[MAXN];
int col[MAXN];
int link[MAXN];
int CHANGEU;
struct chain{
    vector<int>uu;
    vector<Node>nde;
    int n;
    void init(){
        n = uu.size();
        nde.resize(n << 2);
        for(int i = 0;i < n;i++)pos[uu[i]] = i;
        build(0,n−1,1);
    }
    void up(int l,int r,int p){
        int mid = (l+r)>>1;
        nde[p].cc = nde[p<<1].cc ^ nde[(p<<1)|1].cc ^ val[uu[mid]];
        nde[p].l0 = nde[p<<1].l0;
        nde[p].l1 = nde[p<<1].l1;
        if(nde[p<<1].cc^val[uu[mid]]){
            nde[p].l0 += nde[(p<<1)|1].l1;
            nde[p].l1 += nde[(p<<1)|1].l0;
        }
        else {
            nde[p].l0 += nde[(p<<1)|1].l0;
            nde[p].l1 += nde[(p<<1)|1].l1;
        }
        nde[p].r0 = nde[(p<<1)|1].r0;
        nde[p].r1 = nde[(p<<1)|1].r1;
        if(nde[(p<<1)|1].cc^val[uu[mid]]){
            nde[p].r0 += nde[p<<1].r1;
            nde[p].r1 += nde[p<<1].r0;
        }
        else {
            nde[p].r0 += nde[p<<1].r0;
            nde[p].r1 += nde[p<<1].r1;
        }
        if(val[uu[mid]] == 0){
            nde[p].sum = nde[p<<1].sum + nde[(p<<1)|1].sum +
                (long long)nde[p<<1].r0*nde[(p<<1)|1].l1 +
                (long long)nde[p<<1].r1*nde[(p<<1)|1].l0;
        }
        else {
            nde[p].sum = nde[p<<1].sum + nde[(p<<1)|1].sum +
                (long long)nde[p<<1].r0*nde[(p<<1)|1].l0 +
                (long long)nde[p<<1].r1*nde[(p<<1)|1].l1;
        }
    }
    void build(int l,int r,int p){
        if(l == r){
            nde[p].gao(uu[l]);
            return;
        }
        int mid = (l+r)/2;
        build(l,mid,p<<1);
        build(mid+1,r,(p<<1)|1);
        up(l,r,p);
    }
    void update(int k,int l,int r,int p){
        if(l == r){
            nde[p].gao(uu[k]);
            return;
        }
        int mid = (l+r)/2;
        if(k <= mid)update(k,l,mid,p<<1);
        else update(k,mid+1,r,(p<<1)|1);
        up(l,r,p);
    }
    int change(int y){
        int x = uu.back();
        int p = fa[x];
        if(p){
            if(x == CHANGEU)val[x] ^= 1;
            if(val[x]){
                tnum[p] -= (long long)nde[1].r0*(num0[p]-nde[1].r1)
                    ;
                tnum[p] -= (long long)nde[1].r1*(num1[p]-nde[1].r0)
                    ;
                num0[p] -= nde[1].r1;
                num1[p] -= nde[1].r0;
            }
            else {
                tnum[p] -= (long long)nde[1].r1*(num0[p]-nde[1].r0)
                    ;
                tnum[p] -= (long long)nde[1].r0*(num1[p]-nde[1].r1)
                    ;
                num0[p] -= nde[1].r0;
                num1[p] -= nde[1].r1;
            }
            if(x == CHANGEU)val[x] ^= 1;
        }
        ans -= nde[1].sum;
        update(pos[y],0,n-1,1);
        if(p){
            if(val[x]){
                tnum[p] += (long long)nde[1].r0*num0[p];
                tnum[p] += (long long)nde[1].r1*num1[p];
                num0[p] += nde[1].r1;
                num1[p] += nde[1].r0;
            }
            else {
                tnum[p] += (long long)nde[1].r0*num1[p];
                tnum[p] += (long long)nde[1].r1*num0[p];
                num0[p] += nde[1].r0;
                num1[p] += nde[1].r1;
            }
        }
        ans += nde[1].sum;
        return p;
    }
}ch[MAXN];
void dfs1(int u,int pre){
    chain &c = ch[u];
    c.uu.clear();
    int v, x = 0;
    cnt[u] = 1;
    for(int i = head[u];i != −1;i = edge[i].next){
        v = edge[i].to;
        if(v == pre)continue;
        dfs1(v,u);
        link[i/2] = v;
        val[v] = edge[i].f;
        cnt[u] += cnt[v];
        fa[v] = u;
        if(cnt[v] > cnt[x]) x = v;
    }
    if(!x)col[u] = u;
    else col[u] = col[x];
    ch[col[u]].uu.push_back(u);
    num0[u] = 1;
    num1[u] = 0;
    tnum[u] = 0;

}
void dfs2(int x){
    x = col[x];
    chain &c = ch[x];
    int n = c.uu.size();
    int u,v;
    for(int i = 1;i < n;i++){
        u = c.uu[i];
        for(int j = head[u];j != −1;j = edge[j].next){
            v = edge[j].to;
            if(v == c.uu[i−1] || fa[u] == v)continue;
            dfs2(v);
            if(val[v]){
                tnum[u] += (long long)num0[u]*ch[col[v]].nde[1].r0
                   + (long long)num1[u]*ch[col[v]].nde[1].r1;
                num0[u] += ch[col[v]].nde[1].r1;
                num1[u] += ch[col[v]].nde[1].r0;
            }
            else {
                tnum[u] += (long long)num1[u]*ch[col[v]].nde[1].r0
                   + (long long)num0[u]*ch[col[v]].nde[1].r1;
                num0[u] += ch[col[v]].nde[1].r0;
                num1[u] += ch[col[v]].nde[1].r1;
            }
        }
    }
    c.init();
    ans += c.nde[1].sum;
}
char str[100];
char str1[100],str2[100];
int main()
{
    int T;
    int iCase = 0;
    scanf("%d",&T);
    int n;
    while(T--){
        ans = 0;
        iCase++;
        scanf("%d",&n);
        map<string,int>mp;
        init();
        for(int i = 1;i <= n;i++){
            scanf("%s",str);
            mp[str] = i;
        }
        int u,v,f;
        for(int i = 1;i < n;i++){
            scanf("%s%s%d",str1,str2,&f);
            addedge(mp[str1],mp[str2],f);
            addedge(mp[str2],mp[str1],f);
        }
        int Q;
        char op[10];
        scanf("%d",&Q);
        printf("Case␣#%d:\n",iCase);
        val[1] = 0;
        fa[1] = 0;
        dfs1(1,1);
        dfs2(1);
        while(Q--){
            scanf("%s",op);
            if(op[0] == 'Q'){
                printf("%I64d\n",ans*2);
            }
            else {
                int id ;
                scanf("%d",&id);
                id--;
                u = link[id];
                val[u] ^= 1;
                CHANGEU = u;
                while(u)
                    u = ch[col[u]].change(u);
            }
        }
    }
    return 0;
}
```

5 搜索

5.1 Dancing Links

5.1.1 精确覆盖

```
/*
 * POJ3074
 */
const int N = 9; //3*3 数独
const int MaxN = N*N*N + 10;
const int MaxM = N*N*4 + 10;
const int maxnode = MaxN*4 + MaxM + 10;
char g[MaxN];
struct DLX{
    int n,m,size;
    int U[maxnode],D[maxnode],R[maxnode],L[maxnode],Row[maxnode],
       Col[maxnode];
    int H[MaxN],S[MaxM];
    int ansd,ans[MaxN];
    void init(int _n,int _m){
        n = _n;
        m = _m;
        for(int i = 0;i <= m;i++){
            S[i] = 0;
            U[i] = D[i] = i;
            L[i] = i-1;
            R[i] = i+1;
        }
        R[m] = 0; L[0] = m;
        size = m;
        for(int i = 1;i <= n;i++)H[i] = -1;
    }
    void Link(int r,int c){
        ++S[Col[++size]=c];
        Row[size] = r;
        D[size] = D[c];
        U[D[c]] = size;
        U[size] = c;
        D[c] = size;
        if(H[r] < 0)H[r] = L[size] = R[size] = size;
        else{
            R[size] = R[H[r]];
            L[R[H[r]]] = size;
            L[size] = H[r];
            R[H[r]] = size;
        }
    }
    void remove(int c){
        L[R[c]] = L[c]; R[L[c]] = R[c];
        for(int i = D[c];i != c;i = D[i])
            for(int j = R[i];j != i;j = R[j]){
                U[D[j]] = U[j];
                D[U[j]] = D[j];
                --S[Col[j]];
            }
    }
    void resume(int c){
        for(int i = U[c];i != c;i = U[i])
            for(int j = L[i];j != i;j = L[j])
                ++S[Col[U[D[j]]=D[U[j]]=j]];
        L[R[c]] = R[L[c]] = c;
    }
    bool Dance(int d){
        if(R[0] == 0){
            for(int i = 0;i < d;i++)g[(ans[i]-1)/9] = (ans[i]-1)%9
               + '1';
            for(int i = 0;i < N*N;i++)printf("%c",g[i]);
            printf("\n");
            return true;
        }
        int c = R[0];
        for(int i = R[0];i != 0;i = R[i])
            if(S[i] < S[c])
                c = i;
        remove(c);
        for(int i = D[c];i != c;i = D[i]){
            ans[d] = Row[i];
            for(int j = R[i];j != i;j = R[j])remove(Col[j]);
            if(Dance(d+1))return true;
            for(int j = L[i];j != i;j = L[j])resume(Col[j]);
        }
        resume(c);
        return false;
    }
};
void place(int &r,int &c1,int &c2,int &c3,int &c4,int i,int j,int k
   ){
    r = (i*N+j)*N + k; c1 = i*N+j+1; c2 = N*N+i*N+k;
    c3 = N*N*2+j*N+k; c4 = N*N*3+((i/3)*3+(j/3))*N+k;
}
DLX dlx;
int main(){
    while(scanf("%s",g) == 1){
        if(strcmp(g,"end") == 0)break;
        dlx.init(N*N*N,N*N*4);
        int r,c1,c2,c3,c4;
        for(int i = 0;i < N;i++)
            for(int j = 0;j < N;j++)
                for(int k = 1;k <= N;k++)
                    if(g[i*N+j] == '.' || g[i*N+j] == '0'+k){
                        place(r,c1,c2,c3,c4,i,j,k);
                        dlx.Link(r,c1);
                        dlx.Link(r,c2);
                        dlx.Link(r,c3);
                        dlx.Link(r,c4);
                    }
        dlx.Dance(0);
    }
    return 0;
}
```

5.1.2 可重复覆盖

```
/*
 * FZU1686
 */
const int MaxM = 15*15+10;
const int MaxN = 15*15+10;
const int maxnode = MaxN * MaxM;
const int INF = 0x3f3f3f3f;
struct DLX{
    int n,m,size;
    int U[maxnode],D[maxnode],R[maxnode],L[maxnode],Row[maxnode],
       Col[maxnode];
    int H[MaxN],S[MaxM];
    int ansd;
    void init(int _n,int _m){
        n = _n;
        m = _m;
        for(int i = 0;i <= m;i++){
            S[i] = 0;
            U[i] = D[i] = i;
            L[i] = i-1;
            R[i] = i+1;
        }
        R[m] = 0; L[0] = m;
        size = m;
        for(int i = 1;i <= n;i++)H[i] = -1;
    }
    void Link(int r,int c){
        ++S[Col[++size]=c];
        Row[size] = r;
        D[size] = D[c];
        U[D[c]] = size;
        U[size] = c;
        D[c] = size;
        if(H[r] < 0)H[r] = L[size] = R[size] = size;
        else{
            R[size] = R[H[r]];
            L[R[H[r]]] = size;
            L[size] = H[r];
            R[H[r]] = size;
        }
    }
    void remove(int c){
        for(int i = D[c];i != c;i = D[i])
            L[R[i]] = L[i], R[L[i]] = R[i];
    }
    void resume(int c){
        for(int i = U[c];i != c;i = U[i])
            L[R[i]] = R[L[i]] = i;
    }
    bool v[MaxM];
    int f(){
        int ret = 0;
        for(int c = R[0]; c != 0;c = R[c])v[c] = true;
        for(int c = R[0]; c != 0;c = R[c])
            if(v[c])
            {
                ret++;
                v[c] = false;
                for(int i = D[c];i != c;i = D[i])
                    for(int j = R[i];j != i;j = R[j])
                        v[Col[j]] = false;
            }
        return ret;
    }
    void Dance(int d){
        if(d + f() >= ansd)return;
        if(R[0] == 0){
            if(d < ansd)ansd = d;
            return;
        }
        int c = R[0];
        for(int i = R[0];i != 0;i = R[i])
            if(S[i] < S[c])
                c = i;
        for(int i = D[c];i != c;i = D[i]){
            remove(i);
            for(int j = R[i];j != i;j = R[j])remove(j);
            Dance(d+1);
            for(int j = L[i];j != i;j = L[j])resume(j);
            resume(i);
        }
    }
};
DLX g;
int a[20][20];
int id[20][20];
int main(){
    int n,m;
    while(scanf("%d%d",&n,&m) == 2){
        int sz = 0;
        memset(id,0,sizeof(id));
        for(int i = 0;i < n;i++)
```

```
            for(int j = 0;j < m;j++){
                scanf("%d",&a[i][j]);
                if(a[i][j] == 1)id[i][j] = (++sz);
            }
        g.init(n*m,sz);
        sz = 1;
        int n1,m1;
        scanf("%d%d",&n1,&m1);
        for(int i = 0;i < n;i++)
            for(int j = 0;j < m;j++){
                for(int x = 0;x < n1 && i + x < n;x++)
                    for(int y = 0;y < m1 && j + y < m;y++)
                        if(id[i+x][j+y])
                            g.Link(sz,id[i+x][j+y]);
                sz++;
            }
        g.ansd = INF;
        g.Dance(0);
        printf("%d\n",g.ansd);
    }
    return 0;
}
```

5.2 八数码

5.2.1 HDU1043 反向搜索

```
/*
HDU 1043 Eight
八数码，输出路径
思路：反向搜索，从目标状态找回状态对应的路径
用康托展开判重
*/
const int MAXN=1000000;//最多是 9!/2
int fac[]={1,1,2,6,24,120,720,5040,40320,362880};//康拖展开判重
//         0!1!2!3! 4! 5!  6!  7!   8!    9!
bool vis[MAXN];//标记
string path[MAXN];//记录路径
//康拖展开求该序列的 hash 值
int cantor(int s[]){
    int sum=0;
    for(int i=0;i<9;i++){
        int num=0;
        for(int j=i+1;j<9;j++)
          if(s[j]<s[i])num++;
        sum+=(num*fac[9-i-1]);
    }
    return sum+1;
}
struct Node{
    int s[9];
    int loc;//'0' 的位置
    int status;//康拖展开的 hash 值
    string path;//路径
};
int move[4][2]={{-1,0},{1,0},{0,-1},{0,1}};//u,d,l,r
char indexs[5]="durl";//和上面的要相反，因为是反向搜索
int aim=46234;//123456780 对应的康拖展开的 hash 值
void bfs(){
    memset(vis,false,sizeof(vis));
    Node cur,next;
    for(int i=0;i<8;i++)cur.s[i]=i+1;
    cur.s[8]=0;
    cur.loc=8;
    cur.status=aim;
    cur.path="";
    queue<Node>q;
    q.push(cur);
    path[aim]="";
    while(!q.empty()){
        cur=q.front();
        q.pop();
        int x=cur.loc/3;
        int y=cur.loc%3;
        for(int i=0;i<4;i++){
            int tx=x+move[i][0];
            int ty=y+move[i][1];
            if(tx<0||tx>2||ty<0||ty>2)continue;
            next=cur;
            next.loc=tx*3+ty;
            next.s[cur.loc]=next.s[next.loc];
            next.s[next.loc]=0;
            next.status=cantor(next.s);
            if(!vis[next.status]){
                vis[next.status]=true;
                next.path=indexs[i]+next.path;
                q.push(next);
                path[next.status]=next.path;
            }
        }
    }

}
int main(){
    char ch;
    Node cur;
    bfs();
    while(cin>>ch){
        if(ch=='x') {cur.s[0]=0;cur.loc=0;}
        else cur.s[0]=ch-'0';
        for(int i=1;i<9;i++){
            cin>>ch;
            if(ch=='x'){
                cur.s[i]=0;
                cur.loc=i;
            }
            else cur.s[i]=ch-'0';
        }
        cur.status=cantor(cur.s);
        if(vis[cur.status]){
            cout<<path[cur.status]<<endl;
        }
        else cout<<"unsolvable"<<endl;
    }
    return 0;
}
```

6 动态规划

6.1 最长上升子序列 O(nlogn)

```
const int MAXN=500010;
int a[MAXN],b[MAXN];

//用二分查找的方法找到一个位置，使得 num>b[i-1] 并且 num<b[i]，并用 num 代替 b[i]
int Search(int num,int low,int high){
    int mid;
    while(low<=high){
        mid=(low+high)/2;
        if(num>=b[mid])  low=mid+1;
        else   high=mid-1;
    }
    return low;
}
int DP(int n){
    int i,len,pos;
    b[1]=a[1];
    len=1;
    for(i=2;i<=n;i++){
        if(a[i]>=b[len])//如果 a[i] 比 b[] 数组中最大还大直接插入到后面即可
        {
            len=len+1;
            b[len]=a[i];
        }
        else//用二分的方法在 b[] 数组中找出第一个比 a[i] 大的位置并且让 a[i] 替代这个位置
        {
            pos=Search(a[i],1,len);
            b[pos]=a[i];
        }
    }
    return len;
}
```

6.2 背包

```
int nValue,nKind;

//0-1 背包，代价为 cost，获得的价值为 weight
void ZeroOnePack(int cost,int weight){
    for(int i=nValue;i>=cost;i--)
      dp[i]=max(dp[i],dp[i-cost]+weight);
}

//完全背包，代价为 cost，获得的价值为 weight
void CompletePack(int cost,int weight){
    for(int i=cost;i<=nValue;i++)
```

```
        dp[i]=max(dp[i],dp[i-cost]+weight);
}

//多重背包
void MultiplePack(int cost,int weight,int amount){
    if(cost*amount>=nValue) CompletePack(cost,weight);
    else{
        int k=1;
        while(k<amount){
            ZeroOnePack(k*cost,k*weight);
            amount-=k;
            k<<=1;
        }
        ZeroOnePack(amount*cost,amount*weight);//这个不要忘记了，经常
           掉了
    }
}
```

分组背包：

```
for  k = 1 to K
for v = V to 0
for item i in group k
F[v] = maxF[v],F[v-$C_i$]+$W_i$
```

6.3 插头 DP

6.3.1 HDU 4285

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

```
/*
HDU 4285
要形成刚好 K 条回路的方法数
要避免环套环的情况。
所以形成回路时，要保证两边的插头数是偶数
G++ 11265ms 11820K
C++ 10656ms 11764K
*/
const int MAXD=15;
const int STATE=1000010;
const int HASH=300007;//这个大一点可以防止 TLE，但是容易 MLE
const int MOD=1000000007;
int N,M,K;
int maze[MAXD][MAXD];
int code[MAXD];
int ch[MAXD];
int num;//圈的个数
struct HASHMAP
{
    int head[HASH],next[STATE],size;
    long long state[STATE];
    int f[STATE];
    void init()
    {
        size=0;
        memset(head,-1,sizeof(head));
    }
    void push(long long st,int ans)
    {
        int i;
        int h=st%HASH;
        for(i=head[h];i!=-1;i=next[i])
          if(state[i]==st)
          {
              f[i]+=ans;
              f[i]%=MOD;
              return;
          }
        state[size]=st;
        f[size]=ans;
        next[size]=head[h];
        head[h]=size++;
    }
}hm[2];
void decode(int *code,int m,long long  st)
{
    num=st&63;
    st>>=6;
    for(int i=m;i>=0;i--)
    {
        code[i]=st&7;
        st>>=3;
    }
}
long long encode(int *code,int m)//最小表示法
{
    int cnt=1;
    memset(ch,-1,sizeof(ch));
    ch[0]=0;
    long long st=0;
    for(int i=0;i<=m;i++)
    {
        if(ch[code[i]]==-1)ch[code[i]]=cnt++;
        code[i]=ch[code[i]];
        st<<=3;
        st|=code[i];
    }
    st<<=6;
    st|=num;
    return st;
}
void shift(int *code,int m)
```

```
{
    for(int i=m;i>0;i--)code[i]=code[i-1];
    code[0]=0;
}
void dpblank(int i,int j,int cur)
{
    int k,left,up;
    for(k=0;k<hm[cur].size;k++)
    {
        decode(code,M,hm[cur].state[k]);
        left=code[j-1];
        up=code[j];
        if(left&&up)
        {
            if(left==up)
            {
                if(num>=K)continue;
                int t=0;
                //要避免环套环的情况，需要两边插头数为偶数
                for(int p=0;p<j-1;p++)
                  if(code[p])t++;
                if(t&1)continue;
                if(num<K)
                {
                    num++;
                    code[j-1]=code[j]=0;
                    hm[cur^1].push(encode(code,j==M?M-1:M),hm[cur].
                       f[k]);
                }
            }
            else
            {
                code[j-1]=code[j]=0;
                for(int t=0;t<=M;t++)
                  if(code[t]==up)
                    code[t]=left;
                hm[cur^1].push(encode(code,j==M?M-1:M),hm[cur].f[k
                  ]);
            }
        }
        else if(left||up)
        {
            int t;
            if(left)t=left;
            else t=up;
            if(maze[i][j+1])
            {
                code[j-1]=0;
                code[j]=t;
                hm[cur^1].push(encode(code,M),hm[cur].f[k]);
            }
            if(maze[i+1][j])
            {
                code[j]=0;
                code[j-1]=t;
                hm[cur^1].push(encode(code,j==M?M-1:M),hm[cur].f[k
                   ]);
            }
        }
        else
        {
            if(maze[i][j+1]&&maze[i+1][j])
            {
                code[j-1]=code[j]=13;
                hm[cur^1].push(encode(code,j==M?M-1:M),hm[cur].f[k
                   ]);
            }
        }
    }
}
void dpblock(int i,int j,int cur)
{
    int k;
    for(k=0;k<hm[cur].size;k++)
    {
        decode(code,M,hm[cur].state[k]);
        code[j-1]=code[j]=0;
        hm[cur^1].push(encode(code,j==M?M-1:M),hm[cur].f[k]);
    }
}
char str[20];
void init()
{
    scanf("%d%d%d",&N,&M,&K);
    memset(maze,0,sizeof(maze));
    for(int i=1;i<=N;i++)
    {
        scanf("%s",&str);
        for(int j=1;j<=M;j++)
          if(str[j-1]=='.')
            maze[i][j]=1;
    }
}
void solve()
{
    int i,j,cur=0;
    hm[cur].init();
    hm[cur].push(0,1);
    for(i=1;i<=N;i++)
      for(j=1;j<=M;j++)
      {
          hm[cur^1].init();
```

```
            if(maze[i][j])dpblank(i,j,cur);
            else dpblock(i,j,cur);
            cur^=1;
        }
    int ans=0;
    for(i=0;i<hm[cur].size;i++)
      if(hm[cur].state[i]==K)
      {
          ans+=hm[cur].f[i];
          ans%=MOD;
      }
    printf("%d\n",ans);

}
int main()
{
    int T;
    scanf("%d",&T);
    while(T--)
    {
        init();
        solve();
    }
    return 0;
}

/*
Sample Input
4 4 1
**..
....
....
....
4 1
....
....
....
....

Sample Output
6

*/
```

7 计算几何

7.1 二维几何

```
// 计算几何模板
const double eps = 1e-8;
const double inf = 1e20;
const double pi = acos(-1.0);
const int maxp = 1010;
//Compares a double to zero
int sgn(double x){
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
//square of a double
inline double sqr(double x){return x*x;}
/*
 * Point
 * Point()               - Empty constructor
 * Point(double _x,double _y)  - constructor
 * input()             - double input
 * output()            - %.2f output
 * operator ==         - compares x and y
 * operator <          - compares first by x, then by y
 * operator -          - return new Point after subtracting curresponging x and y
 * operator ^          - cross product of 2d points
 * operator *          - dot product
 * len()               - gives length from origin
 * len2()              - gives square of length from origin
 * distance(Point p)   - gives distance from p
 * operator + Point b  - returns new Point after adding curresponging x and y
 * operator * double k - returns new Point after multiplieing x and y by k
 * operator / double k - returns new Point after divideing x and y by k
 * rad(Point a,Point b)- returns the angle of Point a and Point b from this Point
 * trunc(double r)     - return Point that if truncated the distance from center to r
 * rotleft()           - returns 90 degree ccw rotated point
 * rotright()          - returns 90 degree cw rotated point
 * rotate(Point p,double angle) - returns Point after rotateing the Point centering at p by angle radian ccw
 */
struct Point{
    double x,y;
    Point(){}
    Point(double _x,double _y){
        x = _x;
        y = _y;
    }
    void input(){
        scanf("%lf%lf",&x,&y);
    }
    void output(){
        printf("%.2f␣%.2f\n",x,y);
    }
    bool operator == (Point b)const{
        return sgn(x-b.x) == 0 && sgn(y-b.y) == 0;
    }
    bool operator < (Point b)const{
        return sgn(x-b.x)== 0?sgn(y-b.y)<0:x<b.x;
    }
    Point operator -(const Point &b)const{
        return Point(x-b.x,y-b.y);
    }
    //叉积
    double operator ^(const Point &b)const{
        return x*b.y - y*b.x;
    }
    //点积
    double operator *(const Point &b)const{
        return x*b.x + y*b.y;
    }
    //返回长度
    double len(){
        return hypot(x,y);//库函数
    }
    //返回长度的平方
    double len2(){
        return x*x + y*y;
    }
    //返回两点的距离
    double distance(Point p){
        return hypot(x-p.x,y-p.y);
    }
    Point operator +(const Point &b)const{
        return Point(x+b.x,y+b.y);
    }
    Point operator *(const double &k)const{
        return Point(x*k,y*k);
    }
    Point operator /(const double &k)const{
        return Point(x/k,y/k);
    }
    //计算 pa 和 pb 的夹角
    //就是求这个点看 a,b 所成的夹角
    //测试 LightOJ1203
    double rad(Point a,Point b){
        Point p = *this;
        return fabs(atan2( fabs((a-p)^(b-p)),(a-p)*(b-p) ));
    }
    //化为长度为 r 的向量
    Point trunc(double r){
        double l = len();
        if(!sgn(l))return *this;
        r /= l;
        return Point(x*r,y*r);
    }
    //逆时针旋转 90 度
    Point rotleft(){
        return Point(-y,x);
    }
    //顺时针旋转 90 度
    Point rotright(){
        return Point(y,-x);
    }
    //绕着 p 点逆时针旋转 angle
    Point rotate(Point p,double angle){
        Point v = (*this) - p;
        double c = cos(angle), s = sin(angle);
        return Point(p.x + v.x*c - v.y*s,p.y + v.x*s + v.y*c);
    }
};
/*
 * Stores two points
 * Line()                           - Empty constructor
 * Line(Point _s,Point _e)          - Line through _s and _e
 * operator ==                      - checks if two points are same
 * Line(Point p,double angle)       - one end p , another end at
   angle degree
 * Line(double a,double b,double c) - Line of equation ax + by + c
   = 0
 * input()                          - inputs s and e
 * adjust()                         - orders in such a way that s < e
 * length()                         - distance of se
 * angle()                          - return 0 <= angle < pi
 * relation(Point p)                - 3 if point is on line
 *                                    1 if point on the left of line
 *                                    2 if point on the right of line
 * pointonseg(double p)             - return true if point on segment
 * parallel(Line v)                 - return true if they are
   parallel
 * segcrossseg(Line v)              - returns 0 if does not intersect
 *                                    returns 1 if non-standard
   intersection
 *                                    returns 2 if intersects
 * linecrossseg(Line v)             - line and seg
 * linecrossline(Line v)            - 0 if parallel
 *                                    1 if coincides
 *                                 2 if intersects
 * crosspoint(Line v)            - returns intersection point
 * dispointtoline(Point p)       - distance from point p to the
   line
 * dispointtoseg(Point p)        - distance from p to the segment
 * dissegtoseg(Line v)           - distance of two segment
 * lineprog(Point p)             - returns projected point p on se
    line
 * symmetrypoint(Point p)        - returns reflection point of p
   over se
 *
 */
struct Line{
    Point s,e;
    Line(){}
    Line(Point _s,Point _e){
        s = _s;
        e = _e;
    }
    bool operator ==(Line v){
        return (s == v.s)&&(e == v.e);
    }
    //根据一个点和倾斜角 angle 确定直线,0<=angle<pi
    Line(Point p,double angle){
        s = p;
        if(sgn(angle-pi/2) == 0){
            e = (s + Point(0,1));
        }
        else{
            e = (s + Point(1,tan(angle)));
        }
    }
    //ax+by+c=0
    Line(double a,double b,double c){
        if(sgn(a) == 0){
            s = Point(0,-c/b);
            e = Point(1,-c/b);
        }
        else if(sgn(b) == 0){
            s = Point(-c/a,0);
            e = Point(-c/a,1);
        }
        else{
            s = Point(0,-c/b);
            e = Point(1,(-c-a)/b);
        }
    }
    void input(){
        s.input();
        e.input();
    }
    void adjust(){
        if(e < s)swap(s,e);
    }
    //求线段长度
    double length(){
        return s.distance(e);
    }
    //返回直线倾斜角 0<=angle<pi
    double angle(){
        double k = atan2(e.y-s.y,e.x-s.x);
        if(sgn(k) < 0)k += pi;
        if(sgn(k-pi) == 0)k -= pi;
        return k;
    }
    //点和直线关系
    //1 在左侧
    //2 在右侧
    //3 在直线上
    int relation(Point p){
        int c = sgn((p-s)^(e-s));
        if(c < 0)return 1;
        else if(c > 0)return 2;
        else return 3;
    }
    // 点在线段上的判断
    bool pointonseg(Point p){
        return sgn((p-s)^(e-s)) == 0 && sgn((p-s)*(p-e)) <= 0;
    }
    //两向量平行 (对应直线平行或重合)
    bool parallel(Line v){
        return sgn((e-s)^(v.e-v.s)) == 0;
    }
    //两线段相交判断
    //2 规范相交
    //1 非规范相交
    //0 不相交
    int segcrossseg(Line v){
        int d1 = sgn((e-s)^(v.s-s));
        int d2 = sgn((e-s)^(v.e-s));
        int d3 = sgn((v.e-v.s)^(s-v.s));
        int d4 = sgn((v.e-v.s)^(e-v.s));
        if( (d1^d2)==-2 && (d3^d4)==-2 )return 2;
        return (d1==0 && sgn((v.s-s)*(v.s-e))<=0) ||
            (d2==0 && sgn((v.e-s)*(v.e-e))<=0) ||
            (d3==0 && sgn((s-v.s)*(s-v.e))<=0) ||
            (d4==0 && sgn((e-v.s)*(e-v.e))<=0);
    }
    //直线和线段相交判断
    //-*this line -v seg
    //2 规范相交
    //1 非规范相交
    //0 不相交
    int linecrossseg(Line v){
        int d1 = sgn((e-s)^(v.s-s));
        int d2 = sgn((e-s)^(v.e-s));
        if((d1^d2)==-2) return 2;
        return (d1==0||d2==0);
    }
    //两直线关系
    //0 平行
    //1 重合
    //2 相交
    int linecrossline(Line v){
        if((*this).parallel(v))
            return v.relation(s)==3;
        return 2;
    }
    //求两直线的交点
    //要保证两直线不平行或重合
    Point crosspoint(Line v){
        double a1 = (v.e-v.s)^(s-v.s);
        double a2 = (v.e-v.s)^(e-v.s);
        return Point((s.x*a2-e.x*a1)/(a2-a1),(s.y*a2-e.y*a1)/(a2-a1
           ));
    }
    //点到直线的距离
    double dispointtoline(Point p){
        return fabs((p-s)^(e-s))/length();
    }
    //点到线段的距离
    double dispointtoseg(Point p){
        if(sgn((p-s)*(e-s))<0 || sgn((p-e)*(s-e))<0)
            return min(p.distance(s),p.distance(e));
        return dispointtoline(p);
    }
    //返回线段到线段的距离
    //前提是两线段不相交，相交距离就是 0 了
    double dissegtoseg(Line v){
        return min(min(dispointtoseg(v.s),dispointtoseg(v.e)),min(v
           .dispointtoseg(s),v.dispointtoseg(e)));
    }
    //返回点 p 在直线上的投影
    Point lineprog(Point p){
        return s + ( ((e-s)*((e-s)*(p-s)))/((e-s).len2()) );
    }
    //返回点 p 关于直线的对称点
    Point symmetrypoint(Point p){
        Point q = lineprog(p);
        return Point(2*q.x-p.x,2*q.y-p.y);
    }
};
//圆
```

```
struct circle{
    Point p;//圆心
    double r;//半径
    circle(){}
    circle(Point _p,double _r){
        p = _p;
        r = _r;
    }
    circle(double x,double y,double _r){
        p = Point(x,y);
        r = _r;
    }
    //三角形的外接圆
    //需要 Point 的 + / rotate() 以及 Line 的 crosspoint()
    //利用两条边的中垂线得到圆心
    //测试：UVA12304
    circle(Point a,Point b,Point c){
        Line u = Line((a+b)/2,((a+b)/2)+((b-a).rotleft()));
        Line v = Line((b+c)/2,((b+c)/2)+((c-b).rotleft()));
        p = u.crosspoint(v);
        r = p.distance(a);
    }
    //三角形的内切圆
    //参数 bool t 没有作用，只是为了和上面外接圆函数区别
    //测试：UVA12304
    circle(Point a,Point b,Point c,bool t){
        Line u,v;
        double m = atan2(b.y-a.y,b.x-a.x), n = atan2(c.y-a.y,c.x-a.
           x);
        u.s = a;
        u.e = u.s + Point(cos((n+m)/2),sin((n+m)/2));
        v.s = b;
        m = atan2(a.y-b.y,a.x-b.x) , n = atan2(c.y-b.y,c.x-b.x);
        v.e = v.s + Point(cos((n+m)/2),sin((n+m)/2));
        p = u.crosspoint(v);
        r = Line(a,b).dispointtoseg(p);
    }
    //输入
    void input(){
        p.input();
        scanf("%lf",&r);
    }
    //输出
    void output(){
        printf("%.2lf␣%.2lf␣%.2lf\n",p.x,p.y,r);
    }
    bool operator == (circle v){
        return (p==v.p) && sgn(r-v.r)==0;
    }
    bool operator < (circle v)const{
        return ((p<v.p)||((p==v.p)&&sgn(r-v.r)<0));
    }
    //面积
    double area(){
        return pi*r*r;
    }
    //周长
    double circumference(){
        return 2*pi*r;
    }
    //点和圆的关系
    //0 圆外
    //1 圆上
    //2 圆内
    int relation(Point b){
        double dst = b.distance(p);
        if(sgn(dst-r) < 0)return 2;
        else if(sgn(dst-r)==0)return 1;
        return 0;
    }
    //线段和圆的关系
    //比较的是圆心到线段的距离和半径的关系
    int relationseg(Line v){
        double dst = v.dispointtoseg(p);
        if(sgn(dst-r) < 0)return 2;
        else if(sgn(dst-r) == 0)return 1;
        return 0;
    }
    //直线和圆的关系
    //比较的是圆心到直线的距离和半径的关系
    int relationline(Line v){
        double dst = v.dispointtoline(p);
        if(sgn(dst-r) < 0)return 2;
        else if(sgn(dst-r) == 0)return 1;
        return 0;
    }
    //两圆的关系
    //5 相离
    //4 外切
    //3 相交
    //2 内切
    //1 内含
    //需要 Point 的 distance
    //测试：UVA12304
    int relationcircle(circle v){
        double d = p.distance(v.p);
        if(sgn(d-r-v.r) > 0)return 5;
        if(sgn(d-r-v.r) == 0)return 4;
        double l = fabs(r-v.r);
        if(sgn(d-r-v.r)<0 && sgn(d-l)>0)return 3;
        if(sgn(d-l)==0)return 2;
        if(sgn(d-l)<0)return 1;
    }
    //求两个圆的交点，返回 0 表示没有交点，返回 1 是一个交点，2 是两个交点
    //需要 relationcircle
    //测试：UVA12304
    int pointcrosscircle(circle v,Point &p1,Point &p2){
        int rel = relationcircle(v);
        if(rel == 1 || rel == 5)return 0;
        double d = p.distance(v.p);
        double l = (d*d+r*r-v.r*v.r)/(2*d);
        double h = sqrt(r*r-l*l);
        Point tmp = p + (v.p-p).trunc(l);
        p1 = tmp + ((v.p-p).rotleft().trunc(h));
        p2 = tmp + ((v.p-p).rotright().trunc(h));
        if(rel == 2 || rel == 4)
            return 1;
        return 2;
    }
    //求直线和圆的交点，返回交点个数
    int pointcrossline(Line v,Point &p1,Point &p2){
        if(!(*this).relationline(v))return 0;
        Point a = v.lineprog(p);
        double d = v.dispointtoline(p);
        d = sqrt(r*r-d*d);
        if(sgn(d) == 0){
            p1 = a;
            p2 = a;
            return 1;
        }
        p1 = a + (v.e-v.s).trunc(d);
        p2 = a - (v.e-v.s).trunc(d);
        return 2;
    }
    //得到过 a,b 两点，半径为 r1 的两个圆
    int gercircle(Point a,Point b,double r1,circle &c1,circle &c2){
        circle x(a,r1),y(b,r1);
        int t = x.pointcrosscircle(y,c1.p,c2.p);
        if(!t)return 0;
        c1.r = c2.r = r;
        return t;
    }
    //得到与直线 u 相切，过点 q，半径为 r1 的圆
    //测试：UVA12304
    int getcircle(Line u,Point q,double r1,circle &c1,circle &c2){
        double dis = u.dispointtoline(q);
        if(sgn(dis-r1*2)>0)return 0;
        if(sgn(dis) == 0){
            c1.p = q + ((u.e-u.s).rotleft().trunc(r1));
            c2.p = q + ((u.e-u.s).rotright().trunc(r1));
            c1.r = c2.r = r1;
            return 2;
        }
        Line u1 = Line((u.s + (u.e-u.s).rotleft().trunc(r1)),(u.e + (u.e-u.s).rotleft().trunc(r1)));
        Line u2 = Line((u.s + (u.e-u.s).rotright().trunc(r1)),(u.e + (u.e-u.s).rotright().trunc(r1)));
        circle cc = circle(q,r1);
        Point p1,p2;
        if(!cc.pointcrossline(u1,p1,p2))cc.pointcrossline(u2,p1,p2);
        c1 = circle(p1,r1);
        if(p1 == p2){
            c2 = c1;
            return 1;
        }
        c2 = circle(p2,r1);
        return 2;
    }
    //同时与直线 u,v 相切，半径为 r1 的圆
    //测试：UVA12304
    int getcircle(Line u,Line v,double r1,circle &c1,circle &c2,circle &c3,circle &c4){
        if(u.parallel(v))return 0;//两直线平行
        Line u1 = Line(u.s + (u.e-u.s).rotleft().trunc(r1),u.e + (u.e-u.s).rotleft().trunc(r1));
        Line u2 = Line(u.s + (u.e-u.s).rotright().trunc(r1),u.e + (u.e-u.s).rotright().trunc(r1));
        Line v1 = Line(v.s + (v.e-v.s).rotleft().trunc(r1),v.e + (v.e-v.s).rotleft().trunc(r1));
        Line v2 = Line(v.s + (v.e-v.s).rotright().trunc(r1),v.e + (v.e-v.s).rotright().trunc(r1));
        c1.r = c2.r = c3.r = c4.r = r1;
        c1.p = u1.crosspoint(v1);
        c2.p = u1.crosspoint(v2);
        c3.p = u2.crosspoint(v1);
        c4.p = u2.crosspoint(v2);
        return 4;
    }
    //同时与不相交圆 cx,cy 相切，半径为 r1 的圆
    //测试：UVA12304
    int getcircle(circle cx,circle cy,double r1,circle &c1,circle &c2){
        circle x(cx.p,r1+cx.r),y(cy.p,r1+cy.r);
        int t = x.pointcrosscircle(y,c1.p,c2.p);
        if(!t)return 0;
        c1.r = c2.r = r1;
        return t;
    }

    //过一点作圆的切线（先判断点和圆的关系）
    //测试：UVA12304
    int tangentline(Point q,Line &u,Line &v){
        int x = relation(q);
        if(x == 2)return 0;
        if(x == 1){
            u = Line(q,q + (q-p).rotleft());
            v = u;
            return 1;
        }
        double d = p.distance(q);
        double l = r*r/d;
        double h = sqrt(r*r-l*l);
        u = Line(q,p + ((q-p).trunc(l) + (q-p).rotleft().trunc(h)))
           ;
        v = Line(q,p + ((q-p).trunc(l) + (q-p).rotright().trunc(h))
           );
        return 2;
    }
    //求两圆相交的面积
    double areacircle(circle v){
        int rel = relationcircle(v);
        if(rel >= 4)return 0.0;
        if(rel <= 2)return min(area(),v.area());
        double d = p.distance(v.p);
        double hf = (r+v.r+d)/2.0;
        double ss = 2*sqrt(hf*(hf-r)*(hf-v.r)*(hf-d));
        double a1 = acos((r*r+d*d-v.r*v.r)/(2.0*r*d));
        a1 = a1*r*r;
        double a2 = acos((v.r*v.r+d*d-r*r)/(2.0*v.r*d));
        a2 = a2*v.r*v.r;
        return a1+a2-ss;
    }
    //求圆和三角形 pab 的相交面积
    //测试：POJ3675 HDU3982 HDU2892
    double areatriangle(Point a,Point b){
        if(sgn((p-a)^(p-b)) == 0)return 0.0;
        Point q[5];
        int len = 0;
        q[len++] = a;
        Line l(a,b);
        Point p1,p2;
        if(pointcrossline(l,q[1],q[2])==2){
            if(sgn((a-q[1])*(b-q[1]))<0)q[len++] = q[1];
            if(sgn((a-q[2])*(b-q[2]))<0)q[len++] = q[2];
        }
        q[len++] = b;
        if(len == 4 && sgn((q[0]-q[1])*(q[2]-q[1]))>0)swap(q[1],q
           [2]);
        double res = 0;
        for(int i = 0;i < len-1;i++){
            if(relation(q[i])==0||relation(q[i+1])==0){
                double arg = p.rad(q[i],q[i+1]);
                res += r*r*arg/2.0;
            }
            else{
                res += fabs((q[i]-p)^(q[i+1]-p))/2.0;
            }
        }
        return res;
    }
};

/*
 * n,p  Line l for each side
 * input(int _n)                        - inputs _n size polygon
 * add(Point q)                         - adds a point at end of
    the list
 * getline()                            - populates line array
 * cmp                                  - comparision in
    convex_hull order
 * norm()                               - sorting in convex_hull
    order
 * getconvex(polygon &convex)           - returns convex hull in
    convex
 * Graham(polygon &convex)              - returns convex hull in
    convex
 * isconvex()                           - checks if convex
 * relationpoint(Point q)               - returns 3 if q is a
    vertex
 *                                                2 if on a side
 *                                                1 if inside
 *                                                0 if outside
 * convexcut(Line u,polygon &po)        - left side of u in po
 * gercircumference()                   - returns side length
 * getarea()                            - returns area
 * getdir()                             - returns 0 for cw, 1 for
    ccw
 * getbarycentre()                      - returns barycenter
 *
 */
struct polygon{
    int n;
    Point p[maxp];
    Line l[maxp];
    void input(int _n){
        n = _n;
        for(int i = 0;i < n;i++)
            p[i].input();
    }
    void add(Point q){
        p[n++] = q;
    }
    void getline(){
        for(int i = 0;i < n;i++){
            l[i] = Line(p[i],p[(i+1)%n]);
        }
    }
    struct cmp{
        Point p;
        cmp(const Point &p0){p = p0;}
        bool operator()(const Point &aa,const Point &bb){
            Point a = aa, b = bb;
            int d = sgn((a-p)^(b-p));
            if(d == 0){
                return sgn(a.distance(p)-b.distance(p)) < 0;
            }
            return d > 0;
        }
    };
    //进行极角排序
    //首先需要找到最左下角的点
    //需要重载号好 Point 的 < 操作符 (min 函数要用)
    void norm(){
        Point mi = p[0];
        for(int i = 1;i < n;i++)mi = min(mi,p[i]);
        sort(p,p+n,cmp(mi));
    }
    //得到凸包
    //得到的凸包里面的点编号是 0∼n-1 的
    //两种凸包的方法
    //注意如果有影响，要特判下所有点共点，或者共线的特殊情况
    //测试 LightOJ1203 LightOJ1239
    void getconvex(polygon &convex){
        sort(p,p+n);
        convex.n = n;
        for(int i = 0;i < min(n,2);i++){
            convex.p[i] = p[i];
        }
        if(convex.n == 2 && (convex.p[0] == convex.p[1]))convex.n
          --;//特
          判
        if(n <= 2)return;
        int &top = convex.n;
        top = 1;
        for(int i = 2;i < n;i++){
            while(top && sgn((convex.p[top]-p[i])^(convex.p[top-1]-
               p[i])) <= 0)
                top--;
            convex.p[++top] = p[i];
        }
        int temp = top;
        convex.p[++top] = p[n-2];
        for(int i = n-3;i >= 0;i--){
            while(top != temp && sgn((convex.p[top]-p[i])^(convex.p
               [top-1]-p[i])) <= 0)
                top--;
            convex.p[++top] = p[i];
        }
        if(convex.n == 2 && (convex.p[0] == convex.p[1]))convex.n
            --;//特
            判
        convex.norm();//原来得到的是顺时针的点，排序后逆时针
    }
    //得到凸包的另外一种方法
    //测试 LightOJ1203 LightOJ1239
    void Graham(polygon &convex){
        norm();
        int &top = convex.n;
        top = 0;
        if(n == 1){
            top = 1;
            convex.p[0] = p[0];
            return;
        }
        if(n == 2){
            top = 2;
            convex.p[0] = p[0];
            convex.p[1] = p[1];
            if(convex.p[0] == convex.p[1])top--;
            return;
        }
        convex.p[0] = p[0];
        convex.p[1] = p[1];
        top = 2;
        for(int i = 2;i < n;i++){
            while( top > 1 && sgn((convex.p[top-1]-convex.p[top-2])
                ^(p[i]-convex.p[top-2])) <= 0 )
                top--;
            convex.p[top++] = p[i];
        }
        if(convex.n == 2 && (convex.p[0] == convex.p[1]))convex.n
            --;//特
            判
    }
    //判断是不是凸的
    bool isconvex(){
        bool s[2];
        memset(s,false,sizeof(s));
        for(int i = 0;i < n;i++){
            int j = (i+1)%n;
            int k = (j+1)%n;
            s[sgn((p[j]-p[i])^(p[k]-p[i]))+1] = true;
            if(s[0] && s[2])return false;
        }
        return true;
    }
    //判断点和任意多边形的关系
    // 3 点上
    // 2 边上
    // 1 内部
    // 0 外部
    int relationpoint(Point q){
        for(int i = 0;i < n;i++){
            if(p[i] == q)return 3;
        }
        getline();
        for(int i = 0;i < n;i++){
            if(l[i].pointonseg(q))return 2;
        }
        int cnt = 0;
        for(int i = 0;i < n;i++){
            int j = (i+1)%n;
            int k = sgn((q-p[j])^(p[i]-p[j]));
            int u = sgn(p[i].y-q.y);
            int v = sgn(p[j].y-q.y);
            if(k > 0 && u < 0 && v >= 0)cnt++;
            if(k < 0 && v < 0 && u >= 0)cnt--;
        }
        return cnt != 0;
    }
    //直线 u 切割凸多边形左侧
    //注意直线方向
    //测试：HDU3982
    void convexcut(Line u,polygon &po){
        int &top = po.n;//注意引用
        top = 0;
        for(int i = 0;i < n;i++){
            int d1 = sgn((u.e-u.s)^(p[i]-u.s));
            int d2 = sgn((u.e-u.s)^(p[(i+1)%n]-u.s));
            if(d1 >= 0)po.p[top++] = p[i];
            if(d1*d2 < 0)po.p[top++] = u.crosspoint(Line(p[i],p[(i
               +1)%n]));
        }
    }
    //得到周长
    //测试 LightOJ1239
    double getcircumference(){
        double sum = 0;
        for(int i = 0;i < n;i++){
            sum += p[i].distance(p[(i+1)%n]);
        }
        return sum;
    }
    //得到面积
    double getarea(){
        double sum = 0;
        for(int i = 0;i < n;i++){
            sum += (p[i]^p[(i+1)%n]);
        }
        return fabs(sum)/2;
    }
    //得到方向
    // 1 表示逆时针，0 表示顺时针
    bool getdir(){
        double sum = 0;
        for(int i = 0;i < n;i++)
            sum += (p[i]^p[(i+1)%n]);
        if(sgn(sum) > 0)return 1;
        return 0;
    }
    //得到重心
    Point getbarycentre(){
        Point ret(0,0);
        double area = 0;
        for(int i = 1;i < n−1;i++){
            double tmp = (p[i]−p[0])^(p[i+1]−p[0]);
            if(sgn(tmp) == 0)continue;
            area += tmp;
            ret.x += (p[0].x+p[i].x+p[i+1].x)/3*tmp;
            ret.y += (p[0].y+p[i].y+p[i+1].y)/3*tmp;
        }
        if(sgn(area)) ret = ret/area;
        return ret;
    }
    //多边形和圆交的面积
    //测试：POJ3675 HDU3982 HDU2892
    double areacircle(circle c){
        double ans = 0;
        for(int i = 0;i < n;i++){
            int j = (i+1)%n;
            if(sgn( (p[j]−c.p)^(p[i]−c.p) ) >= 0)
                ans += c.areatriangle(p[i],p[j]);
            else ans −= c.areatriangle(p[i],p[j]);
        }
        return fabs(ans);
    }
    //多边形和圆关系
    // 2 圆完全在多边形内
    // 1 圆在多边形里面，碰到了多边形边界
    // 0 其它
    int relationcircle(circle c){
        getline();
        int x = 2;
        if(relationpoint(c.p) != 1)return 0;//圆心不在内部
        for(int i = 0;i < n;i++){
            if(c.relationseg(l[i])==2)return 0;
            if(c.relationseg(l[i])==1)x = 1;
        }
        return x;
    }
};
//AB X AC
double cross(Point A,Point B,Point C){
    return (B-A)^(C-A);
}
//AB*AC
double dot(Point A,Point B,Point C){
    return (B-A)*(C-A);
}
//最小矩形面积覆盖
// A 必须是凸包 (而且是逆时针顺序)
// 测试 UVA 10173
double minRectangleCover(polygon A){
    //要特判 A.n < 3 的情况
    if(A.n < 3)return 0.0;
    A.p[A.n] = A.p[0];
    double ans = -1;
    int r = 1, p = 1, q;
    for(int i = 0;i < A.n;i++){
        //卡出离边 A.p[i] - A.p[i+1] 最远的点
        while( sgn( cross(A.p[i],A.p[i+1],A.p[r+1]) - cross(A.p[i],
           A.p[i+1],A.p[r]) ) >= 0 )
            r = (r+1)%A.n;
        //卡出 A.p[i] - A.p[i+1] 方向上正向 n 最远的点
        while(sgn( dot(A.p[i],A.p[i+1],A.p[p+1]) - dot(A.p[i],A.p[i
           +1],A.p[p]) ) >= 0 )
            p = (p+1)%A.n;
        if(i == 0)q = p;
        //卡出 A.p[i] - A.p[i+1] 方向上负向最远的点
        while(sgn(dot(A.p[i],A.p[i+1],A.p[q+1]) - dot(A.p[i],A.p[i
           +1],A.p[q])) <= 0)
            q = (q+1)%A.n;
        double d = (A.p[i] - A.p[i+1]).len2();
        double tmp = cross(A.p[i],A.p[i+1],A.p[r]) *
            (dot(A.p[i],A.p[i+1],A.p[p]) - dot(A.p[i],A.p[i+1],A.p[
               q]))/d;
        if(ans < 0 || ans > tmp)ans = tmp;
    }
    return ans;
}

//直线切凸多边形
//多边形是逆时针的，在 q1q2 的左侧
//测试:HDU3982
vector<Point> convexCut(const vector<Point> &ps,Point q1,Point q2){
    vector<Point>qs;
    int n = ps.size();
    for(int i = 0;i < n;i++){
        Point p1 = ps[i], p2 = ps[(i+1)%n];
        int d1 = sgn((q2-q1)^(p1-q1)), d2 = sgn((q2-q1)^(p2-q1));
        if(d1 >= 0)
            qs.push_back(p1);
        if(d1 * d2 < 0)
            qs.push_back(Line(p1,p2).crosspoint(Line(q1,q2)));
    }
    return qs;
}
//半平面交
//测试 POJ3335 POJ1474 POJ1279
//***************************
struct halfplane:public Line{
    double angle;
    halfplane(){}
    //表示向量 s->e 逆时针 (左侧) 的半平面
    halfplane(Point _s,Point _e){
        s = _s;
        e = _e;
    }
    halfplane(Line v){
        s = v.s;
        e = v.e;
    }
    void calcangle(){
        angle = atan2(e.y-s.y,e.x-s.x);
    }
    bool operator <(const halfplane &b)const{
        return angle < b.angle;
    }
};
struct halfplanes{
    int n;
    halfplane hp[maxp];
    Point p[maxp];
    int que[maxp];
    int st,ed;
    void push(halfplane tmp){
        hp[n++] = tmp;
    }
    //去重
    void unique(){
        int m = 1;
        for(int i = 1;i < n;i++){
            if(sgn(hp[i].angle-hp[i-1].angle) != 0)
                hp[m++] = hp[i];
            else if(sgn( (hp[m-1].e-hp[m-1].s)^(hp[i].s-hp[m-1].s)
               ) > 0)
                hp[m-1] = hp[i];
        }
        n = m;
    }
    bool halfplaneinsert(){
        for(int i = 0;i < n;i++)hp[i].calcangle();
        sort(hp,hp+n);
        unique();
        que[st=0] = 0;
        que[ed=1] = 1;
        p[1] = hp[0].crosspoint(hp[1]);
        for(int i = 2;i < n;i++){
            while(st<ed && sgn((hp[i].e−hp[i].s)^(p[ed]−hp[i].s))
               <0)ed−−;
            while(st<ed && sgn((hp[i].e−hp[i].s)^(p[st+1]−hp[i].s))
               <0)st++;
            que[++ed] = i;
            if(hp[i].parallel(hp[que[ed−1]]))return false;
            p[ed]=hp[i].crosspoint(hp[que[ed−1]]);
        }
        while(st<ed && sgn((hp[que[st]].e−hp[que[st]].s)^(p[ed]−hp[
           que[st]].s))<0)ed−−;
        while(st<ed && sgn((hp[que[ed]].e−hp[que[ed]].s)^(p[st+1]−
           hp[que[ed]].s))<0)st++;
        if(st+1>=ed)return false;
        return true;
    }
    //得到最后半平面交得到的凸多边形
    //需要先调用 halfplaneinsert() 且返回 true
    void getconvex(polygon &con){
        p[st] = hp[que[st]].crosspoint(hp[que[ed]]);
        con.n = ed−st+1;
        for(int j = st,i = 0;j <= ed;i++,j++)
            con.p[i] = p[j];
    }
};
//*************************** 

const int maxn = 1010;
struct circles{
    circle c[maxn];
    double ans[maxn];//ans[i] 表示被覆盖了 i 次的面积
    double pre[maxn];
    int n;
    circles(){}
    void add(circle cc){
        c[n++] = cc;
    }
    //x 包含在 y 中
    bool inner(circle x,circle y){
        if(x.relationcircle(y) != 1)return 0;
        return sgn(x.r−y.r)<=0?1:0;
    }
    //圆的面积并去掉内含的圆
    void init_or(){
        bool mark[maxn] = {0};
        int i,j,k=0;
        for(i = 0;i < n;i++){
            for(j = 0;j < n;j++)
                if(i != j && !mark[j]){
                    if( (c[i]==c[j])||inner(c[i],c[j]) )break;
                }
            if(j < n)mark[i] = 1;
        }
        for(i = 0;i < n;i++)
            if(!mark[i])
                c[k++] = c[i];
        n = k;
    }
    //圆的面积交去掉内含的圆
    void init_add(){
        int i,j,k;
        bool mark[maxn] = {0};
        for(i = 0;i < n;i++){
            for(j = 0;j < n;j++)
                if(i != j && !mark[j]){
                    if( (c[i]==c[j])||inner(c[j],c[i]) )break;
                }
            if(j < n)mark[i] = 1;
        }
        for(i = 0;i < n;i++)
            if(!mark[i])
                c[k++] = c[i];
        n = k;
    }
    //半径为 r 的圆，弧度为 th 对应的弓形的面积
    double areaarc(double th,double r){
        return 0.5*r*r*(th-sin(th));
    }
    //测试 SPOJVCIRCLES SPOJCIRUT
    //SPOJVCIRCLES 求 n 个圆并的面积，需要加上 init_or() 去掉重复圆（否则
       WA）
    //SPOJCIRUT 是求被覆盖 k 次的面积，不能加 init_or()
    //对于求覆盖多少次面积的问题，不能解决相同圆，而且不能 init_or()
    //求多圆面积并，需要 init_or，其中一个目的就是去掉相同圆
    void getarea(){
        memset(ans,0,sizeof(ans));
        vector<pair<double,int> >v;
        for(int i = 0;i < n;i++){
            v.clear();
            v.push_back(make_pair(-pi,1));
            v.push_back(make_pair(pi,-1));
            for(int j = 0;j < n;j++)
                if(i != j){
                    Point q = (c[j].p - c[i].p);
                    double ab = q.len(),ac = c[i].r, bc = c[j].r;
                    if(sgn(ab+ac-bc)<=0){
                        v.push_back(make_pair(-pi,1));
                        v.push_back(make_pair(pi,-1));
                        continue;
                    }
                    if(sgn(ab+bc-ac)<=0)continue;
                    if(sgn(ab-ac-bc)>0)continue;
                    double th = atan2(q.y,q.x), fai = acos((ac*ac+ab*ab-bc*bc)/(2.0*ac*ab));
                    double a0 = th-fai;
                    if(sgn(a0+pi)<0)a0+=2*pi;
                    double a1 = th+fai;
                    if(sgn(a1-pi)>0)a1-=2*pi;
                    if(sgn(a0-a1)>0){
                        v.push_back(make_pair(a0,1));
                        v.push_back(make_pair(pi,-1));
                        v.push_back(make_pair(-pi,1));
                        v.push_back(make_pair(a1,-1));
                    }
                    else{
                        v.push_back(make_pair(a0,1));
                        v.push_back(make_pair(a1,-1));
                    }
                }
            sort(v.begin(),v.end());
            int cur = 0;
            for(int j = 0;j < v.size();j++){
                if(cur && sgn(v[j].first-pre[cur])){
                    ans[cur] += areaarc(v[j].first-pre[cur],c[i].r);
                    ans[cur] += 0.5*(Point(c[i].p.x+c[i].r*cos(pre[cur]),c[i].p.y+c[i].r*sin(pre[cur]))^Point(c[i].p.x+c[i].r*cos(v[j].first),c[i].p.y+c[i].r*sin(v[j].first)));
                }
                cur += v[j].second;
                pre[cur] = v[j].first;
            }
        }
        for(int i = 1;i < n;i++)
            ans[i] -= ans[i+1];
    }
};
```

7.2 三维几何

```
const double eps = 1e-8;
int sgn(double x){
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
struct Point3{
    double x,y,z;
    Point3(double _x = 0,double _y = 0,double _z = 0){
        x = _x;
        y = _y;
        z = _z;
    }
    void input(){
        scanf("%lf%lf%lf",&x,&y,&z);
    }
    void output(){
        scanf("%.2lf␣%.2lf␣%.2lf\n",x,y,z);
    }
    bool operator ==(const Point3 &b)const{
        return sgn(x-b.x) == 0 && sgn(y-b.y) == 0 && sgn(z-b.z) ==
           0;
    }
    bool operator <(const Point3 &b)const{
        return sgn(x-b.x)==0?(sgn(y-b.y)==0?sgn(z-b.z)<0:y<b.y):x<b
           .x;
    }
    double len(){
        return sqrt(x*x+y*y+z*z);
    }
    double len2(){
        return x*x+y*y+z*z;
    }
    double distance(const Point3 &b)const{
        return sqrt((x-b.x)*(x-b.x)+(y-b.y)*(y-b.y)+(z-b.z)*(z-b.z)
           );
    }
    Point3 operator -(const Point3 &b)const{
        return Point3(x-b.x,y-b.y,z-b.z);
    }
    Point3 operator +(const Point3 &b)const{
        return Point3(x+b.x,y+b.y,z+b.z);
    }
    Point3 operator *(const double &k)const{
        return Point3(x*k,y*k,z*k);
    }
    Point3 operator /(const double &k)const{
        return Point3(x/k,y/k,z/k);
    }
    //点乘
    double operator *(const Point3 &b)const{
        return x*b.x+y*b.y+z*b.z;
    }
    //叉乘
    Point3 operator ^(const Point3 &b)const{
        return Point3(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);
    }
    double rad(Point3 a,Point3 b){
        Point3 p = (*this);
        return acos( ( (a-p)*(b-p) )/ (a.distance(p)*b.distance(p))
            );
    }
    //变换长度
    Point3 trunc(double r){
        double l = len();
        if(!sgn(l))return *this;
        r /= l;
        return Point3(x*r,y*r,z*r);
    }
};
struct Line3
{
    Point3 s,e;
    Line3(){}
    Line3(Point3 _s,Point3 _e)
    {
        s = _s;
        e = _e;
    }
    bool operator ==(const Line3 v)
    {
        return (s==v.s)&&(e==v.e);
    }
    void input()
    {
        s.input();
        e.input();
    }
    double length()
    {
        return s.distance(e);
    }
    //点到直线距离
    double dispointtoline(Point3 p)
    {
        return ((e-s)^(p-s)).len()/s.distance(e);
    }
    //点到线段距离
    double dispointtoseg(Point3 p)
    {
        if(sgn((p-s)*(e-s)) < 0 || sgn((p-e)*(s-e)) < 0)
            return min(p.distance(s),e.distance(p));
        return dispointtoline(p);
    }
    //返回点 p 在直线上的投影
    Point3 lineprog(Point3 p)
    {
        return s + ( ((e-s)*((e-s)*(p-s)))/((e-s).len2()) );
    }
    //p 绕此向量逆时针 arg 角度
    Point3 rotate(Point3 p,double ang)
    {
        if(sgn(((s-p)^(e-p)).len()) == 0)return p;
        Point3 f1 = (e-s)^(p-s);
        Point3 f2 = (e-s)^(f1);
        double len = ((s-p)^(e-p)).len()/s.distance(e);
        f1 = f1.trunc(len); f2 = f2.trunc(len);
        Point3 h = p+f2;
        Point3 pp = h+f1;
        return h + ((p-h)*cos(ang)) + ((pp-h)*sin(ang));
    }
    //点在直线上
    bool pointonseg(Point3 p)
    {
        return sgn( ((s-p)^(e-p)).len() ) == 0 && sgn((s-p)*(e-p))
           == 0;
    }
};
struct Plane
{
    Point3 a,b,c,o;//平面上的三个点，以及法向量
    Plane(){}
    Plane(Point3 _a,Point3 _b,Point3 _c)
    {
        a = _a;
        b = _b;
        c = _c;
        o = pvec();
    }
    Point3 pvec()
    {
        return (b-a)^(c-a);
    }
    //ax+by+cz+d = 0
    Plane(double _a,double _b,double _c,double _d)
    {
        o = Point3(_a,_b,_c);
        if(sgn(_a) != 0)
            a = Point3((-_d-_c-_b)/_a,1,1);
        else if(sgn(_b) != 0)
            a = Point3(1,(-_d-_c-_a)/_b,1);
        else if(sgn(_c) != 0)
            a = Point3(1,1,(-_d-_a-_b)/_c);
    }
    //点在平面上的判断
    bool pointonplane(Point3 p)
    {
        return sgn((p-a)*o) == 0;
    }
    //两平面夹角
    double angleplane(Plane f)
    {
        return acos(o*f.o)/(o.len()*f.o.len());
    }
    //平面和直线的交点，返回值是交点个数
    int crossline(Line3 u,Point3 &p)
    {
        double x = o*(u.e-a);
        double y = o*(u.s-a);
        double d = x-y;
        if(sgn(d) == 0)return 0;
        p = ((u.s*x)-(u.e*y))/d;
        return 1;
    }
    //点到平面最近点 (也就是投影)
    Point3 pointtoplane(Point3 p)
    {
        Line3 u = Line3(p,p+o);
        crossline(u,p);
        return p;
    }
    //平面和平面的交线
    int crossplane(Plane f,Line3 &u)
    {
        Point3 oo = o^f.o;
        Point3 v = o^oo;
        double d = fabs(f.o*v);
        if(sgn(d) == 0)return 0;
        Point3 q = a + (v*(f.o*(f.a-a))/d);
        u = Line3(q,q+oo);
        return 1;
    }
};
```

7.3 平面最近点对

HDU1007/ZOJ2107

```
const int MAXN = 100010;
const double eps = 1e-8;
const double INF = 1e20;
struct Point{
    double x,y;
    void input(){
        scanf("%lf%lf",&x,&y);
    }
};
double dist(Point a,Point b){
    return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
Point p[MAXN];
Point tmpt[MAXN];
bool cmpx(Point a,Point b){
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}
bool cmpy(Point a,Point b){
    return a.y < b.y || (a.y == b.y && a.x < b.x);
}
double Closest_Pair(int left,int right){
    double d = INF;
    if(left == right)return d;
    if(left+1 == right)return dist(p[left],p[right]);
    int mid = (left+right)/2;
    double d1 = Closest_Pair(left,mid);
    double d2 = Closest_Pair(mid+1,right);
    d = min(d1,d2);
    int cnt = 0;
    for(int i = left;i <= right;i++){
        if(fabs(p[mid].x - p[i].x) <= d)
            tmpt[cnt++] = p[i];
    }
    sort(tmpt,tmpt+cnt,cmpy);
    for(int i = 0;i < cnt;i++){
        for(int j = i+1;j < cnt && tmpt[j].y - tmpt[i].y < d;j++)
            d = min(d,dist(tmpt[i],tmpt[j]));
    }
    return d;
}
int main(){
    int n;
    while(scanf("%d",&n) == 1 && n){
        for(int i = 0;i < n;i++)p[i].input();
        sort(p,p+n,cmpx);
        printf("%.2lf\n",Closest_Pair(0,n-1));
    }
    return 0;
}
```

7.4 三维凸包

7.4.1 HDU4273

HDU 4273 给一个三维凸包，求重心到表面的最短距离。

```
const double eps = 1e-8;
const int MAXN = 550;
int sgn(double x){
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
struct Point3{
    double x,y,z;
    Point3(double _x = 0, double _y = 0, double _z = 0){
        x = _x;
        y = _y;
        z = _z;
    }
    void input(){
        scanf("%lf%lf%lf",&x,&y,&z);
    }
    bool operator ==(const Point3 &b)const{
        return sgn(x-b.x) == 0 && sgn(y-b.y) == 0 && sgn(z-b.z) ==
           0;
    }
    double len(){
        return sqrt(x*x+y*y+z*z);
    }
    double len2(){
        return x*x+y*y+z*z;
    }
    double distance(const Point3 &b)const{
        return sqrt((x-b.x)*(x-b.x)+(y-b.y)*(y-b.y)+(z-b.z)*(z-b.z)
           );
    }
    Point3 operator -(const Point3 &b)const{
        return Point3(x-b.x,y-b.y,z-b.z);
    }
    Point3 operator +(const Point3 &b)const{
        return Point3(x+b.x,y+b.y,z+b.z);
    }
    Point3 operator *(const double &k)const{
        return Point3(x*k,y*k,z*k);
    }
    Point3 operator /(const double &k)const{
        return Point3(x/k,y/k,z/k);
    }
    //点乘
    double operator *(const Point3 &b)const{
        return x*b.x + y*b.y + z*b.z;
    }
    //叉乘
    Point3 operator ^(const Point3 &b)const{
        return Point3(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);
    }
};
struct CH3D{
    struct face{
        //表示凸包一个面上的三个点的编号
        int a,b,c;
        //表示该面是否属于最终的凸包上的面
        bool ok;
    };
    //初始顶点数
    int n;
    Point3 P[MAXN];
    //凸包表面的三角形数
    int num;
    //凸包表面的三角形
    face F[8*MAXN];
    int g[MAXN][MAXN];
    //叉乘
    Point3 cross(const Point3 &a,const Point3 &b,const Point3 &c){
        return (b-a)^(c-a);
    }
    //三角形面积 *2
    double area(Point3 a,Point3 b,Point3 c){
        return ((b-a)^(c-a)).len();
    }
    //四面体有向面积 *6
    double volume(Point3 a,Point3 b,Point3 c,Point3 d){
        return ((b-a)^(c-a))*(d-a);
    }
    //正：点在面同向
    double dblcmp(Point3 &p,face &f){
        Point3 p1 = P[f.b] - P[f.a];
        Point3 p2 = P[f.c] - P[f.a];
        Point3 p3 = p - P[f.a];
        return (p1^p2)*p3;
    }
    void deal(int p,int a,int b){
        int f = g[a][b];
        face add;
        if(F[f].ok){
            if(dblcmp(P[p],F[f]) > eps)
                dfs(p,f);
            else {
                add.a = b;
                add.b = a;
                add.c = p;
                add.ok = true;
                g[p][b] = g[a][p] = g[b][a] = num;
                F[num++] = add;
            }
        }
    }
    //递归搜索所有应该从凸包内删除的面
    void dfs(int p,int now){
        F[now].ok = false;
        deal(p,F[now].b,F[now].a);
        deal(p,F[now].c,F[now].b);
        deal(p,F[now].a,F[now].c);
    }
    bool same(int s,int t){
        Point3 &a = P[F[s].a];
        Point3 &b = P[F[s].b];
        Point3 &c = P[F[s].c];
        return fabs(volume(a,b,c,P[F[t].a])) < eps &&
            fabs(volume(a,b,c,P[F[t].b])) < eps &&
            fabs(volume(a,b,c,P[F[t].c])) < eps;
    }
    //构建三维凸包
    void create(){
        num = 0;
        face add;

        //***********************************
        //此段是为了保证前四个点不共面
        bool flag = true;
        for(int i = 1;i < n;i++){
            if(!(P[0] == P[i])){
                swap(P[1],P[i]);
                flag = false;
                break;
            }
        }
        if(flag)return;
        flag = true;
        for(int i = 2;i < n;i++){
            if( ((P[1]-P[0])^(P[i]-P[0])).len() > eps ){
                swap(P[2],P[i]);
                flag = false;
                break;
            }
        }
        if(flag)return;
        flag = true;
        for(int i = 3;i < n;i++){
            if(fabs( ((P[1]-P[0])^(P[2]-P[0]))*(P[i]-P[0]) ) > eps)
               {
                swap(P[3],P[i]);
                flag = false;
                break;
            }
        }
        if(flag)return;
        //**********************************

        for(int i = 0;i < 4;i++){
            add.a = (i+1)%4;
            add.b = (i+2)%4;
            add.c = (i+3)%4;
            add.ok = true;
            if(dblcmp(P[i],add) > 0)swap(add.b,add.c);
            g[add.a][add.b] = g[add.b][add.c] = g[add.c][add.a] =
               num;
            F[num++] = add;
        }
        for(int i = 4;i < n;i++)
            for(int j = 0;j < num;j++)
                if(F[j].ok && dblcmp(P[i],F[j]) > eps){
                    dfs(i,j);
                    break;
                }
        int tmp = num;
        num = 0;
        for(int i = 0;i < tmp;i++)
            if(F[i].ok)
                F[num++] = F[i];
    }
    //表面积
    //测试：HDU3528
    double area(){
        double res = 0;
        if(n == 3){
            Point3 p = cross(P[0],P[1],P[2]);
            return p.len()/2;
        }
        for(int i = 0;i < num;i++)
            res += area(P[F[i].a],P[F[i].b],P[F[i].c]);
        return res/2.0;
    }
    double volume(){
        double res = 0;
        Point3 tmp = Point3(0,0,0);
        for(int i = 0;i < num;i++)
            res += volume(tmp,P[F[i].a],P[F[i].b],P[F[i].c]);
        return fabs(res/6);
    }
    //表面三角形个数
    int triangle(){
        return num;
    }
    //表面多边形个数
    //测试：HDU3662
    int polygon(){
        int res = 0;
        for(int i = 0;i < num;i++){
            bool flag = true;
            for(int j = 0;j < i;j++)
                if(same(i,j)){
                    flag = 0;
                    break;
                }
            res += flag;
        }
        return res;
    }
    //重心
    //测试：HDU4273
    Point3 barycenter(){
        Point3 ans = Point3(0,0,0);
        Point3 o = Point3(0,0,0);
        double all = 0;
        for(int i = 0;i < num;i++){
            double vol = volume(o,P[F[i].a],P[F[i].b],P[F[i].c]);
            ans = ans + (((o+P[F[i].a]+P[F[i].b]+P[F[i].c])/4.0)*
               vol);
            all += vol;
        }
        ans = ans/all;
        return ans;
    }
    //点到面的距离
    //测试：HDU4273
    double ptoface(Point3 p,int i){
        double tmp1 = fabs(volume(P[F[i].a],P[F[i].b],P[F[i].c],p))
           ;
        double tmp2 = ((P[F[i].b]-P[F[i].a])^(P[F[i].c]-P[F[i].a]))
           .len();
        return tmp1/tmp2;
    }
};
CH3D hull;
int main()
{
    while(scanf("%d",&hull.n) == 1){
        for(int i = 0;i < hull.n;i++)hull.P[i].input();
        hull.create();
        Point3 p = hull.barycenter();
        double ans = 1e20;
        for(int i = 0;i < hull.num;i++)
            ans = min(ans,hull.ptoface(p,i));
        printf("%.3lf\n",ans);
    }
    return 0;
}
```

8 其他

8.1 高精度

高精度，支持乘法和加法

```
/*
 * 高精度，支持乘法和加法
 */
struct BigInt{
    const static int mod = 10000;
    const static int DLEN = 4;
    int a[600],len;
    BigInt(){
        memset(a,0,sizeof(a));
        len = 1;
    }
    BigInt(int v){
        memset(a,0,sizeof(a));
        len = 0;
        do{
            a[len++] = v%mod;
            v /= mod;
        }while(v);
    }
    BigInt(const char s[]){
        memset(a,0,sizeof(a));
        int L = strlen(s);
        len = L/DLEN;
        if(L%DLEN)len++;
        int index = 0;
        for(int i = L-1;i >= 0;i -= DLEN){
            int t = 0;
            int k = i - DLEN + 1;
            if(k < 0)k = 0;
            for(int j = k;j <= i;j++)
                t = t*10 + s[j] - '0';
            a[index++] = t;
        }
    }
    BigInt operator +(const BigInt &b)const{
        BigInt res;
        res.len = max(len,b.len);
        for(int i = 0;i <= res.len;i++)
            res.a[i] = 0;
        for(int i = 0;i < res.len;i++){
            res.a[i] += ((i < len)?a[i]:0)+((i < b.len)?b.a[i]:0);
            res.a[i+1] += res.a[i]/mod;
            res.a[i] %= mod;
        }
        if(res.a[res.len] > 0)res.len++;
        return res;
    }
    BigInt operator *(const BigInt &b)const{
        BigInt res;
        for(int i = 0; i < len;i++){
            int up = 0;
            for(int j = 0;j < b.len;j++){
                int temp = a[i]*b.a[j] + res.a[i+j] + up;
                res.a[i+j] = temp%mod;
                up = temp/mod;
            }
            if(up != 0)
                res.a[i + b.len] = up;
        }
        res.len = len + b.len;
        while(res.a[res.len - 1] == 0 &&res.len > 1)res.len--;
        return res;
    }
    void output(){
        printf("%d",a[len-1]);
        for(int i = len-2;i >=0 ;i--)
            printf("%04d",a[i]);
        printf("\n");
    }
};
```

8.2 完全高精度

HDU 1134 求卡特兰数

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

    BigNum operator+(const BigNum &)const;   //重载加法运算符，两个大数
       之间的相加运算
    BigNum operator-(const BigNum &)const;   //重载减法运算符，两个大数
       之间的相减运算
    BigNum operator*(const BigNum &)const;   //重载乘法运算符，两个大数
       之间的相乘运算
    BigNum operator/(const int &)const;      //重载除法运算符，大数对一
       个整数进行相除运算

    BigNum operator^(const int &)const;      //大数的 n 次方运算
    int operator%(const int &)const;         //大数对一个类型的变量进行
       取模运算int
    bool operator>(const BigNum &T)const;    //大数和另一个大数的大小比
       较
    bool operator>(const int &t)const;       //大数和一个 int 类型的变
       量的大小比较

    void print();         //输出大数
};
//将一个 int 类型的变量转化为大数
BigNum::BigNum(const int b){
    int c,d=b;
    len=0;
    memset(a,0,sizeof(a));
    while(d>MAXN){
        c=d-(d/(MAXN+1))*(MAXN+1);
        d=d/(MAXN+1);
        a[len++]=c;
    }
    a[len++]=d;
}
//将一个字符串类型的变量转化为大数
BigNum::BigNum(const char *s){
    int t,k,index,L,i;
    memset(a,0,sizeof(a));
    L=strlen(s);
    len=L/DLEN;
    if(L%DLEN)len++;
    index=0;
    for(i=L-1;i>=0;i-=DLEN){
        t=0;
        k=i-DLEN+1;
        if(k<0)k=0;
        for(int j=k;j<=i;j++)
            t=t*10+s[j]-'0';
        a[index++]=t;
    }
}
//拷贝构造函数
BigNum::BigNum(const BigNum &T):len(T.len){
    int i;
    memset(a,0,sizeof(a));
    for(i=0;i<len;i++)
        a[i]=T.a[i];
}
//重载赋值运算符，大数之间赋值运算
BigNum & BigNum::operator=(const BigNum &n){
    int i;
    len=n.len;
    memset(a,0,sizeof(a));
    for(i=0;i<len;i++)
        a[i]=n.a[i];
    return *this;
}
istream& operator>>(istream &in,BigNum &b){
    char ch[MAXSIZE*4];
    int i=-1;
    in>>ch;
    int L=strlen(ch);
    int count=0,sum=0;
    for(i=L-1;i>=0;){
        sum=0;
        int t=1;
        for(int j=0;j<4&&i>=0;j++,i--,t*=10){
            sum+=(ch[i]-'0')*t;
        }
        b.a[count]=sum;
        count++;
    }
    b.len=count++;
    return in;
}
//重载输出运算符
ostream& operator<<(ostream& out,BigNum& b){
    int i;
    cout<<b.a[b.len-1];
    for(i=b.len-2;i>=0;i--){
        printf("%04d",b.a[i]);
    }
    return out;
}
//两个大数之间的相加运算
BigNum BigNum::operator+(const BigNum &T)const{
    BigNum t(*this);
    int i,big;
    big=T.len>len?T.len:len;
    for(i=0;i<big;i++){
        t.a[i]+=T.a[i];
        if(t.a[i]>MAXN){
            t.a[i+1]++;
            t.a[i]-=MAXN+1;
        }
    }
    if(t.a[big]!=0)
       t.len=big+1;
    else t.len=big;
    return t;
}
//两个大数之间的相减运算
BigNum BigNum::operator-(const BigNum &T)const{
    int i,j,big;
    bool flag;
    BigNum t1,t2;
    if(*this>T){
        t1=*this;
        t2=T;
        flag=0;
    }
    else{
        t1=T;
        t2=*this;
        flag=1;
    }
    big=t1.len;
    for(i=0;i<big;i++){
        if(t1.a[i]<t2.a[i]){
            j=i+1;
            while(t1.a[j]==0)
                j++;
            t1.a[j--]--;
            while(j>i)
                t1.a[j--]+=MAXN;
            t1.a[i]+=MAXN+1-t2.a[i];
        }
        else t1.a[i]-=t2.a[i];
    }
    t1.len=big;
    while(t1.a[t1.len-1]==0 && t1.len>1){
        t1.len--;
        big--;
    }
    if(flag)
        t1.a[big-1]=0-t1.a[big-1];
    return t1;
}
//两个大数之间的相乘
BigNum BigNum::operator*(const BigNum &T)const{
    BigNum ret;
    int i,j,up;
    int temp,temp1;
    for(i=0;i<len;i++){
        up=0;
        for(j=0;j<T.len;j++){
            temp=a[i]*T.a[j]+ret.a[i+j]+up;
            if(temp>MAXN){
                temp1=temp-temp/(MAXN+1)*(MAXN+1);
                up=temp/(MAXN+1);
                ret.a[i+j]=temp1;
            }
            else{
                up=0;
                ret.a[i+j]=temp;
            }
        }
        if(up!=0)
           ret.a[i+j]=up;
    }
    ret.len=i+j;
    while(ret.a[ret.len-1]==0 && ret.len>1)ret.len--;
    return ret;
}
//大数对一个整数进行相除运算
BigNum BigNum::operator/(const int &b)const{
    BigNum ret;
    int i,down=0;
    for(i=len-1;i>=0;i--){
        ret.a[i]=(a[i]+down*(MAXN+1))/b;
        down=a[i]+down*(MAXN+1)-ret.a[i]*b;
    }
    ret.len=len;
    while(ret.a[ret.len-1]==0 && ret.len>1)
        ret.len--;
    return ret;
}
//大数对一个 int 类型的变量进行取模
int BigNum::operator%(const int &b)const{
    int i,d=0;
    for(i=len-1;i>=0;i--)
        d=((d*(MAXN+1))%b+a[i])%b;
    return d;
}
//大数的 n 次方运算
BigNum BigNum::operator^(const int &n)const{
    BigNum t,ret(1);
    int i;
    if(n<0)exit(-1);
    if(n==0)return 1;
    if(n==1)return *this;
    int m=n;
    while(m>1){
        t=*this;
        for(i=1;(i<<1)<=m;i<<=1)
           t=t*t;
        m-=i;
        ret=ret*t;
        if(m==1)ret=ret*(*this);
    }
    return ret;
}
//大数和另一个大数的大小比较
bool BigNum::operator>(const BigNum &T)const{
    int ln;
    if(len>T.len)return true;
    else if(len==T.len){
        ln=len-1;
        while(a[ln]==T.a[ln]&&ln>=0)
          ln--;
        if(ln>=0 && a[ln]>T.a[ln])
           return true;
        else
           return false;
    }
    else
       return false;
}
//大数和一个 int 类型的变量的大小比较
bool BigNum::operator>(const int &t)const{
    BigNum b(t);
    return *this>b;
}
//输出大数
void BigNum::print(){
    int i;
    printf("%d",a[len-1]);
    for(i=len-2;i>=0;i--)
      printf("%04d",a[i]);
    printf("\n");
}
BigNum f[110];//卡特兰数

int main(){
    f[0]=1;
    for(int i=1;i<=100;i++)
        f[i]=f[i-1]*(4*i-2)/(i+1);//卡特兰数递推式
    int n;
    while(scanf("%d",&n)==1){
        if(n==-1)break;
        f[n].print();
    }
    return 0;
}
```

8.3 strtok 和 sscanf 结合输入

空格作为分隔输入，读取一行的整数：

```
            gets(buf);
            int v;
            char *p = strtok(buf,"␣");
            while(p)
            {
                sscanf(p,"%d",&v);
                p = strtok(NULL,"␣");
            }
```

8.4 解决爆栈，手动加栈

防止爆栈最好方法是改变写法，弄成 bfs，或者模拟栈。加栈都是旁门左道，需谨慎！
C++
放在头文件前面

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

G++
放在主函数里面 (汇编开栈不一定适用，和系统有关。需谨慎!)

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

8.5 STL

8.5.1 优先队列 priority_queue

empty() 如果队列为空返回真
pop() 删除对顶元素
push() 加入一个元素
size() 返回优先队列中拥有的元素个数
top() 返回优先队列队顶元素
在默认的优先队列中，优先级高的先出队。在默认的 int 型中先出队的为较大的数。

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

自定义比较函数：

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

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

8.5.2 set 和 multiset

set 和 multiset 用法一样，就是 multiset 允许重复元素。
元素放入容器时，会按照一定的排序法则自动排序，默认是按照 less<> 排序规则来排序。
不能修改容器里面的元素值，只能插入和删除。
自定义 int 排序函数：（默认的是从小到大的，下面这个从大到小）

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

上面这样就定义成了从大到小排列了。
结构体自定义排序函数：
（定义 set 或者 multiset 的时候定义了排序函数，定义迭代器时一样带上排序函数）

```
struct Node
{
    int x,y;
};
struct classcomp//先按照 x 从小到大排序，x 相同则按照 y 从大到小排序
{
    bool operator()(const Node &a,const Node &b)const
    {
        if(a.x!=b.x)return a.x<b.x;
        else return a.y>b.y;
    }
}; //注意这里有个逗号
multiset<Node,classcomp>mt;
multiset<Node,classcomp>::iterator it;
```

```

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

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

8.6 输入输出外挂

```

//适用于正负整数
template <class T>
inline bool scan_d(T &ret) {
   char c; int sgn;
   if(c=getchar(),c==EOF) return 0; //EOF
   while(c!='-'&&(c<'0'||c>'9')) c=getchar();
   sgn=(c=='-')?-1:1;
   ret=(c=='-')?0:(c-'0');
   while(c=getchar(),c>='0'&&c<='9') ret=ret*10+(c-'0');
   ret*=sgn;
   return 1;
}

inline void out(int x) {
   if(x>9) out(x/10);
   putchar(x%10+'0');
}
```

8.7 莫队算法

莫队算法，可以解决一类静态，离线区间查询问题。
BZOJ 2038: [2009 国家集训队] 小 Z 的袜子 (hose)
Description
作为一个生活散漫的人，小 Z 每天早上都要耗费很久从一堆五颜六色的袜子中找出一双来穿。终于有一天，小 Z 再也无法忍受这恼人的找袜子过程，于是他决定听天由命……具体来说，小 Z 把这 N 只袜子从 1 到 N 编号，然后从编号 L 到 R(L
Input
输入文件第一行包含两个正整数 N 和 M。N 为袜子的数量，M 为小 Z 所提的询问的数量。接下来一行包含 N 个正整数 Ci，其中 Ci 表示第 i 只袜子的颜色，相同的颜色用相同的数字表示。再接下来 M 行，每行两个正整数 L，R 表示一个询问。
Output
包含 M 行，对于每个询问在一行中输出分数 A/B 表示从该询问的区间 [L,R] 中随机抽出两只袜子颜色相同的概率。若该概率为 0 则输出 0/1，否则输出的 A/B 必须为最简分数。（详见样例）
Sample Input
6 4
1 2 3 3 3 2

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

题解：
只需要统计区间内各个数出现次数的平方和

莫队算法，两种方法，一种是直接分成 sqrt(n) 块，分块排序。
另外一种是求得曼哈顿距离最小生成树，根据 manhattan MST 的 dfs 序求解。

8.7.1 分块

```
const int MAXN = 50010;
const int MAXM = 50010;
struct Query
{
    int L,R,id;
}node[MAXM];
long long gcd(long long a,long long b){
    if(b == 0)return a;
    return gcd(b,a%b);
}
struct Ans{
    long long a,b;//分数 a/b
    void reduce()//分数化简
    {
        long long d = gcd(a,b);
        a /= d; b /= d;
    }
}ans[MAXM];
int a[MAXN];
int num[MAXN];
int n,m,unit;
bool cmp(Query a,Query b){
    if(a.L/unit != b.L/unit)return a.L/unit < b.L/unit;
    else return a.R < b.R;
}
void work(){
    long long temp = 0;
    memset(num,0,sizeof(num));
    int L = 1;
    int R = 0;
    for(int i = 0;i < m;i++){
        while(R < node[i].R){
            R++;
            temp -= (long long)num[a[R]]*num[a[R]];
            num[a[R]]++;
            temp += (long long)num[a[R]]*num[a[R]];
        }
        while(R > node[i].R){
            temp -= (long long)num[a[R]]*num[a[R]];
            num[a[R]]--;
            temp += (long long)num[a[R]]*num[a[R]];
            R--;
        }
        while(L < node[i].L){
            temp -= (long long)num[a[L]]*num[a[L]];
            num[a[L]]--;
            temp += (long long)num[a[L]]*num[a[L]];
            L++;
        }
        while(L > node[i].L){
            L--;
            temp -= (long long)num[a[L]]*num[a[L]];
            num[a[L]]++;
            temp += (long long)num[a[L]]*num[a[L]];
        }
        ans[node[i].id].a = temp - (R-L+1);
        ans[node[i].id].b = (long long)(R-L+1)*(R-L);
        ans[node[i].id].reduce();
    }
}
int main(){
    while(scanf("%d%d",&n,&m) == 2){
        for(int i = 1;i <= n;i++)
            scanf("%d",&a[i]);
        for(int i = 0;i < m;i++){
            node[i].id = i;
            scanf("%d%d",&node[i].L,&node[i].R);
        }
        unit = (int)sqrt(n);
        sort(node,node+m,cmp);
        work();
        for(int i = 0;i < m;i++)
            printf("%lld/%lld\n",ans[i].a,ans[i].b);
    }
    return 0;
}
```

8.7.2 Manhattan MST 的 dfs 顺序求解

```
const int MAXN = 50010;
const int MAXM = 50010;
const int INF = 0x3f3f3f3f;
struct Point{
    int x,y,id;
```

```
}p[MAXN],pp[MAXN];
bool cmp(Point a,Point b)
{
    if(a.x != b.x) return a.x < b.x;
    else return a.y < b.y;
}
//树状数组，找 y-x 大于当前的，但是 y+x 最小的
struct BIT{
    int min_val,pos;
    void init()
    {
        min_val = INF;
        pos = -1;
    }
}bit[MAXN];
struct Edge{
    int u,v,d;
}edge[MAXN<<2];
bool cmpedge(Edge a,Edge b){
    return a.d < b.d;
}
int tot;
int n;
int F[MAXN];
int find(int x){
    if(F[x] == -1) return x;
    else return F[x] = find(F[x]);
}
void addedge(int u,int v,int d){
    edge[tot].u = u;
    edge[tot].v = v;
    edge[tot++].d = d;
}
struct Graph{
    int to,next;
}e[MAXN<<1];
int total,head[MAXN];
void _addedge(int u,int v){
    e[total].to = v;
    e[total].next = head[u];
    head[u] = total++;
}
int lowbit(int x){
    return x&(-x);
}
void update(int i,int val,int pos){
    while(i > 0){
        if(val < bit[i].min_val){
            bit[i].min_val = val;
            bit[i].pos = pos;
        }
        i -= lowbit(i);
    }
}
int ask(int i,int m){
    int min_val = INF,pos = -1;
    while(i <= m){
        if(bit[i].min_val < min_val){
            min_val = bit[i].min_val;
            pos = bit[i].pos;
        }
        i += lowbit(i);
    }
    return pos;
}
int dist(Point a,Point b){
    return abs(a.x - b.x) + abs(a.y - b.y);
}
void Manhattan_minimum_spanning_tree(int n,Point p[]){
    int a[MAXN],b[MAXN];
    tot = 0;
    for(int dir = 0;dir < 4;dir++){
        if(dir == 1 || dir == 3){
            for(int i = 0;i < n;i++)
                swap(p[i].x,p[i].y);
        }
        else if(dir == 2){
            for(int i = 0;i < n;i++)
                p[i].x = -p[i].x;
        }
        sort(p,p+n,cmp);
        for(int i = 0;i < n;i++)
            a[i] = b[i] = p[i].y - p[i].x;
        sort(b,b+n);
        int m = unique(b,b+n) - b;
        for(int i = 1;i <= m;i++)
            bit[i].init();
        for(int i = n-1;i >= 0;i--){
            int pos = lower_bound(b,b+m,a[i]) - b + 1;
            int ans = ask(pos,m);
            if(ans != -1)
                addedge(p[i].id,p[ans].id,dist(p[i],p[ans]));
            update(pos,p[i].x+p[i].y,i);
        }
    }
    memset(F,-1,sizeof(F));
    sort(edge,edge+tot,cmpedge);
    total = 0;
    memset(head,-1,sizeof(head));
    for(int i = 0;i < tot;i++){
        int u = edge[i].u,  v = edge[i].v;
        int t1 = find(u), t2 = find(v);
        if(t1 != t2){
            F[t1] = t2;
            _addedge(u,v);
            _addedge(v,u);
        }
    }
}
int m;
int a[MAXN];
struct Ans{
    long long a,b;
}ans[MAXM];
long long temp ;
int num[MAXN];
void add(int l,int r){
    for(int i = l;i <= r;i++){
        temp -= (long long)num[a[i]]*num[a[i]];
        num[a[i]]++;
        temp += (long long)num[a[i]]*num[a[i]];
    }
}
void del(int l,int r){
    for(int i = l;i <= r;i++){
        temp -= (long long)num[a[i]]*num[a[i]];
        num[a[i]]--;
        temp += (long long)num[a[i]]*num[a[i]];
    }
}
void dfs(int l1,int r1,int l2,int r2,int idx,int pre){
    if(l2 < l1) add(l2,l1-1);
    if(r2 > r1) add(r1+1,r2);
    if(l2 > l1) del(l1,l2-1);
    if(r2 < r1) del(r2+1,r1);
    ans[pp[idx].id].a = temp - (r2-l2+1);
    ans[pp[idx].id].b = (long long)(r2-l2+1)*(r2-l2);
    for(int i = head[idx];i != -1;i = e[i].next){
        int v = e[i].to;
        if(v == pre) continue;
        dfs(l2,r2,pp[v].x,pp[v].y,v,idx);
    }
    if(l2 < l1)del(l2,l1-1);
    if(r2 > r1)del(r1+1,r2);
    if(l2 > l1)add(l1,l2-1);
    if(r2 < r1)add(r2+1,r1);
}
long long gcd(long long a,long long b){
    if(b == 0) return a;
    else return gcd(b,a%b);
}
int main(){
    while(scanf("%d%d",&n,&m) == 2){
```

```cpp
        for(int i = 1;i <= n;i++)
            scanf("%d",&a[i]);
        for(int i = 0;i < m;i++){
            scanf("%d%d",&p[i].x,&p[i].y);
            p[i].id = i;
            pp[i] = p[i];
        }
        Manhattan_minimum_spanning_tree(m,p);
        memset(num,0,sizeof(num));
        temp = 0;
        dfs(1,0,pp[0].x,pp[0].y,0,-1);
        for(int i = 0;i < m;i++){
            long long d = gcd(ans[i].a,ans[i].b);
            printf("%lld/%lld\n",ans[i].a/d,ans[i].b/d);
        }
    }
    return 0;
}
```

8.8 VIM 配置

```vim
set nu
set history=1000000

set tabstop=4
set shiftwidth=4
set smarttab

set cindent

colo evening

set nobackup
set noswapfile

set mouse=a
map <F6> :call CR()<CR>
func! CR()
exec "w"
exec "!g++␣%␣-o␣%<"
exec "!␣./%<"
endfunc

imap <c-]> {<cr>}<c-o>O<left><right>
map <F2> :call SetTitle()<CR>
func SetTitle()
let l = 0
let l = l + 1 | call setline(l,'#include␣<stdio.h>')
let l = l + 1 | call setline(l,'#include␣<string.h>')
let l = l + 1 | call setline(l,'#include␣<iostream>')
let l = l + 1 | call setline(l,'#include␣<algorithm>')
let l = l + 1 | call setline(l,'#include␣<vector>')
let l = l + 1 | call setline(l,'#include␣<queue>')
let l = l + 1 | call setline(l,'#include␣<set>')
let l = l + 1 | call setline(l,'#include␣<map>')
let l = l + 1 | call setline(l,'#include␣<string>')
let l = l + 1 | call setline(l,'#include␣<math.h>')
let l = l + 1 | call setline(l,'#include␣<stdlib.h>')
let l = l + 1 | call setline(l,'#include␣<time.h>')
let l = l + 1 | call setline(l,'using␣namespace␣std;')
let l = l + 1 | call setline(l,'')
let l = l + 1 | call setline(l,'int␣main()')
let l = l + 1 | call setline(l,'{')
let l = l + 1 | call setline(l,'␣␣␣␣//freopen("in.txt","r",stdin);')
let l = l + 1 | call setline(l,'␣␣␣␣//freopen("out.txt","w",stdout);')
let l = l + 1 | call setline(l,'␣␣␣␣')
let l = l + 1 | call setline(l,'␣␣␣␣return␣0;')
let l = l + 1 | call setline(l,'}')
endfunc
```

现场赛配置:

```
syntax on
set nu
set tabstop=4
set shiftwidth=4
set cin
colo evening
set mouse=a
```