# 打铁专用板

By xd_zhu

# 目录

# 字符串

**KMP 模版**

```
//普通
void getnext1(char *s)
{
    int i = 0,j = -1,len = strlen(s);
    ne[0] = -1;
    while(i < len)
    {
        if(j == -1 || s[i] == s[j]) ne[++i] = ++j;
        else    j = ne[j];
    }
}
//加快预处理
void getnext2(char *s)
{
    int i = 0,j = -1,len = strlen(s);
    ne[0] = -1;
    while(i < len)
    {
        if(j == -1 || s[i] == s[j])
        {
            if(s[++i] == s[++j])    ne[i] = ne[j];
            else    ne[i] = j;
        }
        else    j = ne[j];
    }
}
//返回 x 在 y 中出现的次数，可以重叠
int kmp(char *x,char *y)
{
    getnext1(x);
    int i = 0,j = 0,ans = 0,leny = strlen(y),lenx = strlen(x);
    while(i < leny)
    {
        if(j == -1 || x[j] == y[i])
        {
            i++,j++;
            if(j == lenx)
            {
                ans++;
                j = ne[j];
            }
        }
        else    j = ne[j];
    }
    return ans;
}
```

**浮动匹配**

```
//求浮动匹配
//l[i]表示 i 结点之前(包括 i 结点),小于 a[i]的个数
//le[i]表示 i 结点之前(包括 i 结点),小于等于 a[i]的个数
//l[i] == l[j] && le[i] == le[j],则匹配
int n,k,s,a[100005],b[25005],ne[25005],tree[30],l[100005],le[100005];

inline int lowbit(int x)
{
    return x&-x;
}

void update(int pos,int x)
{
```

```
    while(pos <= s)
    {
        tree[pos] += x;
        pos += lowbit(pos);
    }
}

int getsum(int pos)
{
    int sum = 0;
    while(pos)
    {
        sum += tree[pos];
        pos -= lowbit(pos);
    }
    return sum;
}

void getnext1()
{
    memset(tree,0,sizeof(tree));
    int i = 0,j = -1;
    ne[0] = -1;
    while(i < k)
    {
        if(j == -1 || getsum(b[i]-1) == l[j] && getsum(b[i]) == le[j])
        {
            ne[++i] = ++j;
            if(i < k)   update(b[i],1);
        }
        else
        {
            for(int t = i-j;t < i-ne[j];t++)  update(b[t],-1);
            j = ne[j];
        }
    }
}

vector<int> kmp()
{
    getnext1();
    memset(tree,0,sizeof(tree));
    vector<int> ans;
    int i = 0,j = 0;
    update(a[0],1);
    while(i < n)
    {
        if(j == -1 || getsum(a[i]-1) == l[j] && getsum(a[i]) == le[j])
        {
            i++,j++;
            if(i < n)   update(a[i],1);
            if(j == k)
            {
                ans.push_back(i-k+1);
                for(int t = i-j;t < i-ne[j];t++)  update(a[t],-1);
                j = ne[j];
            }
        }
        else
        {
            for(int t = i-j;t < i-ne[j];t++)  update(a[t],-1);
            j = ne[j];
        }
    }
```

```
    return ans;
}

int main()
{
    while(~scanf("%d%d%d",&n,&k,&s))
    {
        memset(tree,0,sizeof(tree));
        memset(l,0,sizeof(l));
        memset(le,0,sizeof(le));
        memset(ne,0,sizeof(ne));
        for(int i = 0;i < n;i++)    scanf("%d",&a[i]);
        for(int i = 0;i < k;i++)
        {
            scanf("%d",&b[i]);
            update(b[i],1);
            l[i] = getsum(b[i]-1);
            le[i] = getsum(b[i]);
        }
        vector<int> v = kmp();
        printf("%d\n",v.size());
        for(int i = 0;i < v.size();i++) printf("%d\n",v[i]);
    }
    return 0;
}
```

**扩展 KMP**

```
//ne[i]:x[i...m-1]与 x[0...m-1]的最长公共前缀
//ex[i]:y[i...n-1]与 x[0...m-1]的最长公共前缀
void getnext(char *s)
{
    int j = 0,len = strlen(s),k = 1;
    ne[0] = len;
    while(j+1 < len && s[j] == s[j+1])  j++;
    ne[1] = j;
    for(int i = 2;i < len;i++)
    {
        if(ne[i-k]+i < ne[k]+k) ne[i] = ne[i-k];
        else
        {
            j = max(0,ne[k]+k-i);
            while(i+j < len && s[i+j] == s[j])  j++;
            ne[i] = j;
            k = i;
        }
    }
}

void ekmp(char *x,char *y)
{
    getnext(x);
    int j = 0,lenx = strlen(x),leny = strlen(y);
    while(j < lenx && j < leny && x[j] == y[j]) j++;
    ex[0] = j;
    int k = 0;
    for(int i = 1;i < leny;i++)
    {
        if(ne[i-k]+i < ex[k]+k) ex[i] = ne[i-k];
        else
        {
            j = max(0,ex[k]+k-i);
            while(i+j < leny && j < lenx && y[i+j] == x[j]) j++;
            ex[i] = j;
            k = i;
```

```
        }
    }
}
```

**最长回文字串 Manacher**

```
//abaa
//i:    0 1 3 4 5 6 7 8 9 10
//a[i]: $ # a # b # a # a #
//p[i]: 1 1 2 1 4 1 2 3 2 1
char s[100005],a[200005];
int p[200005];

void manacher(int len)
{
    int mx = 0,id;
    for(int i = 1;i < len;i++)
    {
        if(mx > i)    p[i] = min(p[2*id-i],mx-i);
        else    p[i] = 1;
        while(a[i+p[i]] == a[i-p[i]])   p[i]++;
        if(p[i]+i > mx)
        {
            mx = p[i]+i;
            id = i;
        }
    }
}
int main()
{
    scanf("%s",s);
    a[0] = '$';
    a[1] = '#';
    int len = 2;
    for(int i = 0;s[i];i++)
    {
        a[len++] = s[i];
        a[len++] = '#';
    }
    manacher(len);
    int ans = 0;
    for(int i = 0;i < len;i++)    ans = max(ans,p[i]);
    printf("%d\n",ans-1);
    return 0;
}
```

**最小表示**

```
//最小表示：一个环选一个起点使字典序最小
//返回最小表示坐标
int minpre(char *s)
{
    int len = strlen(s);
    for(int i = 0;i < len;i++)  s[i+len] = s[i];
    int i = 0,j = 1;
    while(i < len && j < len)
    {
        int k = 0;
        while(s[i+k] == s[j+k] && k < len)  k++;
        if(k == len)    break;
        if(s[i+k] > s[j+k]) i = i+k+1;
        else    j = j+k+1;
        if(i == j)  j++;
    }
    return min(i,j);
}
```

## Trie 树

```
//添加
void add(char *s,int x)
{
    int now = 0;
    for(int i = 0;i < strlen(s);i++)
    {
        int c = s[i]-'a';
        if(!ch[now][c])
        {
            ch[now][c] = ++sz;
            cnt[sz] = 0;
        }
        now = ch[now][c];
        cnt[now]++;
    }
}

//查找数量
int getnum(char *s)
{
    int now = 0;
    for(int i = 0;i < strlen(s);i++)
    {
        int c = s[i]-'a';
        if(!ch[now][c]) return 0;
        now = ch[now][c];
    }
    return cnt[now];
}
```

## AC 自动机

```
//用 n 个模式串建立自动机
//求目标中出现了几个模式串
struct Trie
{
    int next[500000][26],fail[500005],num[500005],root,cnt;
    int newnode()
    {
        for(int i = 0;i < 26;i++)   next[cnt][i] = -1;
        num[cnt++] = 0;
        return cnt-1;
    }
    void init()
    {
        cnt = 0;
        root = newnode();
    }
    void insert(char *s)
    {
        int now = root,len = strlen(s);
        for(int i = 0;i < len;i++)
        {
            int c = s[i]-'a';
            if(next[now][c] == -1)  next[now][c] = newnode();
            now = next[now][c];
        }
        num[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 *s)
    {
        int now = root,ans = 0,len = strlen(s);
        for(int i = 0;i < len;i++)
        {
            now = next[now][s[i]-'a'];
            int t = now;
            while(t != root)
            {
                ans += num[t];
                num[t] = 0;
                t = fail[t];
            }
        }
        return ans;
    }
    void debug()
    {
        for(int i = 0;i < cnt;i++)
        {
            printf("id = %3d,fail = %3d,num = %3d,chi = [",i,fail[i],num[i]);
            for(int j = 0;j < 26;j++)   printf("%2d",next[i][j]);
            printf("]\n");
        }
    }
};
int n;
char s[1000001];
Trie ac;

int main()
{
    int T;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d",&n);
        ac.init();
        while(n--)
        {
            scanf("%s",s);
```

```
            ac.insert(s);
        }
        ac.build();
        scanf("%s",s);
        printf("%d\n",ac.query(s));
    }
    return 0;
}
```

**后缀数组 DA**

```
//O(nlogn)
/待排序数组长度 n,放在 0~n 中，最后补 0
//sa[i]:每个后缀串从小到大排第 i 小的位置
//rank[i]:i 位置的从小到大排序位置
//height[i]:sa[i]和 sa[i-1]对应后缀的最长公共前缀
//n = 8
//num[i]:   1 1 2 1 1 1 1 2 0       num[8]加 0
//sa[i]:      8 3 4 5 0 6 1 7 2       num[0~n]有效
//rank[i]:  4 6 8 1 2 3 5 7 0       num[0~n-1]有效
//height[i]:0 0 3 2 3 1 2 0 1       num[2~n]有效
int t1[N],t2[N],c[N],sa[N],rank[N],height[N];
int mm[200005],best[20][200005],rmq[200005];

bool cmp(int *r,int a,int b,int l)
{
    return r[a] == r[b] && r[a+l] == r[b+l];
}

void da(int *r,int *sa,int *rank,int *height,int n,int m)
{
    r[n] = 0;
    n++;
    int *x = t1,*y = t2;
    for(int i = 0;i < m;i++)    c[i] = 0;
    for(int i = 0;i < n;i++)    c[x[i] = r[i]]++;
    for(int i = 1;i < m;i++)    c[i] += c[i-1];
    for(int i = n-1;i >= 0;i--) sa[--c[x[i]]] = i;
    for(int j = 1;j <= n;j <<= 1)
    {
        int p = 0;
        for(int i = n-j;i < n;i++)  y[p++] = i;
        for(int i = 0;i < n;i++)
        {
            if(sa[i] >= j)  y[p++] = sa[i]-j;
        }
        for(int i = 0;i < m;i++)    c[i] = 0;
        for(int i = 0;i < n;i++)    c[x[y[i]]]++;
        for(int i = 1;i < m;i++)    c[i] += c[i-1];
        for(int i = n-1;i >= 0;i--) sa[--c[x[y[i]]]] = y[i];
        swap(x,y);
        p = 1;
        x[sa[0]] = 0;
        for(int 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;
    }
    int k = 0;
    n--;
    for(int i = 0;i <= n;i++)   rank[sa[i]] = i;
    for(int i = 0;i < n;i++)
    {
        if(k)   k--;
        int t = sa[rank[i]-1];
        while(r[i+k] == r[t+k]) k++;
```

```
        height[rank[i]] = k;
    }
}

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-1];i++)
    {
        for(int j = 1;j+(1<<i)-1 < n;j++)
        {
            int a = best[i-1][j],b = best[i-1][j+(1<<(i-1))];
            best[i][j] = rmq[a] < rmq[b]?a:b;
        }
    }
}
int askrmq(int a,int b)
{
    int t = mm[b-a+1];
    b -= (1<<t)-1;
    a = best[t][a];
    b = best[t][b];
    return rmq[a] < rmq[b]?a:b;
}

//求 a,b 位置开始的后缀的最长公共前缀
int lcp(int a,int b)
{
    a = rk[a];
    b = rk[b];
    if(a > b)   swap(a,b);
    return height[askrmq(a+1,b)];
}
int main()
{
    gets(s);
    int len = strlen(s);
    for(int i = 0;i < len;i++)  r[i] = s[i];
    da(r,sa,rk,height,len,128);
    return 0;
}
```

**后缀数组 DC3**

```
//O(n)
//所有数组开 3 倍
#define F(x) ((x)/3+((x)%3 == 1?0:tb))
#define G(x) ((x) < tb?(x)*3+1:((x)-tb)*3+2)

int wa[3*N],wb[3*N],wv[3*N],wss[3*N],r[3*N],sa[3*N],rk[3*N],height[3*N];
char s[3*N];

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);
    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)
{
    for(int i = 0;i < n;i++)    wv[i] = r[a[i]];
    for(int i = 0;i < m;i++)    wss[i] = 0;
    for(int i = 0;i < n;i++)    wss[wv[i]]++;
    for(int i = 1;i < m;i++)    wss[i] += wss[i-1];
    for(int i = n-1;i >= 0;i--) b[--wss[wv[i]]] = a[i];
}

void dc3(int *r,int *sa,int n,int m)
{
    int *rn = r+n,*san = sa+n,ta = 0,tb = (n+1)/3,tbc = 0,i,j,p;
    r[n]=r[n+1]=0;
    for(i = 0;i < n;i++)
    {
        if(i%3) 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++];
}

void da(int *r,int *sa,int *rank,int *height,int n,int m)
{
    for(int i = n;i < n*3;i++)  r[i] = 0;
    dc3(r,sa,n+1,m);
    int k = 0;
    for(int i = 0;i <= n;i++)   rank[sa[i]] = i;
    for(int i = 0;i < n;i++)
    {
        if(k)   k--;
        int t = sa[rank[i]-1];
        while(r[i+k] == r[t+k]) k++;
        height[rank[i]] = k;
    }
}
```

**后缀自动机**

1.所有的子串都能够由 root 走到。

2.所有走到终止状态的路径都是后缀。

3.每个状态 s 代表的串的长度是区间($len_{pre}$,$len_s$]。

4.对于每个状态 s,它代表的所有串在原串中出现次数和每次出现的右端点相同。

5.在后缀自动机的 Parent 树中,每个状态的 right 集合都是其父状态 right 集合的子集。

6.后缀自动机的 Parent 树是原串的反向前缀树。

7.两个串的最长公共后缀,位于这两个串对应状态在 Parent 树上的最近公共祖先状态。

**后缀自动机模版**

```
struct samnode
{
    int len,right,pre,sum,next[26];
    void clear()
    {
        len = 0;
        right = 0;
        sum = 0;
        pre = -1;
        memset(next,-1,sizeof(next));
    }
}st[2*N];
int n,sz,root,last;

void saminit()
{
    sz = 0;
    root = last = 0;
    st[root].clear();
}

void samadd(int w)
{
    int p = last,now = ++sz;
    last = now;
    st[now].clear();
    st[now].len = st[p].len+1;
    st[now].right = 1;
    while(p != -1 && st[p].next[w] == -1)
    {
        st[p].next[w] = now;
        p = st[p].pre;
    }
    if(p == -1)
    {
        st[now].pre = root;
        return;
    }
    int q = st[p].next[w];
    if(st[q].len == st[p].len+1)
    {
        st[now].pre = q;
        return;
    }
    int neww = ++sz;
    st[neww].clear();
    memcpy(st[neww].next,st[q].next,sizeof(st[q].next));
    st[neww].len = st[p].len+1;
    st[neww].pre = st[q].pre;
    st[q].pre = neww;
    st[now].pre = neww;
    while(p != -1 && st[p].next[w] == q)
    {
        st[p].next[w] = neww;
        p = st[p].pre;
    }
}
```

```
void sambuild(char *s)
{
    saminit();
    int len = strlen(s);
    for(int i = 0;i < len;i++)
        samadd(s[i]-'a');
}
```

**求最长公共子串**

```
int main()
{
    scanf("%s",s);
    sambuild(s);
    scanf("%s",s);
    int len = strlen(s),ans = 0;
    int p = root,l = 0;
    for(int i = 0;i < len;i++)
    {
        int c = s[i]-'a';
        if(st[p].next[c] != -1)
        {
            l++;
            p = st[p].next[c];
        }
        else
        {
            while(p != -1 &&st[p].next[c] == -1)    p = st[p].pre;
            if(p == -1)
            {
                l = 0;
                p = root;
            }
            else
            {
                l = st[p].len+1;
                p = st[p].next[c];
            }
        }
        ans = max(ans,l);
    }
    printf("%d\n",ans);
    return 0;
}
```

**求最小表示**

```
int main()
{
    int T;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%s",s);
        int len = strlen(s);
        for(int i = 0;i < len;i++)  s[i+len] = s[i];
        s[len+len] = 0;
        sambuild(s);
        int now = root;
        for(int i = 1;i <= len;i++)
        {
            for(int j = 0;j < 26;j++)
            {
                if(st[now].next[j] != -1)
                {
```

```
                    now = st[now].next[j];
                    break;
                }
            }
        }
        printf("%d\n",st[now].len-len+1);
    }
}
```

**求出现次数大于 k 的子串数量**

```
void sambuild(char *s)
{
    saminit();
    int len = strlen(s);
    for(int i = 0;i < len;i++)
        samadd(s[i]-'a');
}

char s[N];
int k,num[N],top[2*N];

int main()
{
    int T;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d%s",&k,s);
        sambuild(s);
        int len = strlen(s);
        memset(num,0,sizeof(num));
        for(int i = 1;i <= sz;i++)  num[st[i].len]++;
        for(int i = 1;i <= len;i++) num[i] += num[i-1];
        for(int i = sz;i >= 1;i--)  top[num[st[i].len]--] = i;
        for(int i = sz;i >= 1;i--)
        {
            int p = top[i];
            if(st[p].pre != -1) st[st[p].pre].right += st[p].right;
        }
        long long ans = 0;
        for(int i = 1;i <= sz;i++)
        {
            if(st[i].right >= k)    ans += st[i].len-st[st[i].pre].len;
        }
        printf("%lld\n",ans);
    }
    return 0;
}
```

**求第 k 小的子串(重复算一个)**

```
char s[N];
int t,k,num[N] = {0},top[2*N];

void dfs(int now,int k)
{
    if(k <= st[now].right)  return;
    k -= st[now].right;
    for(int i = 0;i < 26;i++)
    {
        int p = st[now].next[i];
        if(p == -1) continue;
        if(k <= st[p].sum)
        {
```

```
            printf("%c",i+'a');
            dfs(p,k);
            return;
        }
        k -= st[p].sum;
    }
}

int main()
{
    scanf("%s%d%d",s,&t,&k);
    sambuild(s);
    int len = strlen(s);
    for(int i = 1;i <= sz;i++)  num[st[i].len]++;
    for(int i = 1;i <= len;i++) num[i] += num[i-1];
    for(int i = sz;i >= 1;i--)  top[num[st[i].len]--] = i;
    for(int i = 1;i <= sz;i++)  st[i].right = 1;
    st[0].right = 0;
    for(int i = sz;i >= 0;i--)
    {
        int p = top[i];
        st[p].sum = st[p].right;
        for(int j = 0;j < 26;j++)
        {
            int pp = st[p].next[j];
            if(pp == -1)    continue;
            st[p].sum += st[pp].sum;
        }
    }
    if(k > st[0].sum)   printf("-1\n");
    else
    {
        dfs(0,k);
        printf("\n");
    }
    return 0;
}
```

**求第 k 小的字串(重复算多个)**

```
char s[N];
int t,k,num[N] = {0},top[2*N];

void dfs(int now,int k)
{
    if(k <= st[now].right)  return;
    k -= st[now].right;
    for(int i = 0;i < 26;i++)
    {
        int p = st[now].next[i];
        if(p == -1) continue;
        if(k <= st[p].sum)
        {
            printf("%c",i+'a');
            dfs(p,k);
            return;
        }
        k -= st[p].sum;
    }
}

int main()
{
    scanf("%s%d%d",s,&t,&k);
```

```
    sambuild(s);
    int len = strlen(s);
    for(int i = 1;i <= sz;i++)  num[st[i].len]++;
    for(int i = 1;i <= len;i++) num[i] += num[i-1];
    for(int i = sz;i >= 1;i--)  top[num[st[i].len]--] = i;
    for(int i = sz;i >= 1;i--)
    {
        int p = top[i];
        if(st[p].pre != -1) st[st[p].pre].right += st[p].right;
    }
    for(int i = sz;i >= 0;i--)
    {
        int p = top[i];
        st[p].sum = st[p].right;
        for(int j = 0;j < 26;j++)
        {
            int pp = st[p].next[j];
            if(pp == -1)    continue;
            st[p].sum += st[pp].sum;
        }
    }
    if(k > st[0].sum)   printf("-1\n");
    else
    {
        dfs(0,k);
        printf("\n");
    }
    return 0;
}
```

**字符串 hash**

```
const int HASH = 10007;
const int N = 2010;
const int SEED = 13331;
char s[N];

struct HASHMAP
{
    int head[N],next[N],size,f[N];
    unsigned long long state[N];
    void init()
    {
        size = 0;
        memset(state,-1,sizeof(state));
    }
    int insert(unsigned long long x,int id)
    {
        int h = x%HASH;
        for(int i = head[h];i != -1;i = next[i])
        {
            if(x == state[i])   return f[i];
        }
        f[size] = id;
        state[size] = x;
        next[size] = head[h];
        head[h] = size++;
        return 0;
    }
};
int main()
{
    unsigned long long t = 0;
    for(int i = 1;i <= len;i++) t = t*SEED+s[i-1];
}
```

# 图论

## 最短路

### dij 邻接矩阵

```
//O(n^2)
void dij(int beg)
{
    memset(dis,0x3f,sizeof(dis));
    memset(vis,0,sizeof(vis));
    dis[beg] = 0;
    for(int i = 1;i <= n;i++)
    {
        int k = -1,minn = INF;
        for(int j = 1;j <= n;j++)
        {
            if(!vis[j] && dis[j] < minn)
            {
                minn = dis[j];
                k = j;
            }
        }
        if(k == -1) break;
        vis[k] = 1;
        for(int j = 1;j <= n;j++)
        {
            if(!vis[j] && dis[k]+a[k][j] < dis[j])
            {
                dis[j] = dis[k]+a[k][j];
            }
        }
    }
}
```

### dij 优先队列

```
///O(mlogm)
struct xxx
{
    int to,w;
    xxx(int a,int b):to(a),w(b){};
    friend bool operator <(xxx X,xxx Y)
    {
        return X.w > Y.w;
    }
};

void dij(int beg)
{
    priority_queue<xxx> q;
    memset(dis,0x3f,sizeof(dis));
    memset(vis,0,sizeof(vis));
    dis[beg] = 0;
    q.push(xxx(beg,0));
    while(!q.empty())
    {
        int now = q.top().to;
        q.pop();
        if(vis[now])    continue;
        vis[now] = 1;
        for(int i = 0;i < v[now].size();i++)
        {
            int t = v[now][i].to,w = v[now][i].w;
            if(!vis[t] && dis[now]+w < dis[t])
            {
```

```
                dis[t] = dis[now]+w;
                q.push(xxx(t,dis[t]));
            }
        }
    }
}
```

**Floyd**

```
//O(n^3)
void floyd()
{
    for(int k = 0;k < n;k++)
    {
        for(int i = 0;i < n;i++)
        {
            for(int j = 0;j < n;j++)    a[i][j] = min(a[i][j],a[i][k]+a[k][j]);
        }
    }
}
```

**bellman_ford**

```
//O(nm)
struct xxx
{
    int u,v,w;
    xxx(int a,int b,int c):u(a),v(b),w(c){};
};
vector<xxx> v;

//存在负环返回0，否则返回1
bool bellman_ford(int beg)
{
    memset(dis,0x3f,sizeof(dis));
    dis[beg] = 0;
    for(int i = 1;i < n;i++)
    {
        for(int j = 0;j < v.size();j++)
        {
            int x = v[j].u,y = v[j].v,w = v[j].w;
            dis[y] = min(dis[y],dis[x]+w);
        }
    }
    for(int i = 0;i < v.size();i++)
    {
        int x = v[i].u,y = v[i].v,w = v[i].w;
        if(dis[y] > dis[x]+w)   return 0;
    }
    return 1;
}
```

**Spfa**

```
//O(km)
struct xxx
{
    int to,w;
    xxx(int a,int b):to(a),w(b){};
};
vector<xxx> v[205];

//存在负环返回0，否则返回1
bool spfa(int beg)
{
    queue<int> q;
    memset(c,0,sizeof(c));
    memset(vis,0,sizeof(vis));
    memset(dis,0x3f,sizeof(dis));
```

```
    dis[beg] = 0;
    q.push(beg);
    vis[beg] = 1;
    c[beg] = 1;
    while(!q.empty())
    {
        int now = q.front();
        q.pop();
        vis[now] = 0;
        for(int i = 0;i < v[now].size();i++)
        {
            int t = v[now][i].to,w = v[now][i].w;
            if(dis[now]+w < dis[t])
            {
                dis[t] = dis[now]+w;
                vis[t] = 1;
                q.push(t);
                if(++c[t] > n)   return 0;
            }
        }
    }
    return 1;
}
```

**分层图最短路**

```
struct xxx
{
    int to,w;
    xxx(int a,int b):to(a),w(b){};
    friend bool operator<(xxx x,xxx y)
    {
        return x.w > y.w;
    }
};
vector<xxx> v[11005];
int n,m,k,a[1005][1005],dis[11005],vis[11005];

void dij(int beg)
{
    memset(dis,0x3f,sizeof(dis));
    memset(vis,0,sizeof(vis));
    priority_queue<xxx> q;
    dis[beg] = 0;
    q.push(xxx(beg,0));
    while(!q.empty())
    {
        int now = q.top().to;
        q.pop();
        if(vis[now])    continue;
        vis[now] = 1;
        for(int i = 0;i < v[now].size();i++)
        {
            int t = v[now][i].to,w = v[now][i].w;
            if(w+dis[now] < dis[t])
            {
                dis[t] = w+dis[now];
                q.push(xxx(t,dis[t]));
            }
        }

    }
}

int main()
```

```
{
    while(~scanf("%d%d%d",&n,&m,&k))
    {
        for(int i = 1;i <= 11000;i++)    v[i].clear();
        memset(a,0x3f,sizeof(a));
        for(int i = 1;i <= m;i++)
        {
            int u,v,w;
            scanf("%d%d%d",&u,&v,&w);
            a[u][v] = min(a[u][v],w);
            a[v][u] = min(a[v][u],w);
        }
        for(int i = 1;i <= n;i++)
        {
            for(int j = 1;j <= n;j++)
            {
                if(i == j || a[i][j] == INF)    continue;
                for(int l = 0;l <= k;l++)
                {
                    v[l*n+i].push_back(xxx(l*n+j,a[i][j]));
                    if(l != k)
                    {
                        v[l*n+i].push_back(xxx((l+1)*n+j,0));
                    }
                }
            }
        }
        dij(1);
        printf("%d\n",dis[(k+1)*n]);
    }
    return 0;
}
```

**最小生成树**

**prim 邻接矩阵**

```
//不连通返回-1，否则返回最小花费
int prim()
{
    int ans = 0;
    memset(vis,0,sizeof(vis));
    vis[1] = 1;
    for(int i = 2;i <= n;i++)   cost[i] = a[1][i];
    for(int i = 2;i <= n;i++)
    {
        int minn = INF;
        int k = -1;
        for(int j = 1;j <= n;j++)
        {
            if(!vis[j] && minn > cost[j])
            {
                minn = cost[j];
                k = j;
            }
        }
        if(minn == INF) return -1;
        ans += minn;
        vis[k] = 1;
        for(int j = 1;j <= n;j++)
        {
            if(!vis[j] && cost[j] > a[k][j])    cost[j] = a[k][j];
        }
    }
    return ans;
}
```

**prim 优先队列**

```
struct xxx
{
    int to,w;
    xxx(int a,int b):to(a),w(b){}
    friend bool operator<(xxx a,xxx b)
    {
        return a.w > b.w;
    }
};
//非连通返回-1，否则返回最小花费
int prim()
{
    memset(vis,0,sizeof(vis));
    for(int i = 0;i < n;i++)    dis[i] = INT_MAX;
    int ans = 0;
    priority_queue<xxx> q;
    q.push(xxx(1,0));
    while(!q.empty())
    {
        while(!q.empty() && vis[q.top().to])    q.pop();
        if(q.empty())   break;
        ans += q.top().w;
        int now = q.top().to;
        q.pop();
        vis[now] = 1;
        for(int i = 0;i < v[now].size();i++)
        {
            int t = v[now][i].to,w = v[now][i].w;
            if(vis[t])  continue;
            if(dis[t] <= w) continue;
            dis[t] = w;
            q.push(xxx(t,w));
        }
    }
    return ans;
}
```

**Kruskal**

```
struct xxx
{
    int from,to,w;
    xxx(int a,int b,int c):from(a),to(b),w(c){};
    friend bool operator <(xxx a,xxx b)
    {
        return a.w < b.w;
    }
};
vector<xxx> v;

int findd(int x)
{
    return pre[x] == x?x:findd(pre[x]);
}

//不连通返回-1，否则返回最小花费
bool kruskal()
{
    for(int i = 1;i <= n;i++)   pre[i] = i;
    int ans = 0,cnt = 0;
    sort(v.begin(),v.end());
    for(int i = 0;i < v.size();i++)
    {
        int x = findd(v[i].from),y = findd(v[i].to);
```

```
        if(x != y)
        {
            ans += v[i].w;
            pre[x] = y;
            cnt++;
        }
    }
    if(cnt < n-1)   return -1;
    return ans;
}
```

## 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。

3.G 的 Kirchhoff 矩阵(也称为拉普拉斯算子)C[G]=D[G]-A[G]。

则 G 的生成树个数等于 C[G]任何一个 n-1 阶主子式的行列式的绝对值。

## 矩阵树

```
int n,m,k,a[55][55];
long long g[55][55];

long long calc()
{
    long long ans = 1;
    for(int i = 1;i < n;i++)
    {
        for(int j = i+1;j < n;j++)
        {
            while(g[j][i] != 0)
            {
                long long t = g[i][i]/g[j][i];
                for(int k = i;k < n;k++)    g[i][k] -= t*g[j][k];
                for(int k = i;k < n;k++)    swap(g[i][k],g[j][k]);
                ans = -ans;
            }
        }
        if(g[i][i] == 0)    return 0;
        ans = ans*g[i][i];
    }
    return abs(ans);
}

int main()
{
    ios::sync_with_stdio(false);
    while(cin >> n >> m >> k)
    {
        memset(a,0,sizeof(a));
        memset(g,0,sizeof(g));
        while(m--)
        {
            int x,y;
            cin >> x >> y;
            a[x][y] = a[y][x] = 1;
        }
        for(int i = 1;i <= n;i++)
        {
            int t = 0;
            for(int j = 1;j <= n;j++)
            {
                if(i != j && !a[i][j])
```

```
                {
                    t++;
                    g[i][j] = -1;
                }
            }
            g[i][i] = t;
        }
        cout << calc() << endl;
    }
    return 0;
}
```

**次小生成树**

```
int n,vis[505],used[505],pre[505],cost[505],a[505][505],maxd[505][505];
//不连通返回-1，否则返回次小花费
int prim()
{
    int ans = 0;
    memset(vis,0,sizeof(vis));
    memset(used,0,sizeof(used));
    memset(maxd,0,sizeof(maxd));
    vis[1] = 1;
    for(int i = 2;i <= n;i++)   cost[i] = a[1][i];
    for(int i = 2;i <= n;i++)
    {
        int minn = INF;
        int k = -1;
        for(int j = 1;j <= n;j++)
        {
            if(!vis[j] && minn > cost[j])
            {
                minn = cost[j];
                k = j;
            }
        }
        if(minn == INF) return -1;
        ans += minn;
        vis[k] = 1;
        for(int j = 1;j <= n;j++)
        {
            if(vis[j])  maxd[j][k] = maxd[k][j] = max(maxd[j][pre[k]],cost[k]);
            if(!vis[j] && cost[j] > a[k][j])
            {
                cost[j] = a[k][j];
                pre[j] = k;
            }
        }
    }
    return ans;
}
```

**Tarjan 缩点**

```
//O(n+m)
//缩点，belong 保存每个点属于的强联通集合编号
int n,m,vis[10005],dfn[10005],low[10005],belong[10005],cnt,ans;
vector<int> v[10005];

void tarjan(int now)
{
    stack<int> s;
    s.push(now);
    vis[now] = 1;
    dfn[now] = low[now] = ++cnt;
    for(int i = 0;i < v[now].size();i++)
    {
```

```
        int t = v[now][i];
        if(dfn[t] == 0)
        {
            tarjan(t);
            low[now] = min(low[now],low[t]);
        }
        else if(vis[t])    low[now] = min(low[now],dfn[t]);
    }
    if(dfn[now] == low[now])
    {
        ans++;
        while(!s.empty())
        {
            int t = s.top();
            s.pop();
            vis[t] = 0;
            belong[t] = ans;
            if(t == now)    break;
        }
    }
}

int main()
{
    for(int i = 1;i <= n;i++)
    {
        if(dfn[i] == 0) tarjan(i);
    }
}
```

**Kosaraju 求强联通分量**

```
int n,m,vis1[10005],vis2[10005],ans;
vector<int> v1[10005],v2[10005];
stack<int> s;

void dfs1(int now)
{
    vis1[now] = 1;
    for(int i = 0;i < v1[now].size();i++)
    {
        int t = v1[now][i];
        if(vis1[t])  continue;
        dfs1(t);
    }
    s.push(now);
}

void dfs2(int now)
{
    vis2[now] = 1;
    for(int i = 0;i < v2[now].size();i++)
    {
        int t = v2[now][i];
        if(vis2[t]) continue;
        dfs2(t);
    }
}

int main()
{
    ios::sync_with_stdio(false);
    while(scanf("%d%d",&n,&m) &&(n+m))
    {
        for(int i = 1;i <= n;i++)
```

```
        {
            v1[i].clear();
            v2[i].clear();
        }
        memset(vis1,0,sizeof(vis1));
        memset(vis2,0,sizeof(vis2));
        while(!s.empty())    s.pop();
        ans = 0;
        while(m--)
        {
            int x,y;
            scanf("%d%d",&x,&y);
            v1[x].push_back(y);
            v2[y].push_back(x);
        }
        for(int i = 1;i <= n;i++)
        {
            if(!vis1[i]) dfs1(i);
        }
        while(!s.empty())
        {
            if(!vis2[s.top()])
            {
                ans++;
                dfs2(s.top());
            }
            s.pop();
        }
        if(ans == 1)  printf("Yes\n");
        else    printf("No\n");
    }
    return 0;
}
```

**无向图中一些基本概念**

割点集合：删除这个顶点集合,以及这个集合中所有顶点关联的边以后,原图变成多个连通块。
点连通度：最小割点集合中的顶点数。
割边集合：删除这个边集合以后,原图变成多个连通块。
边连通度：最小割边集合中的边数。

点双连通（双连通或重连通）：无向连通图的点连通度大于 1。
割点（关节点）：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。
边双连通（双连通或重连通）：无向连通图的边连通度大于 1。
桥（关节边 ）：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。

边双连通分支：在连通分支中去掉一条边，连通分支仍连通。
点双连通分支（块）：在连通分支中去掉一个点，连通分支仍连通。

**求割点和桥**

```
struct bridge
{
    int u,int v;
    bridge(int a,int b):u(a),v(b){}
};
vector<bridge> ansb;
vector<int> absc;

//ansb 保存桥，ansc 保存割点
void tarjan(int now,int pre)
{
    low[now] = dfn[now] = ++cnt;
    int son = 0;
    for(int i = 0;i < v[now].size();i++)
    {
        int t = v[now][i];
```

```
        if(t == pre)    continue;
        if(!dfn[t])
        {
            son++;
            tarjan(t,now);
            low[now] = max(low[now],low[t]);
            //桥，(u,v)为树枝，low[v] > dfn[u]
            if(low[t] > dfn[now])
            {
                ansb.push_back(bridge(now,t));
                ansb.push_back(bridge(t,now));
            }
            //割点，(u,v)父子边，u 不为树根，low[v] >= dfn[u]
            if(now != root && low[t] >= dfn[now]) ansc.push_back(t);
        }
        else    low[now] = max(low[now],low[t]);
    }
    //割点，u 为根，分支数大于 1
    if(now == root && son > 1)  ansc.push_back(now);
}
```

**求边双连通分支**

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双连通图的话，把双连通子图缩成一个点，形成一棵树。

再加上（leaf+1）/2 条边。

**求点双连通分支**

```
//cutcnt 表示个数
//block 储存每个点连通分支
stack<int> s;
vector<int> block[10005];
void tarjan(int now,int pre)
{
    low[now] = dfn[now] = ++cnt;
    s.push(now);
    for(int i = 0;i < v[now].size();i++)
    {
        int t = v[now][i];
        if(t == pre)    continue;
        if(!dfn[t])
        {
            tarjan(t,now);
            low[now] = max(low[now],low[t]);
            if(low[t] >= dfn[now])
            {
                cutcnt++;
                block[cutcnt].clear();
                while(!s.empty())
                {
                    int tt = s.top();
                    s.pop();
                    block[cutcnt].push_back(tt);
                    if(tt = t)  break;
                }
                block[cutcnt].push_back(now);
            }
        }
        else    low[now] = max(low[now],low[t]);
    }
}
```

## 最小树形图

```
struct xxx
{
    int u,v,w;
}e[10005];
int n,m,a[55],sum[55],cnt,in[800],pre[800],vis[800],id[800];

void addedge(int u,int v,int w)
{
    e[cnt].u = u;
    e[cnt].v = v;
    e[cnt++].w = w;
}

//已处理重边，不存在最小树形图返回-1，否则返回最小值
int zhuliu(int root, int n, int m)
{
    int ans = 0;
    while(1)
    {
        //找最小入边
        for(int i = 0;i < n;i++)    in[i] = INF;
        for(int i = 0;  < m;i++)
        {
            int u = e[i].u;
            int v = e[i].v;
            if(e[i].w < in[v] && u != v)
            {
                pre[v] = u;
                in[v] = e[i].w;
            }     //去除自环和重边
        }
        for(int i = 0;i < n;i++)
        {
            if(i == root)   continue;
            if(in[i] == INF)    return -1;                                  //除了跟以外
有点没有入边,则根无法到达它
        }
        int cnt = 0;                                //找环
        memset(id,-1,sizeof(id));
        memset(vis,-1,sizeof(vis));
        in[root] = 0;
        for(int i = 0;i < n;i++)
        {
            ans += in[i];
            int v = i;
            while(vis[v] != i && id[v] == -1 && v != root)  //每个点寻找其前序点，要么
最终寻找至根部，要么找到一个环
            {
                vis[v] = i;
                v = pre[v];
            }
            if(v != root && id[v] == -1)//缩点
            {
                for(int u = pre[v];u != v;u = pre[u])   id[u] = cnt;
                id[v] = cnt++;
            }
        }
        if(cnt == 0)    break; //无环   则break
        for(int i = 0;i < n;i++)
        {
            if(id[i] == -1) id[i] = cnt++;
        }
```

```
        for(int i = 0;i < m;)
        {
            int u = e[i].u;
            int v = e[i].v;
            e[i].u = id[u];
            e[i].v = id[v];
            if(id[u] != id[v])  e[i++].w -= in[v];
            else    e[i]=e[--m];
        }
        n = cnt;
        root = id[root];
    }
    return ans;
}
```

**二分图**

最大匹配 == 最小点覆盖

最小路径覆盖 == 最大独立集 == 顶点数-最大匹配

最大团 == 补图最大独立集

**匈牙利**

```
int n,m,l;
int linker[505];
bool used[505];
vector<int> v[505];

bool dfs(int u)
{
    for(int i = 0;i < v[u].size();i++)
    {
        int t = v[u][i];
        if(!used[t])
        {
            used[t] = 1;
            if(linker[t] == -1 || dfs(linker[t]))
            {
                linker[t] = u;
                return 1;
            }
        }
    }
    return 0;
}

int MaxMatch()
{
    int ans = 0;
    memset(linker,-1,sizeof(linker));
    for(int i = 1;i <= n;i++)
    {
        memset(used,0,sizeof(used));
        if(dfs(i))    ans++;
    }
    return ans;
}
```

**Hopcroft-Carp**

```
//O(V^0.5 E)
//适用于数据较大的二分匹配
int g[MAXN][MAXN],Mx[MAXN],My[MAXN],Nx,Ny;
int dx[MAXN],dy[MAXN],dis;
bool vst[MAXN];
bool searchP()
```

```
{
    queue<int>Q;
    dis=INF;
    memset(dx,-1,sizeof(dx));
    memset(dy,-1,sizeof(dy));
    for(int i=0;i<Nx;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;
        for(int v=0;v<Ny;v++)
            if(g[u][v]&&dy[v]==-1)
            {
                dy[v]=dx[u]+1;
                if(My[v]==-1)  dis=dy[v];
                else
                {
                    dx[My[v]]=dy[v]+1;
                    Q.push(My[v]);
                }
            }
    }
    return dis!=INF;
}
bool DFS(int u)
{
    for(int v=0;v<Ny;v++)
       if(!vst[v]&&g[u][v]&&dy[v]==dx[u]+1)
       {
           vst[v]=1;
           if(My[v]!=-1&&dy[v]==dis) continue;
           if(My[v]==-1||DFS(My[v]))
           {
               My[v]=u;
               Mx[u]=v;
               return 1;
           }
       }
    return 0;
}
int MaxMatch()
{
    int res=0;
    memset(Mx,-1,sizeof(Mx));
    memset(My,-1,sizeof(My));
    while(searchP())
    {
        memset(vst,0,sizeof(vst));
        for(int i=0;i<Nx;i++)
          if(Mx[i]==-1&&DFS(i))  res++;
    }
    return res;
}
```

**多重匹配匈牙利**

```
bool dfs(int u)
{
    for(int i = 0;i < v[u].size();i++)
```

```
    {
        int t = v[u][i];
        if(used[t]) continue;
        used[t] = 1;
        if(linker[t][0] < capacity[t])
        {
            linker[t][++linker[t][0]] = u;
            return 1;
        }
        for(int j = 1;j <= capacity[t];j++)
        {
            if(dfs(linker[t][j]))
            {
                linker[t][j] = u;
                return 1;
            }
        }
    }
        return 0;
}

int MaxMatch()
{
    int ans = 0;
    memset(linker,0,sizeof(linker));
    for(int i = 1;i <= n;i++)
    {
        memset(used,0,sizeof(used));
        if(dfs(i))    ans++;
    }
    return ans;
}
```

**二分图最大权 KM**

```
int n,m,g[305][305],linker[305],lx[305],ly[305],slack[305],visx[305],visy[305];

bool dfs(int u)
{
    visx[u] = 1;
    for(int i = 1;i <= m;i++)
    {
        if(visy[i]) continue;
        int t = lx[u]+ly[i]-g[u][i];
        if(t == 0)
        {
            visy[i] = 1;
            if(!linker[i]|| dfs(linker[i]))
            {
                linker[i] = u;
                return 1;
            }
        }
        else    slack[i] = min(slack[i],t);
    }
    return 0;
}

int KM()
{
    memset(linker,0,sizeof(linker));
    memset(ly,0,sizeof(ly));
    for(int i = 1;i <= n;i++)
    {
        lx[i] = -INF;
```

```
        for(int j = 1;j <= m;j++)
        {
            if(g[i][j] > lx[i])  lx[i] = g[i][j];
        }
    }
    for(int i = 1;i <= n;i++)
    {
        memset(slack,0x3f,sizeof(slack));
        while(1)
        {
            memset(visx,0,sizeof(visx));
            memset(visy,0,sizeof(visy));
            if(dfs(i))   break;
            int d = INF;
            for(int j = 1;j <= m;j++)
            {
                if(!visy[j]) d = min(d,slack[j]);
            }
            for(int j = 1;j <= n;j++)
            {
                if(visx[j])  lx[j] -= d;
            }
            for(int j = 1;j <= m;j++)
            {
                if(visy[j])  ly[j] += d;
                else slack[j] -= d;
            }
        }
    }
    int ans = 0;
    for(int i = 1;i <= m;i++)
    {
        if(linker[i])   ans += g[linker[i]][i];
    }
    return ans;
}
```

## 网络流

最大流 == 最小割

### EdmondsKarp

```
//s 为源点，t 为汇点
//O(nm^2)
int n,m,s,t,a[20][20],pre[20],vis[20];

bool bfs()
{
    memset(pre,0,sizeof(pre));
    memset(vis,0,sizeof(vis));
    vis[s] = 1;
    queue<int> q;
    q.push(s);
    while(!q.empty())
    {
        int now = q.front();
        q.pop();
        if(now == t)    return 1;
        for(int i = 1;i <= n;i++)
        {
            if(vis[i])  continue;
            if(a[now][i])
            {
                q.push(i);
```

```
                pre[i] = now;
                vis[i] = 1;
            }
        }
    }
    return 0;
}

int EdmondsKarp()
{
   int ans = 0;
   while(bfs())
   {
       int minn = 1e9;
       for(int i = t;i != s;i = pre[i]) minn = min(minn,a[pre[i]][i]);
       for(int i = t;i != s;i = pre[i])
       {
           a[pre[i]][i] -= minn;
           a[i][pre[i]] += minn;
       }
       ans += minn;
   }
   return ans;
}
```

**Dinic**

```
struct edge
{
    int u,v,c,next;
}e[20*N];
int n,m,sp,tp,cnt,head[N],d[N];
//源点汇点全局定义

void init()
{
    cnt = 0;
    memset(head,-1,sizeof(head));
}

void addedge(int u,int v,int c)
{
    e[cnt].u = u;e[cnt].v = v;e[cnt].c = c;
    e[cnt].next = head[u];head[u] = cnt++;
    e[cnt].u = v;e[cnt].v = u;e[cnt].c = 0;
    e[cnt].next = head[v];head[v] = cnt++;
}

int bfs()
{
    queue<int> q;
    memset(d,-1,sizeof(d));
    d[sp]=0;
    q.push(sp);
    while(!q.empty())
    {
        int cur = q.front();
        q.pop();
        for(int i = head[cur];i != -1;i = e[i].next)
        {
            int u = e[i].v;
            if(d[u] == -1 && e[i].c > 0)
            {
                d[u] = d[cur]+1;
                q.push(u);
```

```
            }
        }
    }
    return d[tp] != -1;
}

int dfs(int a,int b)
{
    int r = 0;
    if(a == tp) return b;
    for(int i = head[a];i != -1 && r < b;i = e[i].next)
    {
        int u = e[i].v;
        if(e[i].c > 0 && d[u] == d[a]+1)
        {
            int x = min(e[i].c,b-r);
            x = dfs(u,x);
            r += x;
            e[i].c -= x;
            e[i^1].c += x;
        }
    }
    if(!r)  d[a] =- 2;
    return r;
}

int dinic()
{
    int ans = 0,t;
    while(bfs())
    {
        while(t = dfs(sp,INF))
        ans += t;
    }
    return ans;
}
```

**Isap**

```
struct Edge
{
    int to,next,cap,flow;
} edge[M];
int n,m,cnt,head[N],gap[N],d[N],cur[N],que[N],p[N];

void init()
{
    cnt = 0;
    memset(head,-1,sizeof(head));
}

void addedge(int u,int v,int c)
{
    edge[cnt].to = v;
    edge[cnt].cap = c;
    edge[cnt].flow = 0;
    edge[cnt].next = head[u];
    head[u] = cnt++;
    edge[cnt].to = u;
    edge[cnt].cap = 0;
    edge[cnt].flow = 0;
    edge[cnt].next = head[v];
    head[v] = cnt++;
}
```

```
void bfs(int source,int sink)
{
    memset(d,-1,sizeof(d));
    memset(gap,0,sizeof(gap));
    gap[0] = 1;
    int front = 0,rear = 0;
    d[sink] = 0;
    que[rear++] = sink;
    while(front != rear)
    {
        int u = que[front++];
        for(int i = head[u];i != -1;i = edge[i].next)
        {
            int v = edge[i].to;
            if(d[v] != -1) continue;
            que[rear++] = v;
            d[v] = d[u]+1;
            gap[d[v]]++;
        }
    }
}

int isap(int source,int sink,int N)         //源点，汇点，总点数
{
    bfs(source,sink);
    memcpy(cur,head,sizeof(head));
    int top = 0,x = source,flow = 0;
    while(d[source] < N)
    {
        if(x == sink)
        {
            int Min = INF,inser;
            for(int i = 0;i < top;i++)
            {
                if(Min > edge[p[i]].cap-edge[p[i]].flow)
                {
                    Min = edge[p[i]].cap-edge[p[i]].flow;
                    inser = i;
                }
            }
            for(int i = 0;i < top;i++)
            {
                edge[p[i]].flow += Min;
                edge[p[i]^1].flow -= Min;
            }
            flow += Min;
            top = inser;
            x = edge[p[top]^1].to;
            continue;
        }
        int ok = 0;
        for(int i = cur[x];i != -1;i = edge[i].next)
        {
            int v = edge[i].to;
            if(edge[i].cap > edge[i].flow && d[v]+1 == d[x])
            {
                ok = 1;
                cur[x] = i;
                p[top++] = i;
                x = edge[i].to;
                break;
            }
        }
        if(!ok)
```

```
        {
            int Min = N;
            for(int i = head[x];i != -1;i = edge[i].next)
            {
                if(edge[i].cap > edge[i].flow && d[edge[i].to] < Min)
                {
                    Min = d[edge[i].to];
                    cur[x] = i;
                }
            }
            if(--gap[d[x]] == 0) break;
            gap[d[x] = Min+1]++;
            if(x != source) x = edge[p[--top]^1].to;
        }
    }
    return flow;
}
```

**最小费用最大流**

```
int n,m,s,t,pre[1005],vis[1005],dis[1005],head[1005],cnt = 0;
struct ee
{
    int u,v,cap,cost,next;
}e[2005];

void add(int u,int v,int cap,int cost)
{
    e[cnt].u = u;
    e[cnt].v = v;
    e[cnt].cap = cap;
    e[cnt].cost = cost;
    e[cnt].next = head[u];
    head[u] = cnt++;
    e[cnt].u = v;
    e[cnt].v = u;
    e[cnt].cap = 0;
    e[cnt].cost = -cost;
    e[cnt].next = head[v];
    head[v] = cnt++;
}

bool spfa()
{
    memset(vis,0,sizeof(vis));
    memset(pre,-1,sizeof(pre));
    for(int i = s;i <= t;i++)   dis[i] = 1e9;
    vis[s] = 1;
    dis[s] = 0;
    queue<int> q;
    q.push(s);
    while(!q.empty())
    {
        int u = q.front();
        q.pop();
        vis[u] = 0;
        for(int i = head[u];i != -1;i = e[i].next)
        {
            int v = e[i].v;
            if(e[i].cap && dis[v] > dis[u]+e[i].cost)
            {
                dis[v] = dis[u]+e[i].cost;
                pre[v] = i;
                if(vis[v])  continue;
                q.push(v);
```

```
                vis[v] = 1;
            }
        }
    }
    if(dis[t] == 1e9)   return 0;
    return 1;
}
int mcmf()
{
    int flow = 0,ans = 0;
    while(spfa())
    {
        int minn = 1e9;
        for(int i = pre[t];i != -1;i = pre[e[i].u]) minn = min(e[i].cap,minn);
        for(int i = pre[t];i != -1;i = pre[e[i].u])
        {
            e[i].cap -= minn;
            e[i^1].cap += minn;
        }
        flow += minn;
        ans += dis[t]*minn;
    }
    return ans;
}
```

**2-sat**

x == 1　　: x' -> x　(x 必取)

x and y == 1　: x' -> x ,y' -> y　(两个数必须全为 1)

x and y == 0　: y -> x' ,x -> y'　(两个数至少有一个为 0)

x or y == 1　: x' -> y ,y' -> x　(两个数至少有一个为 1)

x or y == 0　: x -> x' ,y -> y'　(两个数全为 0)

x xor y == 1　: x -> y' ,y -> x' ,y' -> x ,x '-> y　（两个数不同）

x xor y == 0　: x -> y ,x' -> y' ,y -> x ,y' -> x'　（两个数相同）

**染色法**

```
//可得到最小字典序的解
int n,m,vis[16005];
vector<int> v[16005];
stack<int> s;

bool dfs(int u)
{
    if(vis[u^1])    return 0;
    if(vis[u])  return 1;
    vis[u] = 1;
    s.push(u);
    for(int i = 0;i < v[u].size();i++)
    {
        int t = v[u][i];
        if(!dfs(t)) return 0;
    }
    return 1;
}

bool twosat(int n)
{
    memset(vis,0,sizeof(vis));
    for(int i = 0;i < n;i += 2)
    {
        if(vis[i] || vis[i^1])  continue;
```

```
        while(!s.empty())    s.pop();
        if(!dfs(i))
        {
            while(!s.empty())
            {
                vis[s.top()] = 0;
                s.pop();
            }
            if(!dfs(i^1))    return 0;
        }
    }
    return 1;
}

int main()
{
    ios::sync_with_stdio(false);
    while(cin >> n >> m)
    {
        for(int i = 0;i < 2*n;i++)  v[i].clear();
        memset(vis,0,sizeof(vis));
        while(m--)
        {
            int a,b;
            cin >> a >> b;
            a--;
            b--;
            v[a].push_back(b^1);
            v[b].push_back(a^1);
        }
        if(twosat(2*n))
        {
            for(int i = 0;i < 2*n;i++)
            {
                if(vis[i])  cout << i+1 << endl;
            }
        }
        else    cout << "NIE" << endl;
    }
    return 0;
}
```

**tarjan 缩点+拓扑排序**

```
int n,m,vis[2*N],dfn[2*N],low[2*N],belong[2*N],pos[2*N],deg[2*N],ok[2*N],cnt,color;
vector<int> v[2*N],v2[2*N],ans;
stack<int> st;

//求任意解
void top(int n)
{
    ans.clear();
    memset(ok,0,sizeof(ok));
    queue<int> q;
    for(int i = 1;i <= color;i++)
    {
        if(deg[i] == 0) q.push(i);
    }
    while(!q.empty())
    {
        int now = q.front();
        q.pop();
        if(ok[now] == 0)
        {
            ok[now] = 1;
```

```
            ok[pos[now]] = 2;
        }
        for(int i = 0;i < v2[now].size();i++)
        {
            int t = v2[now][i];
            if(--deg[t] == 0)   q.push(t);
        }
    }
    for(int i = 0;i < n;i++)
    {
        if(ok[belong[i]] == 1)  ans.push_back(i);
    }
}

void tarjan(int now)
{
    st.push(now);
    vis[now] = 1;
    dfn[now] = low[now] = ++cnt;
    for(int i = 0;i < v[now].size();i++)
    {
        int t = v[now][i];
        if(dfn[t] == 0)
        {
            tarjan(t);
            low[now] = min(low[now],low[t]);
        }
        else if(vis[t])    low[now] = min(low[now],dfn[t]);
    }
    if(dfn[now] == low[now])
    {
        color++;
        while(!st.empty())
        {
            int t = st.top();
            st.pop();
            vis[t] = 0;
            belong[t] = color;
            if(t == now)    break;
        }
    }
}

bool solve(int n)
{
    for(int i = 0;i < n;i++)
    {
        if(dfn[i] == 0) tarjan(i);
    }
    for(int i = 0;i < n;i += 2)
    {
        if(belong[i] == belong[i^1])    return 0;
        pos[belong[i]] = belong[i^1];
        pos[belong[i^1]] = belong[i];
    }
    for(int i = 0;i < n;i++)
    {
        for(int j = 0;j < v[i].size();j++)
        {
            int t = v[i][j];
            if(belong[i] != belong[t])
            {
                deg[belong[i]]++;
                v2[belong[t]].push_back(belong[i]);
```

```
            }
        }
    }
    top(n);
    return 1;
}
```

**曼哈顿最小生成树**

```
struct point
{
    int x,y,id;
    friend bool operator <(point a,point b)
    {
        if(a.x != b.x)  return a.x < b.x;
        return a.y < b.y;
    }
}p[10005];
struct Bit
{
    int minn,pos;
    void init()
    {
        minn = INF;
        pos = -1;
    }
}bit[10005];
struct edge
{
    int u,v,d;
    friend bool operator<(edge a,edge b)
    {
        return a.d < b.d;
    }
}e[40005];
int n,k,cnt,pre[10005],a[10005],b[10005];

int findd(int x)
{
    return pre[x] == x?x:findd(pre[x]);
}

void addedge(int u,int v,int d)
{
    e[++cnt].u = u;
    e[cnt].v = v;
    e[cnt].d = d;
}

int lowbit(int x)
{
    return x&-x;
}

void update(int i,int x,int pos)
{
    while(i > 0)
    {
        if(x < bit[i].minn)
        {
            bit[i].minn = x;
            bit[i].pos = pos;
        }
        i -= lowbit(i);
    }
```

```
}

int ask(int i,int m)
{
    int minn = INF,pos = -1;
    while(i <= m)
    {
        if(bit[i].minn < minn)
        {
            minn = bit[i].minn;
            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()
{
    cnt = 0;
    for(int dir = 0;dir < 4;dir++)
    {
        if(dir == 1 || dir == 3)
        {
            for(int i = 1;i <= n;i++)    swap(p[i].x,p[i].y);
        }
        else if(dir == 2)
        {
            for(int i = 1;i <= n;i++)    p[i].x = -p[i].x;
        }
        sort(p+1,p+1+n);
        for(int i = 1;i <= n;i++)    a[i] = b[i] = p[i].y-p[i].x;
        sort(b+1,b+1+n);
        int t = unique(b+1,b+n+1)-b-1;
        for(int i = 1;i <= t;i++)    bit[i].init();
        for(int i = n;i >= 1;i--)
        {
            int pos = lower_bound(b+1,b+1+t,a[i])-b+1;
            int ans = ask(pos,t);
            if(ans != -1)    addedge(p[i].id,p[ans].id,dist(p[i],p[ans]));
            update(pos,p[i].x+p[i].y,i);
        }
    }
}

//返回第k小的边长
int solve(int k)
{
    manhattan();
    for(int i = 1;i <= n;i++)    pre[i] = i;
    sort(e+1,e+1+cnt);
    for(int i = 1;i <= cnt;i++)
    {
        int u = e[i].u,v = e[i].v;
        int x = findd(u),y = findd(v);
        if(x != y)
        {
            pre[x] = y;
            if(--k == 0)    return e[i].d;
```

```
        }
    }
}
```

**一般图匹配带花树**

```
//g[i][j]存放关系图：i,j 是否有边,match[i]存放 i 所匹配的点
//最终匹配方案为 match
//复杂度 O(n^3)
//点是从 1 到 n 的
bool g[MAXN][MAXN],inque[MAXN],inblossom[MAXN],inpath[MAXN];
int match[MAXN],pre[MAXN],base[MAXN];
queue<int> q;
//找公共祖先
int findancestor(int u,int v)
{
    memset(inpath,false,sizeof(inpath));
    while(1)
    {
        u = base[u];
        inpath[u] = 1;
        if(match[u] == -1)  break;
        u = pre[match[u]];
    }
    while(1)
    {
        v = base[v];
        if(inpath[v])   return v;
        v = pre[match[v]];
    }
}

//压缩花
void reset(int u,int anc)
{
    while(u != anc)
    {
        int v = match[u];
        inblossom[base[u]] = 1;
        inblossom[base[v]] = 1;
        v = pre[v];
        if(base[v] != anc)  pre[v] = match[u];
        u = v;
    }
}

void contract(int u,int v,int n)
{
    int anc = findancestor(u,v);
    memset(inblossom,0,sizeof(inblossom));
    reset(u,anc);
    reset(v,anc);
    if(base[u] != anc)  pre[u] = v;
    if(base[v] != anc)  pre[v] = u;
    for(int i = 1;i <= n;i++)
    {
        if(inblossom[base[i]])
        {
            base[i] = anc;
            if(!inque[i])
            {
                q.push_back(i);
                inque[i] = 1;
            }
        }
```

```
    }

}

bool bfs(int S,int n)
{
    for(int i = 0;i <= n;i++)    pre[i]=-1,inque[i]=0,base[i]=i;
    q.clear();
    q.push_back(S);
    inque[S] = 1;
    while(!q.empty())
    {
        int u = q.front();
        q.pop();
        for(int v = 1;v <= n;v++)
        {
            if(g[u][v] && base[v] != base[u] && match[u] != v)
            {
                if(v == S || match[v] !=-1 && pre[match[v]] != -1)  contract(u,v,n);
                else if(pre[v] == -1)
                {
                    pre[v] = u;
                    if(match[v] != -1)
                    {
                        q.push_back(match[v]);
                        inque[match[v]] = 1;
                    }
                    else
                    {
                        u = v;
                        while(u! = -1)
                        {
                            v = pre[u];
                            int w = match[v];
                            match[u] = v;
                            match[v] = u;
                            u = w;
                        }
                        return 1;
                    }
                }
            }
        }
    }
    return 0;
}

int solve(int n)
{
    memset(match,-1,sizeof(match));
    int ans = 0;
    for(int i=1;i<=n;i++)
    {
        if(match[i] == -1 && dfs(i,n))  ans++;
    }
    return ans;
}
```

**最近公共祖先倍增 lca**

```
void dfs(int now,int pre)
{
    dep[now] = dep[pre]+1;
    for(int i = 0;i < v[now].size();i++)
    {
```

```
        int t = v[now][i];
        if(t == pre)    continue;
        fa[t][0] = now;
        dfs(t,now);
    }
}

int lca(int x,int y)
{
    if(dep[x] < dep[y]) swap(x,y);
    int t = dep[x]-dep[y];
    for(int i = 0;i <= 20;i++)
    {
        if((1<<i)&t)    x = fa[x][i];
    }
    if(x == y)  return x;
    for(int i = 20;i >= 0;i--)
    {
        if(fa[x][i] != fa[y][i])
        {
            x = fa[x][i];
            y = fa[y][i];
        }
    }
    return fa[x][0];
}


int main()
{
    dfs(1,0);
    for(int i = 1;i <= 20;i++)
    {
        for(int j = 1;j <= n;j++)
        {
            fa[j][i] = fa[fa[j][i-1]][i-1];
        }
    }
}
```

lca+rmq

```
int n,m,f[N*2],rmq[N*2];
vector<int> v[N];

struct ST
{
    int mm[2*N],dp[2*N][20];
    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];
        }
    }
    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];
    }
}st;

void dfs(int now,int pre,int dep)
{
    f[++cnt] = now;
    rmq[cnt] = dep;
    p[now] = cnt;
    for(int i = 0;i < v[now].size();i++)
    {
        int t = v[now][i];
        if(t == pre)    continue;
        dfs(t,now,dep+1);
        f[++cnt] = now;
        rmq[cnt] = dep;
    }
}

void initlca(int root,int n)
{
    cnt = 0;
    dfs(root,root,0);
    st.init(2*n-1);
}

int querylca(int x,int y)
{
    return f[st.query(p[x],p[y])];
}
```

**欧拉路径**

①无向图：连通，每个顶点度数都为偶数或者仅有两个点的度数为奇数。

②有向图：底图连通，所有顶点的出度 == 入度 或者 有且仅有一个顶点 出度 == 入度+1，有且仅有一个顶点 入度 == 出度+1。

③混合图：底图连通，存在欧拉回路，则一定存在欧拉路径。否则如果有且仅有两个点的（出度-入度）是奇数，那么给这两个点加边，判断是否存在欧拉回路。

**欧拉回路**

①无向图：连通，每个顶点度数都为偶数。

②有向图：底图连通，每个顶点出度等于入度。

③混合图：底图连通，网络流判断。

**无向图欧拉路径**

```
vector<int> ans;
struct edge
{
    int to,next,id,dir,flag;
}e[20005];

void addedge(int u,int v,int id)
{
    e[cnt].to = v;
    e[cnt].id = id;
    e[cnt].dir = 0;
    e[cnt].flag = 0;
    e[cnt].next = head[u];
    head[u] = cnt++;
    e[cnt].to = u;
    e[cnt].id = id;
```

```
    e[cnt].dir = 1;
    e[cnt].flag = 0;
    e[cnt].next = head[v];
    head[v] = cnt++;
}

vector<edge> ans;
void dfs(int u)
{
    for(int i = head[u];i != -1;i = e[i].next)
    {
        if(!e[i].flag)
        {
            e[i].flag = 1;
            e[i^1].flag = 0;
            dfs(e[i].to);
            ans.push_back(i);
        }
    }
}
```

**有向图欧拉路径**

```
#define MAXN 250
vector<int> ans;

struct edge
{
    int to,next,id,flag;
}e[20005];

void addedge(int u,int v,int id)
{
    e[cnt].to = v;
    e[cnt].id = id;
    e[cnt].flag = 0;
    e[cnt].next = head[u];
    head[u] = cnt++;
}
vector<edge> ans;
void dfs(int u)
{
    for(int i = head[u];i != -1;i = e[i].next)
    {
        if(!e[i].flag)
        {
            e[i].flag = 1;
            dfs(e[i].to);
            ans.push_back(i);
        }
    }
}
```

**混合图欧拉回路判断**

```
struct edge
{
    int u,v,cap,next;
}e[20005];
int n,m,cnt,head[N],dep[N],gap[N],cur[N],st[N],que[N],ind[N],outd[N];
int main()
{
    int T;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d%d",&n,&m);
```

```
        cnt = 0;
        memset(head,-1,sizeof(head));
        memset(ind,0,sizeof(ind));
        memset(outd,0,sizeof(outd));
        while(m--)
        {
            int x,y,z;
            scanf("%d%d%d",&x,&y,&z);
            outd[x]++;
            ind[y]++;
            if(z == 0)  addedge(x,y,1);
        }
        int sum = 0,flag = 1;
        for(int i = 1;i <= n;i++)
        {
            if(outd[i]-ind[i] > 0)
            {
                addedge(0,i,(outd[i]-ind[i])/2);
                sum += (outd[i]-ind[i])/2;
            }
            else    if(ind[i]-outd[i] > 0)  addedge(i,n+1,(ind[i]-outd[i])/2);
            if((outd[i]-ind[i])%2)  flag = 0;
        }
        if(flag && isap(0,n+1,n+2) == sum)    printf("possible\n");
        else    printf("impossible\n");
    }
    return 0;
}
```

# 数据结构

**并查集**

```
void init()
{
    for(int i = 1;i <= n;i++)    pre[i] = i;
}

int findd(int x)       递归
{
    return pre[x] == x?x:pre[x] = findd(pre[x]);
}

void join(int a,int b)
{
    int x = findd(a),y = findd(b);
    if(x != y)  pre[x] = y;
}
```

**带权值并查集**

```
const int N = 50005;
int pre[N],rk[N];
/*
    解题思路: father[y] = x
    如果 rank[y]为 0 代表 x 和 y 属于同一种
    如果 rank[y]为 1 代表 x 吃 y
    如果 rank[y]为 2 代表 y 吃 x
    本题的关键就是如何在路径压缩的过程中更新关系权值
    需要总结更新公式
*/
int findd(int x)
{
    if(pre[x] == x) return x;
    int xx = findd(pre[x]);
    rk[x] = (rk[x]+rk[pre[x]])%3;
    pre[x] = xx;
    return xx;
}
bool ok(int n,int type,int x,int y)
{
    if(x > n || y > n)  return 0;
    int fx = findd(x),fy = findd(y);
    if(fx == fy)
    {
        if((rk[y]-rk[x]+3)%3 != type-1) return 0;
        return 1;
    }
    else
    {
        pre[fy] = fx;
        rk[fy] = (rk[x]-rk[y]+type-1+3)%3;
        return 1;
    }
}

int main()
{
    int n,k;
    scanf("%d %d",&n,&k);
    for(int i = 1;i <= n;i++)
    {
        pre[i] = i;
        rk[i] = 0;
    }
```

```
    int cnt = 0;
    for(int i = 0;i < k;i++)
    {
        int d,x,y;
        scanf("%d %d %d",&d,&x,&y);
        if(!ok(n,d,x,y))    cnt++;
    }
    printf("%d\n",cnt);
    return 0;
}
```

一维 rmq

```
struct ST
{
    int mm[100005],dp[100005][20];
    void init()
    {
        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] = a[i];
        }
        for(int j = 1;j <= mm[n];j++)
        {
            for(int i = 1;i+(1<<j)-1 <= n;i++)  dp[i][j] = min(dp[i][j-1],dp[i+(1<<(j-
1))][j-1]);
        }
    }
    int query(int a,int b)
    {
        if(a > b)   swap(a,b);
        int k = mm[b-a+1];
        return min(dp[a][k],dp[b-(1<<k)+1][k]);
    }
}st;
```

二维 rmq

```
//同一维一样 用 dp[row][col][i][j]表示(row,col)到(row+2^i,col+2^j)矩形内的最大值
int a[305][305],mm[305],dp[305][305][9][9];

void init(int n,int m)
{
    for(int i = 1;i <= 300;i++) mm[i] = ((i&(i-1)) == 0)?mm[i-1]+1:mm[i-1];
    for(int i = 1;i <= n;i++)
    {
        for(int j = 1;j <= m;j++)   dp[i][j][0][0] = a[i][j];
    }
    for(int i = 0;i <= mm[n];i++)
    {
        for(int j = 0;j <= mm[m];j++)
        {
            if(i==0 && j==0)    continue;
            for(int row = 1;row+(1<<i)-1 <= n;row++)
            {
                for(int col = 1;col+(1<<j)-1 <= m;col++)
                {
                    if(i)   dp[row][col][i][j]  = max(dp[row][col][i-
1][j],dp[row+(1<<(i-1))][col][i-1][j]);
                    else    dp[row][col][i][j]  = max(dp[row][col][i][j-
1],dp[row][col+(1<<(j-1))][i][j-1]);
                }
            }
        }
    }
```

```
}
int rmq(int x1,int y1,int x2,int y2)
{
    int k1 = mm[x2-x1+1],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
]));
}
```

**单调栈**

```
stack<long long> s;
long long a[100005];
int main()
{
    int n;
    while(scanf("%d",&n) && n)
    {
        while(!s.empty())    s.pop();
        long long ans = 0;
        for(int i = 1;i <= n;i++)    scanf("%lld",&a[i]);
        a[n+1] = 0;
        for(int i = 1;i <= n+1;i++)
        {
            while(!s.empty() && a[s.top()] > a[i])
            {
                int temp = s.top();
                s.pop();
                int len = s.empty()?i-1:i-s.top()-1;
                ans = max(ans,len*a[temp]);
            }
            s.push(i);
        }
        printf("%lld\n",ans);
    }
    return 0;
}
```

**单调队列**

```
//XDOJ1156
struct st
{
    int num,t;
}s;
deque<st> q;
int n,x;

int main()
{
    while(~scanf("%d",&n))
    {
        while(!q.empty())    q.pop_back();
        int out = 1,cnt = 1;
        for(int i = 1;i <= n;i++)
        {
            scanf("%d",&x);
            switch(x)
            {
                case 1:
                    int tt;
                    scanf("%d",&tt);
                    s.t = tt-i;
                    s.num = cnt++;
                    while(!q.empty() && q.back().t <= s.t)     q.pop_back();
                    q.push_back(s);
```

```
                    break;
                case 2:
                    if(q.front().num == out++)  q.pop_front();
                    break;
                case 3:
                    printf("%d\n",q.front().t+i);
            }
        }
    }
    return 0;
}
```

**一维树状数组**

```
int tree[50005],n;
inline int lowbit(int x)
{
    return x & (-x);
}

void update(int pos,int x)
{
    while(pos <= n)
    {
        tree[pos] += x;
        pos += lowbit(pos);
    }
}

int getsum(int pos)
{
    int sum = 0;
    while(pos > 0)
    {
        sum += tree[pos];
        pos -= lowbit(pos);
    }
    return sum;
}
```

**二维树状数组**

```
int tree[1005][1005] = {0},a[1005][1005] = {0};

int lowbit(int x)
{
    return x&(-x);
}

void update(int x,int y,int w)
{
    for(int i = x;i <= 1001;i += lowbit(i))
    {
        for(int j = y;j <= 1001;j += lowbit(j)) tree[i][j] += w;
    }
}

int getsum(int x,int y)
{
    int ans = 0;
    for(int i = x;i > 0;i -= lowbit(i))
    {
        for(int j = y;j > 0;j -= lowbit(j)) ans += tree[i][j];
    }
    return ans;
}
```

## 划分树查询区间第 k 大

```
int n,m,tree[20][N],sorted[N],toleft[20][N];

void build(int l,int r,int dep)
{
    if(l == r)return;
    int mid = (l+r)/2,same = mid-l+1;
    for(int i = l;i <= r;i++)
    {
        if(tree[dep][i] < sorted[mid])  same--;
    }
    int lpos = l,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);
}

int query(int l,int r,int ql,int qr,int dep,int k)
{
    if(ql == qr)    return tree[dep][ql];
    int mid = (l+r)/2,cnt = toleft[dep][qr]-toleft[dep][ql-1];
    if(cnt >= k)
    {
        int ll = l+toleft[dep][ql-1]-toleft[dep][l-1],rr = ll+cnt-1;
        return query(l,mid,ll,rr,dep+1,k);
    }
    else
    {
        int rr = qr+toleft[dep][r]-toleft[dep][qr],ll = rr-(qr-ql-cnt);
        return query(mid+1,r,ll,rr,dep+1,k-cnt);
    }
}

int main()
{
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        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);
        while(m--)
        {
            int l,r,k;
            scanf("%d%d%d",&l,&r,&k);
            printf("%d\n",query(1,n,l,r,0,k));
        }
    }
    return 0;
}
```

## 点分治求边权和等于 k 的数量

```
int n,k,root,son[10005],vis[10005],minn,ans,sum,dis[10005],dep[10005];
struct xx
{
    int to,w;
    xx(int a,int b):to(a),w(b){};
};
vector<xx> v[10005];

void dfsroot(int now,int pre)
{
    son[now] = 1;
    int maxx = 0;
    for(int i = 0;i < v[now].size();i++)
    {
        int t = v[now][i].to,w = v[now][i].w;
        if(t == pre || vis[t])  continue;
        dfsroot(t,now);
        son[now] += son[t];
        maxx = max(maxx,son[t]);
    }
    maxx = max(maxx,sum-son[now]);
    if(maxx < minn)
    {
        minn = maxx;
        root = now;
    }
}

void dfsdep(int now,int pre)
{
    dep[++dep[0]] = dis[now];
    for(int i = 0;i < v[now].size();i++)
    {
        int t = v[now][i].to,w = v[now][i].w;
        if(t == pre || vis[t])  continue;
        dis[t] = dis[now]+w;
        dfsdep(t,now);
    }
}

int calc(int now,int w)
{
    dis[now] = w;
    dep[0] = 0;
    dfsdep(now,0);
    sort(dep+1,dep+dep[0]+1);
    int l = 1,r = dep[0],cnt = 0;
    while(l < r)
    {
        if(dep[l]+dep[r] <= k)
        {
            cnt += r-l;
            l++;
        }
        else    r--;
    }
    return cnt;
}

void solve(int now)
{
    ans += calc(now,0);
```

```
    vis[now] = 1;
    for(int i = 0;i < v[now].size();i++)
    {
        int t = v[now][i].to,w = v[now][i].w;
        if(vis[t])  continue;
        ans -= calc(t,w);
        minn = 1e9;
        sum = son[t];
        dfsroot(t,0);
        solve(root);
    }
}

int main()
{
    while(scanf("%d%d",&n,&k))
    {
        if(n == 0 || k == 0)    break;
        ans = 0,sum = n;
        memset(vis,0,sizeof(vis));
        for(int i = 1;i <= n;i++)   v[i].clear();
        for(int i = 1;i < n;i++)
        {
            int x,y,z;
            scanf("%d%d%d",&x,&y,&z);
            v[x].push_back(xx(y,z));
            v[y].push_back(xx(x,z));
        }
        minn = 1e9;
        dfsroot(1,0);
        solve(root);
        printf("%d\n",ans);
    }
    return 0;
}
```

点分治求点权积等于 k

```
int n,k,root,son[100005],vis[100005],dep[100005],id[1000005],minn,ans1,ans2,sum;
long long a[100005],inv[1000005],mp[1000005];
vector<int> v[100005];

void dfsroot(int now,int pre)
{
    son[now] = 1;
    int maxx = 0;
    for(int i = 0;i < v[now].size();i++)
    {
        int t = v[now][i];
        if(t == pre || vis[t])  continue;
        dfsroot(t,now);
        son[now] += son[t];
        maxx = max(maxx,son[t]);
    }
    maxx = max(maxx,sum-son[now]);
    if(maxx < minn)
    {
        minn = maxx;
        root = now;
    }
}

void dfsdep(int now,int pre,long long x)
{
    dep[++dep[0]] = x*a[now]%MOD;
```

```
    id[dep[0]] = now;
    for(int i = 0;i < v[now].size();i++)
    {
        int t = v[now][i];
        if(t == pre || vis[t])  continue;
        dfsdep(t,now,x*a[now]%MOD);
    }
}

void update(int now,int x,int y)
{
    int t = mp[inv[x*a[now]%MOD]*k%MOD];
    if(!t)  return;
    if(t > y)   swap(t,y);
    if(t < ans1 || t == ans1 && y < ans2)
    {
        ans1 = t;
        ans2 = y;
    }
}

void solve(int now)
{
    vis[now] = 1;
    mp[1] = now;
    for(int i = 0;i < v[now].size();i++)
    {
        int t = v[now][i];
        if(vis[t])  continue;
        dep[0] = 0;
        dfsdep(t,now,1);
        for(int j = 1;j <= dep[0];j++)  update(now,dep[j],id[j]);
        for(int j = 1;j <= dep[0];j++)
        {
            if(!mp[dep[j]] || mp[dep[j]] > id[j])   mp[dep[j]] = id[j];
        }
    }
    mp[1] = 0;
    for(int i = 0;i < v[now].size();i++)
    {
        int t = v[now][i];
        if(vis[t])  continue;
        dep[0] = 0;
        dfsdep(t,now,1);
        for(int j = 1;j <= dep[0];j++)  mp[dep[j]] = 0;
    }
    for(int i = 0;i < v[now].size();i++)
    {
        int t = v[now][i];
        if(vis[t])  continue;
        minn = 1e9;
        sum = son[t];
        dfsroot(t,0);
        solve(root);
    }
}

int main()
{
    inv[1] = 1;
    for(int i = 2;i < MOD;i++)  inv[i] = (MOD-MOD/i)*inv[MOD%i]%MOD;
    while(~scanf("%d%d",&n,&k))
    {
        ans1 = 1e9,ans2 = 0;
```

```
        memset(vis,0,sizeof(vis));
        for(int i = 1;i <= n;i++)   v[i].clear();
        for(int i = 1;i <= n;i++)   scanf("%lld",&a[i]);
        for(int i = 1;i < n;i++)
        {
            int x,y;
            scanf("%d%d",&x,&y);
            v[x].push_back(y);
            v[y].push_back(x);
        }
        minn = 1e9;
        sum = n;
        dfsroot(1,0);
        solve(root);
        if(ans1 == 1e9) printf("No solution\n");
        else    printf("%d %d\n",ans1,ans2);
    }
    return 0;
}
```

**cdq 分治求三维偏序**

```
int n,maxx,tree[100005],ans[100005];
struct xx
{
    int x,y,z,id;
    xx(int a,int b,int c,int d):x(a),y(b),z(c),id(d){};
    xx(){};
}a[100005],b[100005];

inline lowbit(int x)
{
    return x&-x;
}

void add(int pos,int x)
{
    while(pos <= maxx)
    {
        tree[pos] += x;
        pos += lowbit(pos);
    }
}

int getsum(int pos)
{
    int ans = 0;
    while(pos > 0)
    {
        ans += tree[pos];
        pos -= lowbit(pos);
    }
    return ans;
}

bool cmp1(xx a,xx b)
{
    if(a.x != b.x)  return a.x < b.x;
    if(a.y != b.y)  return a.y < b.y;
    return a.z < b.z;
}

bool cmp2(xx a,xx b)
{
    if(a.y != b.y)  return a.y < b.y;
```

```
    return a.id < b.id;
}

void cdq(int l,int r)
{
    if(l == r)  return;
    int mid = (l+r)/2;
    int cnt = 0;
    for(int i = l;i <= mid;i++) b[++cnt] = xx(0,a[i].y,a[i].z,0);
    for(int i = mid+1;i <= r;i++)   b[++cnt] = xx(0,a[i].y,a[i].z,a[i].id);
    sort(b+1,b+1+cnt,cmp2);
    for(int i = 1;i <= cnt;i++)
    {
        if(b[i].id == 0)    add(b[i].z,1);
        else    ans[b[i].id] += getsum(b[i].z);
    }
    for(int i = 1;i <= cnt;i++)
    {
        if(b[i].id == 0)    add(b[i].z,-1);
    }
    cdq(l,mid);
    cdq(mid+1,r);
}

int main()
{
    int T;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d",&n);
        maxx = 0;
        for(int i = 1;i <= n;i++)
        {
            scanf("%d%d%d",&a[i].x,&a[i].y,&a[i].z);
            a[i].id = i;
            maxx = max(maxx,a[i].z);
        }
        sort(a+1,a+1+n,cmp1);
        memset(ans,0,sizeof(ans));
        int cnt = 0;
        for(int i = n-1;i >= 1;i--)
        {
            if(a[i].x == a[i+1].x && a[i].y == a[i+1].y && a[i].z == a[i+1].z)  cnt++;
            else    cnt = 0;
            ans[a[i].id] += cnt;
        }
        cdq(1,n);
        for(int i = 1;i <= n;i++)   printf("%d\n",ans[i]);
    }
    return 0;
}
```

**树链剖分线段树(点权)**

```
//点权，修改路径点权，查询单点权
int n,m,q,cnt,a[N],top[N],fa[N],dep[N],num[N],p[N],fp[N],son[N];
vector<int> v[N];
char s[10];

struct segtree
{
    int l,r;
    long long x,lazy;
}tree[4*N];
```

```
inline void pushup(int pos)
{
    tree[pos].x = tree[pos<<1].x+tree[pos<<1|1].x;
}

inline void pushdown(int pos)
{
    if(tree[pos].lazy)
    {
        tree[pos<<1].x += (tree[pos<<1].r-tree[pos<<1].l+1)*tree[pos].lazy;
        tree[pos<<1|1].x += (tree[pos<<1|1].r-tree[pos<<1|1].l+1)*tree[pos].lazy;
        tree[pos<<1].lazy += tree[pos].lazy;
        tree[pos<<1|1].lazy += tree[pos].lazy;
        tree[pos].lazy = 0;
    }
}

void build(int pos,int l,int r)
{
    tree[pos].l = l;
    tree[pos].r = r;
    tree[pos].lazy = 0;
    if(l == r)
    {
        tree[pos].x = a[fp[l]];
        return;
    }
    int mid = (l+r)/2;
    build(pos<<1,l,mid);
    build(pos<<1|1,mid+1,r);
    pushup(pos);
}

void update(int pos,int l,int r,long long x)
{
    if(r < tree[pos].l || tree[pos].r < l)  return;
    if(l <= tree[pos].l && tree[pos].r <= r)
    {
        tree[pos].x += (tree[pos].r-tree[pos].l+1)*x;
        tree[pos].lazy += x;
        return;
    }
    pushdown(pos);
    update(pos<<1,l,r,x);
    update(pos<<1|1,l,r,x);
    pushup(pos);
}

long long getsum(int pos,int l,int r)
{
    if(r < tree[pos].l || tree[pos].r < l)  return 0;
    if(l <= tree[pos].l && tree[pos].r <= r)    return tree[pos].x;
    pushdown(pos);
    return  getsum(pos<<1,l,r)+getsum(pos<<1|1,l,r);
}

void dfs(int now,int pre,int d)
{
    dep[now] = d;
    fa[now] = pre;
    num[now] = 1;
    for(int i = 0;i < v[now].size();i++)
    {
```

```
        int t = v[now][i];
        if(t == pre)    continue;
        dfs(t,now,d+1);
        num[now] += num[t];
        if(son[now] == -1 || num[t] > num[son[now]])    son[now] = t;
    }
}

void getpos(int now,int sp)
{
    top[now] = sp;
    p[now] = ++cnt;
    fp[p[now]] = now;
    if(son[now] == -1)  return;
    getpos(son[now],sp);
    for(int i = 0;i < v[now].size();i++)
    {
        int t = v[now][i];
        if(t != son[now] && t != fa[now])   getpos(t,t);
    }
}

void change(int u,int v,int x)
{
    int f1 = top[u],f2 = top[v];
    while(f1 != f2)
    {
        if(dep[f1] < dep[f2])
        {
            swap(f1,f2);
            swap(u,v);
        }
        update(1,p[f1],p[u],x);
        u = fa[f1];
        f1 = top[u];
    }
    if(dep[u] > dep[v]) swap(u,v);
    update(1,p[u],p[v],x);
}

int main()
{
    while(~scanf("%d%d%d",&n,&m,&q))
    {
        memset(son,-1,sizeof(son));
        memset(tree,0,sizeof(tree));
        for(int i = 1;i <= n;i++)   v[i].clear();
        cnt = 0;
        for(int i = 1;i <= n;i++)   scanf("%d",&a[i]);
        while(m--)
        {
            int x,y;
            scanf("%d%d",&x,&y);
            v[x].push_back(y);
            v[y].push_back(x);
        }
        dfs(1,0,0);
        getpos(1,1);
        build(1,1,cnt);
        while(q--)
        {
            scanf("%s",s);
            if(s[0] == 'Q')
            {
```

```
                int x;
                scanf("%d",&x);
                printf("%lld\n",getsum(1,p[x],p[x]));
            }
            else
            {
                int x,y,z;
                scanf("%d%d%d",&x,&y,&z);
                if(s[0] == 'D')     z = -z;
                change(x,y,z);
            }
        }
    }
    return 0;
}
```

**树链剖分线段树(边权)**

```
int n,m,q,cnt,a[N],val[N],top[N],fa[N],dep[N],num[N],p[N],fp[N],son[N],e[N][3];
struct segtree
{
    int l,r,x;
}tree[4*N];
vector<int> v[N];
char s[10];

void pushup(int pos)
{
    tree[pos].x = max(tree[pos<<1].x,tree[pos<<1|1].x);
}

void build(int pos,int l,int r)
{
    tree[pos].l = l;
    tree[pos].r = r;
    if(l == r)
    {
        tree[pos].x = val[l];
        return;
    }
    int mid = (l+r)/2;
    build(pos<<1,l,mid);
    build(pos<<1|1,mid+1,r);
    pushup(pos);
}

void update(int pos,int l,int r,int x)
{
    if(r < tree[pos].l || tree[pos].r < l)  return;
    if(l <= tree[pos].l && tree[pos].r <= r)
    {
        tree[pos].x = x;
        return;
    }
    update(pos<<1,l,r,x);
    update(pos<<1|1,l,r,x);
    pushup(pos);
}

int query(int pos,int l,int r)
{
    if(r < tree[pos].l || tree[pos].r < l)  return 0;
    if(l <= tree[pos].l && tree[pos].r <= r)    return tree[pos].x;
    return  max(query(pos<<1,l,r),query(pos<<1|1,l,r));
}
```

```
void dfs(int now,int pre,int d)
{
    dep[now] = d;
    fa[now] = pre;
    num[now] = 1;
    for(int i = 0;i < v[now].size();i++)
    {
        int t = v[now][i];
        if(t == pre)    continue;
        dfs(t,now,d+1);
        num[now] += num[t];
        if(son[now] == -1 || num[t] > num[son[now]])    son[now] = t;
    }
}

void getpos(int now,int sp)
{
    top[now] = sp;
    p[now] = ++cnt;
    fp[p[now]] = now;
    if(son[now] == -1)  return;
    getpos(son[now],sp);
    for(int i = 0;i < v[now].size();i++)
    {
        int t = v[now][i];
        if(t != son[now] && t != fa[now])   getpos(t,t);
    }
}

int solve(int u,int v)
{
    int f1 = top[u],f2 = top[v],t = 0;
    while(f1 != f2)
    {
        if(dep[f1] < dep[f2])
        {
            swap(f1,f2);
            swap(u,v);
        }
        t = max(t,query(1,p[f1],p[u]));
        u = fa[f1];
        f1 = top[u];
    }
    if(u == v)  return t;
    if(dep[u] > dep[v]) swap(u,v);
    return max(t,query(1,p[son[u]],p[v]));
}

int main()
{
    int T;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d",&n);
        memset(son,-1,sizeof(son));
        for(int i = 1;i <= n;i++)   v[i].clear();
        cnt = 0;
        for(int i = 1;i < n;i++)
        {
            scanf("%d%d%d",&e[i][0],&e[i][1],&e[i][2]);
            v[e[i][0]].push_back(e[i][1]);
            v[e[i][1]].push_back(e[i][0]);
```

```
        }
        dfs(1,0,0);
        getpos(1,1);
        for(int i = 1;i < n;i++)
        {
            if(dep[e[i][0]] > dep[e[i][1]]) swap(e[i][0],e[i][1]);
            val[p[e[i][1]]] = e[i][2];
        }
        build(1,1,cnt);
        while(scanf("%s",s))
        {
            if(s[0] == 'D') break;
            int x,y;
            scanf("%d%d",&x,&y);
            if(s[0] == 'Q') printf("%d\n",solve(x,y));
            else    update(1,p[e[x][1]],p[e[x][1]],y);
        }
    }
    return 0;
}
```

**线段树模版**

```
struct segtree
{
    int l,r;
    long long x,lazy;
}tree[400005];

inline void pushup(int pos)
{
    tree[pos].x = tree[pos<<1].x+tree[pos<<1|1].x;
}

inline void pushdown(int pos)
{
    if(tree[pos].lazy)
    {
        tree[pos<<1].x += (tree[pos<<1].r-tree[pos<<1].l+1)*tree[pos].lazy;
        tree[pos<<1|1].x += (tree[pos<<1|1].r-tree[pos<<1|1].l+1)*tree[pos].lazy;
        tree[pos<<1].lazy += tree[pos].lazy;
        tree[pos<<1|1].lazy += tree[pos].lazy;
        tree[pos].lazy = 0;
    }
}

void build(int pos,int l,int r)
{
    tree[pos].l = l;
    tree[pos].r = r;
    tree[pos].lazy = 0;
    if(l == r)
    {
        tree[pos].x = a[l];
        return;
    }
    int mid = (l+r)/2;
    build(pos<<1,l,mid);
    build(pos<<1|1,mid+1,r);
    pushup(pos);
}

void update(int pos,int l,int r,long long x)
{
    if(r < tree[pos].l || tree[pos].r < l)  return;
```

```
    if(l <= tree[pos].l && tree[pos].r <= r)
    {
        tree[pos].x = (tree[pos].r-tree[pos].l+1)*x;
        tree[pos].lazy = x;
        return;
    }
    pushdown(pos);
    update(pos<<1,l,r,x);
    update(pos<<1|1,l,r,x);
    pushup(pos);
}

long long getsum(int pos,int l,int r)
{
    if(r < tree[pos].l || tree[pos].r < l)  return 0;
    if(l <= tree[pos].l && tree[pos].r <= r)    return tree[pos].x;
    pushdown(pos);
    return  getsum(pos<<1,l,r)+getsum(pos<<1|1,l,r);
}
```

**动态开点线段树**

```
int n,k,q,cnt = 0,a[100005];
struct segtree
{
    int l,r,x,lazy;
    segtree *lson,*rson;
}*root;

segtree *newnode(int l,int r)
{
    segtree *t = new segtree;
    t->l = l;
    t->r = r;
    t->lson = NULL;
    t->rson = NULL;
    t->lazy = 0;
    t->x = a[l];
    return t;
}

segtree *newlson(segtree *pos)
{
    int mid = (pos->l+pos->r)/2;
    return newnode(pos->l,mid);
}

segtree *newrson(segtree *pos)
{
    int mid = (pos->l+pos->r)/2;
    return newnode(mid+1,pos->r);
}

void pushup(segtree *pos)
{
    if(!pos->lson)  pos->lson = newlson(pos);
    if(!pos->rson)  pos->rson = newrson(pos);
    pos->x = min(pos->lson->x,pos->rson->x);
}

void pushdown(segtree *pos)
{
    if(!pos->lson)  pos->lson = newlson(pos);
    if(!pos->rson)  pos->rson = newrson(pos);
    if(pos->lazy)
```

```
    {
        pos->lson->x = pos->lazy;
        pos->rson->x = pos->lazy;
        pos->lson->lazy = pos->lazy;
        pos->rson->lazy = pos->lazy;
        pos->lazy = 0;
    }
}

void update(segtree *pos,int l,int r,int x)
{
    if(r < pos->l || pos->r < l)    return;
    if(l <= pos->l && pos->r <= r)
    {
        pos->x = x;
        pos->lazy = x;
        return;
    }
    pushdown(pos);
    update(pos->lson,l,r,x);
    update(pos->rson,l,r,x);
    pushup(pos);
}

int getmin(segtree *pos,int l,int r)
{
    if(r < pos->l || pos->r < l)    return 1e9;
    if(l <= pos->l && pos->r <= r)  return pos->x;
    pushdown(pos);
    return min(getmin(pos->lson,l,r),getmin(pos->rson,l,r));
}
```

**区间取模，单点修改，询问区间和**

```
int n,m;
long long a[100005];
struct segtree
{
    int l,r;
    long long x,sum;
}tree[400005];

inline void pushup(int pos)
{
    tree[pos].x = max(tree[pos<<1].x,tree[pos<<1|1].x);
    tree[pos].sum = tree[pos<<1].sum+tree[pos<<1|1].sum;
}

void build(int pos,int l,int r)
{
    tree[pos].l = l;
    tree[pos].r = r;
    if(l >= r)
    {
        tree[pos].x = a[l];
        tree[pos].sum = a[l];
        return;
    }
    int mid = (l+r)/2;
    build(pos<<1,l,mid);
    build(pos<<1|1,mid+1,r);
    pushup(pos);
}

void edit(int pos,int l,int r,long long x,int t)
```

```
{
    if(t < l || t > r)  return;
    if(l == r)
    {
        tree[pos].x = x;
        tree[pos].sum = x;
        return;
    }
    int mid = (l+r)/2;
    edit(pos<<1,l,mid,x,t);
    edit(pos<<1|1,mid+1,r,x,t);
    pushup(pos);
}

void mod(int pos,int l,int r,long long x)
{
    if(r < tree[pos].l || tree[pos].r < l || tree[pos].x < x)   return;
    if(tree[pos].l == tree[pos].r)
    {
        tree[pos].x %= x;
        tree[pos].sum = tree[pos].x;
        return;
    }
    mod(pos<<1,l,r,x);
    mod(pos<<1|1,l,r,x);
    pushup(pos);
}

long long getsum(int pos,int l,int r)
{
    if(r < tree[pos].l || tree[pos].r < l)  return 0;
    if(l <= tree[pos].l && tree[pos].r <= r)    return tree[pos].sum;
    return  getsum(pos<<1,l,r)+getsum(pos<<1|1,l,r);
}
```

**区间加减，区间开根，询问区间和**

```
int n,m;
long long a[100005];
struct segtree
{
    int l,r;
    long long ma,mi,sum,lazy;
}tree[400005];

inline void pushup(int pos)
{
    tree[pos].ma = max(tree[pos<<1].ma,tree[pos<<1|1].ma);
    tree[pos].mi = min(tree[pos<<1].mi,tree[pos<<1|1].mi);
    tree[pos].sum = tree[pos<<1].sum+tree[pos<<1|1].sum;
}

inline void pushdown(int pos)
{
    if(tree[pos].lazy)
    {
        long long t = tree[pos].lazy;
        tree[pos<<1].lazy += t;
        tree[pos<<1|1].lazy += t;
        tree[pos<<1].mi += t;
        tree[pos<<1|1].mi += t;
        tree[pos<<1].ma += t;
        tree[pos<<1|1].ma += t;
        tree[pos<<1].sum += (tree[pos<<1].r-tree[pos<<1].l+1)*t;
        tree[pos<<1|1].sum += (tree[pos<<1|1].r-tree[pos<<1|1].l+1)*t;
```

```
        tree[pos].lazy = 0;
    }
}

void build(int pos,int l,int r)
{
    tree[pos].l = l;
    tree[pos].r = r;
    tree[pos].lazy = 0;
    if(l == r)
    {
        tree[pos].mi = a[l];
        tree[pos].ma = a[l];
        tree[pos].sum = a[l];
        return;
    }
    int mid = (l+r)/2;
    build(pos<<1,l,mid);
    build(pos<<1|1,mid+1,r);
    pushup(pos);
}

long long getsum(int pos,int l,int r)
{
    if(r < tree[pos].l || tree[pos].r < l)  return 0;
    if(l <= tree[pos].l && tree[pos].r <= r)    return tree[pos].sum;
    pushdown(pos);
    return  getsum(pos<<1,l,r)+getsum(pos<<1|1,l,r);
}

void update(int pos,int l,int r,long long x)
{
    if(r < tree[pos].l || tree[pos].r < l)    return;
    if(l <= tree[pos].l && tree[pos].r <= r)
    {
        tree[pos].mi += x;
        tree[pos].ma += x;
        tree[pos].sum += (tree[pos].r-tree[pos].l+1)*x;
        tree[pos].lazy += x;
        return;
    }
    pushdown(pos);
    update(pos<<1,l,r,x);
    update(pos<<1|1,l,r,x);
    pushup(pos);
}

void sq(int pos,int l,int r)
{
    if(r < tree[pos].l || tree[pos].r < l)    return;
    if(l <= tree[pos].l && tree[pos].r <= r && (long long)sqrt(tree[pos].ma)-(long
long)sqrt(tree[pos].mi) == tree[pos].ma-tree[pos].mi)
    {
        long long t = tree[pos].ma-(long long)sqrt(tree[pos].ma);
        tree[pos].mi -= t;
        tree[pos].ma -= t;
        tree[pos].sum -= (tree[pos].r-tree[pos].l+1)*t;
        tree[pos].lazy -= t;
        return;
    }
    pushdown(pos);
    sq(pos<<1,l,r);
    sq(pos<<1|1,l,r);
    pushup(pos);
```

```
}
```

区间最小值，询问区间最大值，询问区间和

```
int n,m;
long long a[1000005];
struct segtree
{
    int l,r,max1,max2,num;
    long long sum;
}tree[4000005];

inline void pushup(int pos)
{
    tree[pos].max1 = max(tree[pos<<1].max1,tree[pos<<1|1].max1);
    tree[pos].sum = tree[pos<<1].sum+tree[pos<<1|1].sum;
    tree[pos].num = 0;
    tree[pos].max2 = max(tree[pos<<1].max2,tree[pos<<1|1].max2);
    if(tree[pos].max1 == tree[pos<<1].max1) tree[pos].num += tree[pos<<1].num;
    else    tree[pos].max2 = max(tree[pos].max2,tree[pos<<1].max1);
    if(tree[pos].max1 == tree[pos<<1|1].max1)  tree[pos].num += tree[pos<<1|1].num;
    else    tree[pos].max2 = max(tree[pos].max2,tree[pos<<1|1].max1);
}

inline void pushdown(int pos)
{
    if(tree[pos].max1 < tree[pos<<1].max1)
    {
        tree[pos<<1].sum -= (long long)tree[pos<<1].num*(tree[pos<<1].max1-
tree[pos].max1);
        tree[pos<<1].max1 = tree[pos].max1;
    }
    if(tree[pos].max1 < tree[pos<<1|1].max1)
    {
        tree[pos<<1|1].sum -= (long long)tree[pos<<1|1].num*(tree[pos<<1|1].max1-
tree[pos].max1);
        tree[pos<<1|1].max1 = tree[pos].max1;
    }
}

void build(int pos,int l,int r)
{
    tree[pos].l = l;
    tree[pos].r = r;
    if(l == r)
    {
        scanf("%d",&tree[pos].max1);
        tree[pos].sum = tree[pos].max1;
        tree[pos].max2 = -1;
        tree[pos].num = 1;
        return;
    }
    int mid = (l+r)/2;
    build(pos<<1,l,mid);
    build(pos<<1|1,mid+1,r);
    pushup(pos);
}

void update(int pos,int l,int r,int x)
{
    if(r < tree[pos].l || tree[pos].r < l || x >= tree[pos].max1)   return;
    if(l <= tree[pos].l && tree[pos].r <= r && tree[pos].max2 < x)
    {
        tree[pos].sum -= (long long)tree[pos].num*(tree[pos].max1-x);
        tree[pos].max1 = x;
```

```
        return;
    }
    pushdown(pos);
    update(pos<<1,l,r,x);
    update(pos<<1|1,l,r,x);
    pushup(pos);
}

long long getsum(int pos,int l,int r)
{
    if(r < tree[pos].l || tree[pos].r < l)  return 0;
    if(l <= tree[pos].l && tree[pos].r <= r)    return tree[pos].sum;
    pushdown(pos);
    return  getsum(pos<<1,l,r)+getsum(pos<<1|1,l,r);
}

int getmax(int pos,int l,int r)
{
    if(r < tree[pos].l || tree[pos].r < l)  return 0;
    if(l <= tree[pos].l && tree[pos].r <= r)    return tree[pos].max1;
    pushdown(pos);
    return max(getmax(pos<<1,l,r),getmax(pos<<1|1,l,r));
}
```

**区间加减，区间覆盖，询问区间最大值，询问历史区间最大值**

```
int n,m;
char s[10];
long long a[1000005];
struct segtree
{
    int l,r,nmax,pmax,nlazy,plazy,ncov,pcov;
}tree[4000005];

inline void pushup(int pos)
{
    tree[pos].nmax = max(tree[ls].nmax,tree[rs].nmax);
    tree[pos].pmax = max(tree[ls].pmax,tree[rs].pmax);
}

inline void pushdown(int pos)
{
    tree[ls].pmax =
max(tree[ls].pmax,max(tree[pos].pcov,tree[pos].plazy+tree[ls].nmax));
    tree[rs].pmax =
max(tree[rs].pmax,max(tree[pos].pcov,tree[pos].plazy+tree[rs].nmax));
    if(tree[pos].ncov == -INF)
    {
        tree[ls].nmax += tree[pos].nlazy;
        tree[rs].nmax += tree[pos].nlazy;
        if(tree[ls].ncov == -INF)
        {
            tree[ls].plazy = max(tree[ls].plazy,tree[ls].nlazy+tree[pos].plazy);
            tree[ls].nlazy += tree[pos].nlazy;
        }
        else
        {
            tree[ls].pcov = max(tree[ls].pcov,tree[ls].ncov+tree[pos].plazy);
            tree[ls].ncov = tree[ls].nmax;
        }
        if(tree[rs].ncov == -INF)
        {
            tree[rs].plazy = max(tree[rs].plazy,tree[rs].nlazy+tree[pos].plazy);
            tree[rs].nlazy += tree[pos].nlazy;
        }
```

```
        else
        {
            tree[rs].pcov = max(tree[rs].pcov,tree[rs].ncov+tree[pos].plazy);
            tree[rs].ncov = tree[rs].nmax;
        }
    }
    else
    {
        if(tree[ls].ncov == -INF)   tree[ls].plazy =
max(tree[ls].plazy,tree[ls].nlazy+tree[pos].plazy);
        else    tree[ls].pcov = max(tree[ls].pcov,tree[ls].nmax+tree[pos].plazy);
        if(tree[rs].ncov == -INF)   tree[rs].plazy =
max(tree[rs].plazy,tree[rs].nlazy+tree[pos].plazy);
        else    tree[rs].pcov = max(tree[rs].pcov,tree[rs].nmax+tree[pos].plazy);
        tree[ls].nmax = tree[pos].ncov;
        tree[rs].nmax = tree[pos].ncov;
        tree[ls].ncov = tree[pos].ncov;
        tree[rs].ncov = tree[pos].ncov;
        tree[ls].pcov = max(tree[ls].pcov,tree[pos].pcov);
        tree[rs].pcov = max(tree[rs].pcov,tree[pos].pcov);
    }
    tree[pos].nlazy = 0;
    tree[pos].plazy = 0;
    tree[pos].ncov = -INF;
    tree[pos].pcov = -INF;
}

void build(int pos,int l,int r)
{
    tree[pos].l = l;
    tree[pos].r = r;
    tree[pos].ncov = -INF;
    tree[pos].pcov = -INF;
    tree[pos].nlazy = 0;
    tree[pos].plazy = 0;
    if(l == r)
    {
        scanf("%d",&tree[pos].nmax);
        tree[pos].pmax = tree[pos].nmax;
        return;
    }
    int mid = (l+r)/2;
    build(ls,l,mid);
    build(rs,mid+1,r);
    pushup(pos);
}

long long getans(int pos,int l,int r,int t)
{
    if(r < tree[pos].l || tree[pos].r < l)  return -INF;
    if(l <= tree[pos].l && tree[pos].r <= r)    return t?tree[pos].pmax:tree[pos].nmax;
    pushdown(pos);
    return  max(getans(ls,l,r,t),getans(rs,l,r,t));
}

void update1(int pos,int l,int r,int x)
{
    if(r < tree[pos].l || tree[pos].r < l)  return;
    if(l <= tree[pos].l && tree[pos].r <= r)
    {
        tree[pos].nmax += x;
        tree[pos].pmax = max(tree[pos].pmax,tree[pos].nmax);
        if(tree[pos].ncov == -INF)
        {
```

```
            tree[pos].nlazy += x;
            tree[pos].plazy = max(tree[pos].plazy,tree[pos].nlazy);
        }
        else
        {
            tree[pos].ncov = tree[pos].nmax;
            tree[pos].pcov = max(tree[pos].pcov,tree[pos].ncov);
        }
        return;
    }
    pushdown(pos);
    update1(ls,l,r,x);
    update1(rs,l,r,x);
    pushup(pos);
}

void update2(int pos,int l,int r,int x)
{
    if(r < tree[pos].l || tree[pos].r < l)  return;
    if(l <= tree[pos].l && tree[pos].r <= r)
    {
        tree[pos].nmax = x;
        tree[pos].pmax = max(tree[pos].pmax,tree[pos].nmax);
        tree[pos].ncov = x;
        tree[pos].pcov = max(tree[pos].pcov,tree[pos].ncov);
        return;
    }
    pushdown(pos);
    update2(ls,l,r,x);
    update2(rs,l,r,x);
    pushup(pos);
}
```

**块状数组**

```
int n,m,a[200005],cnt[200005],next[200005],belong[200005],l[1005],r[1005];

void build()
{
    int len = sqrt(n);
    int num = (n+len-1)/len;
    if(n%len) num++;
    for(int i = 1;i <= num;i++)
    {
        l[i] = (i-1)*len+1;
        r[i] = i*len;
    }
    r[num] = n;
    for(int i = 1;i <= n;i++)   belong[i] = (i-1)/len+1;
    for(int i = n;i >= 1;i--)
    {
        int t = i+a[i];
        if(t > n)
        {
            cnt[i] = 1;
            next[i] = -1;
        }
        else if(belong[i] == belong[t])
        {
            cnt[i] = cnt[t]+1;
            next[i] = next[t];
        }
        else
        {
            cnt[i] = 1;
```

```
            next[i] = t;
        }
    }
}

void update(int pos,int x)
{
    a[pos] = x;
    for(int i = pos;i >= l[belong[pos]];i--)
    {
        int t = i+a[i];
        if(t > n)
        {
            cnt[i] = 1;
            next[i] = -1;
        }
        else if(belong[i] == belong[t])
        {
            cnt[i] = cnt[t]+1;
            next[i] = next[t];
        }
        else
        {
            cnt[i] = 1;
            next[i] = t;
        }
    }
}

int getsum(int x)
{
    int ans = cnt[x];
    while(next[x] > 0)
    {
        x = next[x];
        ans += cnt[x];
    }
    return ans;
}
```

**莫队**

```
int a[30005],num[30005],n,m;
long long ans[30005];
struct node
{
    int l,r,num,belong;
    friend bool operator<(node a,node b)
    {
        if(a.belong == b.belong)    return a.r < b.r;
        return a.l < b.l;
    }
}query[30005];

int main()
{
    while(~scanf("%d%d",&n,&m))
    {
        int size = sqrt(n);
        for(int i = 1;i <= n;i++)    scanf("%d",&a[i]);
        for(int i = 1;i <= m;i++)
        {
            scanf("%d%d",&query[i].l,&query[i].r);
            query[i].num = i;
            query[i].belong = (query[i].l-1)/size+1;
```

```
        }
        sort(query+1,query+1+m);
        memset(num,0,sizeof(num));
        int ll = 1,rr = 1;
        LL now = 1;
        num[a[1]]++;
        for(int i = 1;i <= m;i++)
        {
            while(rr < query[i].r)
            {
                rr++;
                num[a[rr]]++;
                //now = ;
            }
            while(ll > query[i].l)
            {
                ll--;
                num[a[ll]]++;
                //now = ;
            }
            while(ll < query[i].l)
            {
                //now = ;
                num[a[ll]]--;
                ll++;
            }
            while(rr > query[i].r)
            {
                //now = ;
                num[a[rr]]--;
                rr--;
            }
            ans[query[i].num] = now;
        }
        for(int i = 1;i <= m;i++)    printf("%lld\n",ans[i]);
    }
    return 0;
}
```

## 主席树

### 静态区间第 k 大

```
int n,q,m,tot,a[N],b[N];
int tree[N],lson[N*30],rson[N*30],c[N*30];

void init_hash()
{
    for(int i = 1;i <= n;i++) b[i]=a[i];
    sort(b+1,b+n+1);
    m = unique(b+1,b+n+1)-b-1;
}

int gethash(int x)
{
    return lower_bound(b+1,b+1+m,x)-b;
}

void build(int &now,int l,int r)
{
    now = tot++;
    c[now] = 0;
    if(l == r)  return;
    int mid = (l+r)/2;
    build(lson[now],l,mid);
    build(rson[now],mid+1,r);
}

void update(int &now,int last,int l,int r,int pos,int x)
{
    now = tot++;
    lson[now] = lson[last];
    rson[now] = rson[last];
    c[now] = c[last]+x;
    if(l == r)  return;
    int mid = (l+r)/2;
    if(pos <= mid)  update(lson[now],lson[last],l,mid,pos,x);
    else    update(rson[now],rson[last],mid+1,r,pos,x);
}

int query(int lt,int rt,int l,int r,int k)
{
    if(l == r)  return l;
    int mid = (l+r)/2;
    if(c[lson[rt]]-c[lson[lt]] >= k)    return query(lson[lt],lson[rt],l,mid,k);
    return query(rson[lt],rson[rt],mid+1,r,k-c[lson[rt]]+c[lson[lt]]);
}

int main()
{
    ios::sync_with_stdio(0);
    while(~scanf("%d%d",&n,&q))
    {
        for(int i = 1;i <= n;i++)   scanf("%d",&a[i]);
        init_hash();
        tot = 0;
        build(tree[0],1,m);
        for(int i = 1;i <= n;i++)    update(tree[i],tree[i-1],1,m,gethash(a[i]),1);
        while(q--)
        {
            int l,r,k;
            scanf("%d%d%d",&l,&r,&k);
            printf("%d\n",b[query(tree[l-1],tree[r],1,m,k)]);
        }
```

```
    }
    return 0;
}
```

**静态区间查询不同的数个数**

```
int n,q,m,tot,a[N],b[N];
int tree[N],lson[N*100],rson[N*100],c[N*100];
map<int,int> mp;

int build(int l,int r)
{
    int now = tot++;
    c[now] = 0;
    if(l == r)  return now;
    int mid = (l+r)/2;
    lson[now] = build(l,mid);
    rson[now] = build(mid+1,r);
    return now;
}

int update(int last,int pos,int x)
{
    int now = tot++,t = now;
    c[now] = c[last]+x;
    int l = 1,r = n;
    while(l < r)
    {
        int mid  = (l+r)/2;
        if(pos <= mid)
        {
            lson[now] = tot++;
            rson[now] = rson[last];
            now = lson[now];
            last = lson[last];
            r = mid;
        }
        else
        {
            rson[now] = tot++;
            lson[now] = lson[last];
            now = rson[now];
            last = rson[last];
            l = mid+1;
        }
        c[now] = c[last]+x;
    }
    return t;
}

int query(int pos,int now,int l,int r)
{
    if(l == r)  return c[now];
    int mid = (l+r)/2;
    if(pos <= mid)  return query(pos,lson[now],l,mid)+c[rson[now]];
    return query(pos,rson[now],mid+1,r);
}

int main()
{

    while(~scanf("%d",&n))
    {
        for(int i = 1;i <= n;i++)    scanf("%d",&a[i]);
        tot = 0;
```

```
        tree[0] = build(1,n);
        for(int i = 1;i <= n;i++)
        {
            if(mp.find(a[i]) == mp.end())   tree[i] = update(tree[i-1],i,1);
            else
            {
                int t = update(tree[i-1],mp[a[i]],-1);
                tree[i] = update(t,i,1);
            }
            mp[a[i]] = i;
        }
        scanf("%d",&q);
        while(q--)
        {
            int l,r;
            scanf("%d%d",&l,&r);
            printf("%d\n",query(l,tree[r],1,n));
        }
    }
    return 0;
}
```

**树上路径点权第 k 大**

```
int n,q,m,tot,cnt,a[N],b[N],p[N],f[N*2],rmq[N*2];
int tree[N],lson[N*40],rson[N*40],c[N*40];
vector<int> v[N];
struct ST
{
    int mm[2*N],dp[2*N][20];
    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];
        }
    }
    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];
    }
}st;

void inithash()
{
    for(int i = 1;i <= n;i++)   b[i] = a[i];
    sort(b+1,b+1+n);
    m = unique(b+1,b+1+n)-b-1;
}

int gethash(int x)
{
    return lower_bound(b+1,b+1+m,x)-b;
}

void dfs(int now,int pre,int dep)
```

```
{
    f[++cnt] = now;
    rmq[cnt] = dep;
    p[now] = cnt;
    for(int i = 0;i < v[now].size();i++)
    {
        int t = v[now][i];
        if(t == pre)    continue;
        dfs(t,now,dep+1);
        f[++cnt] = now;
        rmq[cnt] = dep;
    }
}

void initlca(int root,int n)
{
    cnt = 0;
    dfs(root,root,0);
    st.init(2*n-1);
}

int querylca(int x,int y)
{
    return f[st.query(p[x],p[y])];
}

int build(int l,int r)
{
    int now = tot++;
    c[now] = 0;
    if(l == r)  return now;
    int mid = (l+r)/2;
    lson[now] = build(l,mid);
    rson[now] = build(mid+1,r);
    return now;
}

int update(int last,int pos,int x)
{
    int now = tot++,t = now;
    c[now] = c[last]+x;
    int l = 1,r = m;
    while(l < r)
    {
        int mid  = (l+r)/2;
        if(pos <= mid)
        {
            lson[now] = tot++;
            rson[now] = rson[last];
            now = lson[now];
            last = lson[last];
            r = mid;
        }
        else
        {
            rson[now] = tot++;
            lson[now] = lson[last];
            now = rson[now];
            last = rson[last];
            l = mid+1;
        }
        c[now] = c[last]+x;
    }
    return t;
```

```
}

void dfsbuild(int now,int pre)
{
    tree[now] = update(tree[pre],gethash(a[now]),1);
    for(int i = 0;i < v[now].size();i++)
    {
        int t = v[now][i];
        if(t == pre)    continue;
        dfsbuild(t,now);
    }
}

int query(int lt,int rt,int lcat,int pos,int k,int l,int r)
{
    if(l == r)  return l;
    int mid = (l+r)/2;
    int t = c[lson[lt]]+c[lson[rt]]-2*c[lson[lcat]];
    if(pos >= l && pos <= mid)  t++;
    if(t >= k)  return query(lson[lt],lson[rt],lson[lcat],pos,k,l,mid);
    return query(rson[lt],rson[rt],rson[lcat],pos,k-t,mid+1,r);
}

int main()
{
    scanf("%d%d",&n,&q);
    for(int i = 1;i <= n;i++)   scanf("%d",&a[i]);
    inithash();
    tot = 0;
    for(int i = 1;i < n;i++)
    {
        int x,y;
        scanf("%d%d",&x,&y);
        v[x].push_back(y);
        v[y].push_back(x);
    }
    initlca(1,n);
    tree[0] = build(1,m);
    dfsbuild(1,0);
    while(q--)
    {
        int x,y,k;
        scanf("%d%d%d",&x,&y,&k);
        int t = querylca(x,y);
        printf("%d\n",b[query(tree[x],tree[y],tree[t],gethash(a[t]),k,1,m)]);
    }
    return 0;
}
```

**动态第 k 大**

```
int n,q,m,tot,a[N],b[N],use[N];;
int tree[N],tree2[N],lson[N*50],rson[N*50],c[N*50];

struct Query
{
    int kind;
    int l,r,k;
}query[10010];

void Init_hash(int k)
{
    sort(b,b+k);
    m = unique(b,b+k) - b;
}
```

```
int hash(int x)
{
    return lower_bound(b,b+m,x)-b;
}
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);
}

void add(int x,int pos,int val)
{
    while(x <= n)
    {
        tree2[x] = Insert(tree2[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 = tree[left-1];
    int right_root = tree[right];
    int l = 0, r = m-1;
    for(int i = left-1;i;i -= lowbit(i)) use[i] = tree2[i];
    for(int i = right;i ;i -= lowbit(i)) use[i] = tree2[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)
    {
        tree2[x] = Insert(tree2[x],p,d);
        x += lowbit(x);
    }
}

int main()
{
    int T;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d%d",&n,&q);
        tot = 0;
        m = 0;
        for(int i = 1;i <= n;i++)
        {
            scanf("%d",&a[i]);
            b[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);
                b[m++] = query[i].r;
            }
        }
        Init_hash(m);
        tree[0] = build(0,m-1);
        for(int i = 1;i <= n;i++)
            tree[i] = Insert(tree[i-1],hash(a[i]),1);
        for(int i = 1;i <= n;i++)
            tree2[i] = tree[0];
        for(int i = 0;i < q;i++)
        {
            if(query[i].kind == 0)
                printf("%d\n",b[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;
}
```

# 动态规划

**最长公共子序列 LCS**

```
int dp[1005][1005] = {0},len1,len2;
char a[1005],b[1005];

void lcs()
{
    for(int i = 1; i <= len1;i++)
    {
        for(int j = 1;j <= len2;j++)
        {
            if(a[i-1] == b[j-1])    dp[i][j] = dp[i-1][j-1]+1;
            else    dp[i][j] = max(dp[i-1][j],dp[i][j-1]);
        }
    }
}

void printflcs(int x,int y)
{
    if(x == 0 || y == 0)    return;
    if(a[x-1] == b[y-1])
    {
        printflcs(x-1,y-1);
        printf("%c",a[x-1]);
    }
    else if(dp[x][y-1] > dp[x-1][y])    printflcs(x,y-1);
    else    printflcs(x-1,y);
}
```

**最长上升子序列 LIS**

```
int n,a[50005],b[50005] = {0};

int main()
{
    scanf("%d",&n);
    for(int i = 1;i <= n;i++)   scanf("%d",&a[i]);
    b[1] = a[1];
    int len = 1;
    for(int i = 2;i <= n;i++)
    {
        int t = lower_bound(b+1,b+len+1,a[i])-b;
        b[t] = a[i];
        if(t == len+1)  len++;
    }
    printf("%d\n",len);
    return 0;
}
```

**01 背包**

```
int n,m,w[3500],v[3500],dp[13000] = {0};
//数量 n，背包 m，重量 w，价值 v

int main()
{
    scanf("%d%d",&n,&m);
    for(int i = 1;i <= n;i++)    scanf("%d%d",&w[i],&v[i]);
    for(int i = 1;i <= n;i++)
    {
        for(int j = m;j >= w[i];j--)    dp[j] = max(dp[j],dp[j-w[i]]+v[i]);
    }
    printf("%d\n",dp[m]);
    return 0;
}
```

**01 背包(容量很大)**

```
int dp[50005],v[1005],w[1005],n,W;
//数量 n，背包 W，重量 w，价值 v
//W 很大，考虑每个价值的最小重量
int main()
{
    ios::sync_with_stdio(0);
    while(cin >> n >> W)
    {
        memset(dp, 0x3f, sizeof(dp));
        dp[0] = 0;
        int sum = 0;
        for(int i = 1;i <= n;i++)
        {
            cin >> w[i] >> v[i];
            sum += v[i];
        }
        for(int i = 1;i <= n;i++)
        {
            for(int j = sum;j >= v[i];j--)  dp[j] = min(dp[j-v[i]]+w[i],dp[j]);
        }
        for(int i = sum;i >= 0;i--)
        {
            if(dp[i] <= W)
            {
                cout << i << endl;
                break;
            }
        }
    }
    return 0;
}
```

**完全背包**

```
//1、如果背包要求正好装满则初始化 dp[0] = 0, dp[1~m] = INF。
//2、如果不需要正好装满 dp[0~m] = 0。
int n,m,e,f,w[505],v[505],dp[10005];
//数量 n，背包 m，重量 w，价值 v

int main()
{
    int T;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d%d%d",&e,&f,&n);
        m = f-e;
        memset(dp,0x3f,sizeof(dp));
        dp[0] = 0;
        for(int i = 1;i <= n;i++)    scanf("%d%d",&v[i],&w[i]);
        for(int i = 1;i <= n;i++)
        {
            for(int j = w[i];j <= m;j++)    dp[j] = min(dp[j],dp[j-w[i]]+v[i]);
        }
        if(dp[m] == INF)    printf("This is impossible.\n");
        else    printf("The minimum amount of money in the piggy-bank is %d.\n",dp[m]);
    }
    return 0;
}
```

**多重背包二进制**

```
int n,m,dp[105];

int main()
{
```

```
    int T;
    cin >> T;
    while(T--)
    {
        memset(dp,0,sizeof(dp));
        cin >> n >> m;
        for(int i = 1;i <= m;i++)
        {
            int p,h,c;    //花费，价值，数量
            cin >> p >> h >> c;
            int now = 1;
            while(1)
            {
                int t = min(now,c);
                c -= t;
                int pp = p*t,hh = h*t;
                for(int j = n;j >= pp;j--)  dp[j] = max(dp[j],dp[j-pp]+hh);
                if(c == 0)  break;
                now *= 2;
            }
        }
        cout << dp[n] << endl;
    }
    return 0;
}
```

**多重背包优先队列**

```
int n,m,dp[105],a[105],b[105];

int main()
{
    int T;
    cin >> T;
    while(T--)
    {
        memset(dp,0,sizeof(dp));
        cin >> n >> m;
        for(int i = 1;i <= m;i++)
        {
            int p,h,c;
            cin >> p >> h >> c;
            if(n/p < c) c = n/p;
            for(int d = 0;d < p;d++)
            {
                int l = 1,r = 0;
                for(int j = 0;j <= (n-d)/p;j++)
                {
                    while(l <= r && b[r] <= dp[j*p+d]-j*h)  r--;
                    a[++r] = j;
                    b[r] = dp[j*p+d]-j*h;
                    if(a[l] < j-c)  l++;
                    dp[j*p+d] = b[l]+j*h;
                }
            }
        }
        cout << dp[n] << endl;
    }
    return 0;
}
```

**数位 dp**

```
//求 1-n 中不包含 m 字段的数量
int a[25],b[25],x[25],cnt1,cnt2;
long long n,m,dp[25][25];
```

```
long long dfs(int now1,int now2,int limit)
{
    if(now2 == 0)   return 0;
    if(now1 == 0)   return 1;
    if(!limit && dp[now1][now2] != -1)  return dp[now1][now2];
    int endd = limit?a[now1]:9;
    long long ans = 0;
    for(int i = endd;i >= 0;i--)
    {
        int t = now2;
        while(b[t] != i && t != cnt2)   t = x[t];
        if(b[t] == i)   t--;
        ans += dfs(now1-1,t,limit && i == endd);
    }
    if(!limit)  dp[now1][now2] = ans;
    return ans;
}

int main()
{
    ios::sync_with_stdio(0);
    int T;
    cin >> T;
    while(T--)
    {
        cin >> n >> m;
        cnt1 = 0;
        cnt2 = 0;
        while(n)
        {
            a[++cnt1] = n%10;
            n /= 10;
        }
        while(m)
        {
            b[++cnt2] = m%10;
            m /= 10;
        }
        int i = cnt2,j = cnt2+1;
        x[cnt2] = cnt2+1;
        while(i >= 1)
        {
            if(j == cnt2+1 || b[i] == b[j])   x[--i] = --j;
            else    j = x[j];
        }
        x[cnt2] = cnt2;
        memset(dp,-1,sizeof(dp));
        cout << dfs(cnt1,cnt2,1)-1 << endl;
    }
    return 0;
}
```

**状压 dp**

```
//裁玻璃 2*2
int a[15][15],sta[100],dp[15][100],sum[15][100],n,m;
int main()
{
    ios::sync_with_stdio(false);
    int T;
    cin >> T;
    while(T--)
    {
        memset(dp,0,sizeof(dp));
        memset(sum,0,sizeof(sum));
```

```
        cin >> n >> m;
        for(int i = 1;i <= n;i++)
        {
            for(int j = 1;j <= m;j++)   cin >> a[i][j];
        }
        int cnt = 0,endd = 1<<(m-1);
        for(int i = 0;i < endd;i++)
        {
            if(i & (i<<1))  continue;
            sta[++cnt] = i;
        }
        for(int i = 1;i < n;i++)
        {
            for(int j = 1;j <= cnt;j++)
            {
                for(int k = 1;k <= m-1;k++)
                {
                    if(1<<(k-1) & sta[j])
                    {
                        if(a[i][k] && a[i][k+1] && a[i+1][k] && a[i+1][k+1])
sum[i][j]++;
                    }
                }
            }
        }
        for(int i = 1;i <= cnt;i++) dp[1][i] = sum[1][i];
        for(int i = 1;i < n;i++)
        {
            for(int j = 1;j <= cnt;j++)
            {
                for(int k = 1;k <= cnt;k++)
                {
                    dp[i+2][k] = max(dp[i+2][k],dp[i][j]+sum[i+2][k]);
                    if(sta[j] & sta[k])          continue;
                    if(sta[j] & (sta[k]<<1))     continue;
                    if(sta[j] & (sta[k]>>1))     continue;
                    dp[i+1][k] = max(dp[i+1][k],dp[i][j]+sum[i+1][k]);
                }
            }
        }
        int ans = 0;
        for(int i = 1;i <= cnt;i++) ans = max(ans,dp[n-1][i]);
        cout << ans << endl;
    }
    return 0;
}
```

# 数论

**素数筛**

```
int cnt,prime[MAXN+1],mi[MAXN+1],vis[MAXN+1] = {0};
//cnt 表示素数个数
//prime 存放每个素数
//mi 存放每个数的最小素数因子

void getprime()
{
    cnt = 0;
    memset(prime,0,sizeof(prime));
    for(int i = 2;i <= MAXN;i++)
    {
        if(!vis[i])
        {
            prime[++cnt] = i;
            mi[i] = i;
        }
        for(int j = 1;j <= cnt && (long long)i*prime[j] <= MAXN;j++)
        {
            vis[prime[j]*i] = 1;
            mi[prime[j]*i] = prime[j];
            if(!(i%prime[j]))   break;
        }
    }
}

int cnt2,notprime2[1000005],prime2[1000005];
//区间 l,r 之间的素数
void getprime2(int l,int r)
{
    cnt2 = 0;
    memset(notprime2,0,sizeof(notprime2));
    if(l < 2)   l = 2;
    for(int i = 1;i <= cnt && (long long)prime[i]*prime[i] <= r;i++)
    {
        int t = l/prime[i];
        if(l%prime[i])  t++;
        if(t == 1)  t = 2;
        for(int j = t;(long long)j*prime[i] <= r;j++)
        {
            if((long long)j*prime[i] >= l)  notprime2[j*prime[i]-l] = 1;
        }
    }
    for(int i = 0;i <= r-l;i++)
    {
        if(!notprime2[i])   prime2[++cnt2] = i+l;
    }
}
```

**合数分解**

```
long long factor[105][2];
//返回素因子个数
//factor[i][0]表示第 i 个素因子，factor[i][1]表示第 i 个素因子的个数
int getfac(long long x)
{
    int cnt = 0;
    for(int i = 1;(long long)prime[i]*prime[i] <= x;i++)
    {
        if(x%prime[i] == 0)
        {
            factor[++cnt][0] = prime[i];
            while(x%prime[i] == 0)
```

```
            {
                factor[cnt][1]++;
                x /= prime[i];
            }
        }
    }
    if(x != 1)
    {
        factor[++cnt][0] = x;
        factor[cnt][1] = 1;
    }
    return cnt;
}
```

**快速乘**

```
long long qmul(long long a,long long b,long long c)
{
    long long ans = 0;
    a = a%c;
    b = b%c;
    while(b)
    {
        if(b%2)
        {
            ans += a;
            if(ans > c) ans -= c;
        }
        a = a+a;
        if(a > c)   a -= c;
        b /= 2;
    }
    return ans;
}
```

**快速幂**

```
long long qpower(long long a, long long b, long long c)
{
    long long ans = 1;
    a = a%c;
    while(b)
    {
        if(b%2)    ans = ans*a%c;
        a = a*a%c;
        b /= 2;
    }
    return ans;
}
```

**string 快速幂**

```
string mul(string a,string b)
{
    int arr[200],len = a.length()+b.length();
    memset(arr,0,sizeof(arr));
    reverse(a.begin(),a.end());
    reverse(b.begin(),b.end());
    for(int i = 0;i < a.length();i++)
    {
        for(int j = 0;j < b.length();j++)   arr[i+j] += (a[i]-'0')*(b[j]-'0');
    }
    for(int i = 0;i < len;i++)
    {
        arr[i+1] += arr[i]/10;
        arr[i] %= 10;
    }
    string ans = string(len,'0');
    for(int i = 0;i < len;i++)  ans[i] += arr[i];
```

```
    reverse(ans.begin(),ans.end());
    cout << ans << endl;
    return ans;
}

string strpow(string x,int b)
{
    string ans = "1";
    while(b)
    {
        if(b%2) ans = mul(ans,x);
        x = mul(x,x);
        b /= 2;
    }
    return ans;
}
```

**矩阵快速幂**

```
struct matrix
{
    long long m[2][2];
};

matrix one = {  1,0,
                0,1 };

matrix mul(matrix a, matrix b)
{
    matrix tmp;
    for(int i = 0; i < 2;i++)
    {
        for(int j = 0; j < 2;j++)
        {
            tmp.m[i][j] = 0;
            for(int k = 0; k < 2;k++)    tmp.m[i][j] =
(tmp.m[i][j]+a.m[i][k]*b.m[k][j])%MOD;
        }
    }
    return tmp;
}

matrix maqower(matrix a,int b)
{
    matrix ans = one;
    while(b)
    {
        if(b%2) ans = mul(ans,a);
        a = mul(a,a);
        b /= 2;
    }
    return ans;
}
```

**miller_rabin 判断素数**

```
//用 a 来检验 n 是否为素数
//是素数返回 1，不是返回 0
bool check(long long a,long long n,long long x,long long t)
{
    long long ans = qpower(a,x,n);
    if(ans == 1 || ans == n-1)  return 1;
    while(t--)
    {
        ans = qmul(ans,ans,n);
        if(ans == n-1)  return 1;
    }
```

```
    return 0;
}

//判断 n 是否为素数
//是素数返回 1，不是返回 0
bool miller_rabin(long long n)
{
    if(n < 2)   return 0;
    if(n == 2)  return 1;
    if(n%2 == 0)    return 0;
    long long x = n-1;
    long long t = 0;
    while(x%2 == 0)
    {
        x /= 2;
        t++;
    }
    srand(time(0));
    for(int i = 1;i <= 10;i++)
    {
        long long a = rand()%(n-1)+1;
        if(!check(a,n,x,t))  return 0;
    }
    return 1;
}
```

**pollard_rho 分解质因数**

```
long long factor[105];
int cnt;

long long pollard_rho(long long n,long long c)
{
    long long i = 1,k = 2;
    srand(time(0));
    long long x = rand()%(n-2)+1,y = x;
    while(1)
    {
        i++;
        x = (qmul(x,x,n)+c)%n;
        long long d = __gcd(y-x,n);
        if(1 < d && d < n)  return d;
        if(x == y)  return n;
        if(i == k)
        {
            y = x;
            k *= 2;
        }
    }
}

//求 n 的质因数，k 设为 107
void findfac(long long n,int k)
{
    if(n == 1)  return;
    if(miller_rabin(n))
    {
        factor[++cnt] = n;
        return;
    }
    long long p = n;
    int c = k;
    while(p >= n)   p = pollard_rho(p,c--);
    findfac(p,k);
    findfac(n/p,k);
```

```
}
```

**gcd、lcm**

```
LL gcd(LL x,LL y)        //最大公约数
{
    return y?gcd(y,x%y):x;
}

LL lcm(LL x,LL y)        //最小公倍数
{
    return x*y/gcd(x,y);
}
```

**lcm(1,2,3,…n)**

```
//lcm(C(n,0), C(n,1),…, C(n,n)) = lcm(1,2,3,…n+1)/(n+1)
//计算 lcm(1,2,3,…n)
const int N=100000007;
int visit[N/32+50];
unsigned int data[5800000];
int prime[5800000],np=0;

void Prime()   //筛素数,数组从 0 开始
{
    prime[0] = data[0] = 2;
    np = 1;
    for(int i = 3;i < N;i += 2)
    {
        if(!(visit[i/32]&(1<<((i/2)%16))))
        {
            prime[np] = i;
            data[np] = data[np-1]*i;
            np++;
            for(int j = 3*i;j < N;j += 2*i) visit[j/32] |= (1<<((j/2)%16));
        }
    }
}

long long Deal(int n)
{
    int p = upper_bound(prime,prime+np,n)-prime-1;
    long long ans = data[p];
    for (int i = 0;i < np && prime[i]*prime[i] <= n;i++)
    {
        int mul = prime[i],tmp = prime[i] * prime[i];  ;
        while (tmp/mul == prime[i] && tmp <= n)
        {
            tmp *= prime[i];
            mul *= prime[i];
        }
        ans = ans*(mul/prime[i])%MOD;
    }
    return ans;
}
```

**欧拉函数**

小于 N 且与 N 互质的数的个数(包括 1)。

对于一个正整数 N 的素数幂分解 N = P1^q1*P2^q2*...*Pn^qn。

φ(N)=N*(1-1/P1)*(1-1/P2)*...*(1-1/Pn)。

```
//先分解质因数
int euler(int n)
{
    int ans = n;
    for(int i = 1;i <= cnt;i++) ans = ans/factor[i][0]*(factor[i][0]-1);
```

```
}

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

**线性筛**

```
int cnt,phi[N+10],prime[N+10],vis[N+10];
//同时得到欧拉函数和素数表
void geteuler()
{
    memset(vis,0,sizeof(vis));
    phi[1] = 1;
    cnt = 0;
    for(int i = 2;i <= N;i++)
    {
        if(!vis[i])
        {
            prime[++cnt] = i;
            phi[i] = i-1;
        }
        for(int j = 1;j <= cnt;j++)
        {
            if((long long)i*prime[j] > N)   break;
            vis[i*prime[j]] = 1;
            if(i%prime[j] == 0)
            {
                phi[i*prime[j]] = phi[i]*prime[j];
                break;
            }
            else    phi[i*prime[j]] = phi[i]*(prime[j]-1);
        }
    }
}
```

**lucas 定理**

A、B 是非负整数，p 是质数。AB 写成 p 进制：A=a[n]a[n-1]...a[0]，B=b[n]b[n-1]...b[0]。
则组合数 C(A,B)与 C(a[n],b[n])*C(a[n-1],b[n-1])*...*C(a[0],b[0]) modp 同余。

即：Lucas(n,m,p)=c(n%p,m%p)*Lucas(n/p,m/p,p) 。

```
//用与 n,m 很大，MOD 不是很大的情况
long long qmod(long long a,long long b,long long c)
{
    long long ans = 1;
    a = a%c;
    while(b)
    {
        if(b%2)  ans = (ans*a)%c;
```

```
        a = (a*a)%c;
        b /= 2;
    }
    return ans;
}

long long c(long long m,long long n)
{
    if(m < n)   return 0;
    if(m == n)  return 1;
    if(n > m-n) n = m-n;
    long long mm = 1,nn = 1;
    for(long long i = 0;i < n;i++)
    {
        mm = mm*(m-i)%MOD;
        nn = nn*(n-i)%MOD;
    }
    return mm*qmod(nn,MOD-2,MOD)%MOD;
}

long long lucas(long long m,long long n)
{
    long long ans = 1;
    while(m && n && ans)
    {
        ans = ans%MOD*c(m%MOD,n%MOD)%MOD;
        n /= MOD;
        m /= MOD;
    }
    return ans;
}
```

**扩展 lucas(模合数)**

```
long long inv(long long a,long long MOD)
{
    if(!a)  return 0;
    long long b = MOD,x = 0,y = 0;
    e_gcd(a,b,x,y);
    x = ((x%b)+b)%b;
    if(!x)  x += b;
    return x;
}

long long mul(long long n,long long pi,long long pk)
{
    if(!n)  return 1;
    long long ans=1;
    for(long long i = 2;i <= pk;i++)
    {
        if(i%pi)    ans = ans*i%pk;
    }
    ans = qpower(ans,n/pk,pk);
    for(long long i = 2;i <= n%pk;i++)
    {
        if(i%pi)    ans = ans*i%pk;
    }
    return ans*mul(n/pi,pi,pk)%pk;
}

long long C(long long n,long long m,long long MOD,long long pi,long long pk)
{
    if(m > n)   return 0;
    long long a = mul(n,pi,pk),b = mul(m,pi,pk),c = mul(n-m,pi,pk),k = 0,ans = 0;
    for(long long i = n;i;i /= pi)  k += i/pi;
```

```
    for(long long i = m;i;i /= pi)  k -= i/pi;
    for(long long i = n-m;i;i /= pi)    k -= i/pi;
    ans = a*inv(b,pk)%pk*inv(c,pk)%pk*qpower(pi,k,pk)%pk;
    return ans*(MOD/pk)%MOD*inv(MOD/pk,pk)%MOD;
}

long long lucas(long long n,long long m,long long MOD)
{
    long long ans = 0,x = MOD;
    for(long long i = 2;i*i <= x;i++)
    {
        if(x%i == 0)
        {
            long long t = 1;
            while(x%i == 0)
            {
                t *= i;
                x /= i;
            }
            ans = (ans+C(n,m,MOD,i,t))%MOD;
        }
    }
    if(x > 1)   ans = (ans+C(n,m,MOD,x,x))%MOD;
    return ans;
}
```

**逆元**

费马小定理：假如 p 是素数，且 a 与 p 互质，那么 a^(p-1) = 1 (mod p)。

费马大定理：当整数 n>2 时，关于 x,y,z 的方程 x^n+y^n=z^n 没有正整数解。

欧拉定理：若 n,a 为正整数，且 n,a 互质，则:a^(phi(n)) = 1(mod n)，可得 a^b%p = a^(b%phi(p))%p

另外，a,b 不互质且 b<phi(n)时，有 a^b%p = a^(b%phi(p)+phi(p))%p

逆元：ax = 1(mod m)的 x 值(0 < a < m)。

当 p 是质数的时候 a/x mod p == a*x^(p-2) mod p 。

当 p 不是质数的时候 a/x mod p == a*x^(phi(p)-1) mod p。

**逆元 1**

```
long long inv(long long a,long long m)
{
    return qmod(a,m-2,m);
}
```

**逆元 2**

```
long long inv(long long a,long long m)
{
    if(a == 1)  return 1;
    return inv(m%a,m)*(m-m/a)%m;
}
```

**逆元 3**

```
//扩展欧几里德
long long e_gcd(long long a,long long b,long long &x,long long &y)
{
    if(!b)
    {
        x = 1;
        y = 0;
        return a;
    }
    long long d = e_gcd(b,a%b,y,x);
```

```
    y -= a/b*x;
    return d;
}

long long inv(long long a,long long m)
{
    long long x,y;
    long long gcd = e_gcd(a,m,x,y);
    if(gcd == 1)    return (x%m+m)%m;
    return -1;
}
```

**中国剩余定理**

```
//求解方程组 x%m = a,x 的最小值
//无解返回-1
long long e_gcd(long long a,long long b,long long &x,long long &y)
{
    if(!b)
    {
        x = 1;
        y = 0;
        return a;
    }
    long long d = e_gcd(b,a%b,y,x);
    y -= a/b*x;
    return d;
}

long long solve(long long *m,long long *a,long long n)
{
    long long M = m[1],A = a[1],x,y;
    for(int i = 2;i <= n;i++)
    {
        long long d = e_gcd(M,m[i],x,y);
        if((a[i]-A)%d)  return -1;
        x = (a[i]-A)/d*x%(m[i]/d);
        A += x*M;
        M = M/d*m[i];
        A %= M;
    }
    if(A < 0)   A += M;
    return A;
}
```

**浮点数高斯消元**

```
//浮点型只有唯一解时可计算
//返回 0 表示无解
#define eps 1e-9
double a[N][N],x[N];        //左边矩阵和右边值，结果存在 x 数组内
int Gauss(int equ,int var)  //方程个数和未知数个数
{
    for(int row = 0,col = 0;col < var&&row < equ;col++,row++)
    {
        int max_r=row;
        for(int i = row+1;i < equ;i++)
        {
            if(fabs(a[i][col])-fabs(a[max_r][col]) > eps)   max_r=i;
        }
        if(fabs(a[max_r][col]) < eps)   return 0;
        if(max_r != row)
        {
            for(int j = 0;j <= var;j++) swap(a[row][j],a[max_r][j]);
        }
        for(int i = row+1;i < equ;i++)
        {
```

```
            if(fabs(a[i][col]) < eps)    continue;
            double t = -a[i][col]/a[row][col];
            for(int j = col;j <= var;j++)    a[i][j] += t*a[row][j];
        }
    }
    for(int i = var-1;i >= 0;i--)    //计算唯一解。
    {
        double t = 0;
        for(int j = i+1;j < var;j++)    t += a[i][j]*x[j];
        x[i] = (a[i][var]-t)/a[i][i];
    }
    return 1;
}
```

**01 异或高斯消元**

```
//有 equ 个方程，var 个变元。增广矩阵行数为 equ,分别为 0 到 equ-1,列数为 var+1,分别为 0 到
var.
int equ,var;
int a[250][250];    //增广矩阵
int x[250];         //解集
int free_x[250];    //标记是否不确定的变元
int free_num;       //不确定变元个数
int n;
char s[30];
//-1 表示无解，0 表示唯一解，大于 0 表示无穷解，并返回自由变元的个数)
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 Debug()
{
    for(int i = 0;i < equ;i++)
    {
        for(int j = 0;j < var+1;j++)  cout << a[i][j] << " ";
        cout << endl;
    }
    cout << endl;
}

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(max_r != k)
        {
            for(int i = col;i < var+1;i++)  swap(a[k][i],a[max_r][i]);
        }
        if(a[k][col] == 0)
        {
            k--;
            free_x[free_num++] = col;
            continue;
        }
        for(int i = k+1;i < equ;i++)
        {
            if(a[i][col] == 0)  continue;
            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;
    }
    return var-k;
}

void solve()
{
    int t = Gauss();
    if(t == -1) printf("inf\n");
    else
    {
        int ans = INT_MAX,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 t = a[j][var];
                for(int k = j+1;k < var;k++)
                {
                    if(a[j][k]) t ^= x[k];
                }
                x[j] = t;
                cnt += x[j];
            }
            ans = min(ans,cnt);
        }
        printf("%d\n",ans);
    }
}
```

**同余方程组高斯消元**

```
int a[MAXN][MAXN];  //增广矩阵
int x[MAXN];        //解集
```

```
bool free_x[MAXN];  //标记是否是不确定的变元

int gcd(int a,int b)
{
    return b?gcd(b,a%b):a;
}
int lcm(int a,int b)
{
    return a/gcd(a,b)*b;
}

//-1 表示无解，0 表示唯一解，大于 0 表示无穷解，并返回自由变元的个数)
//有 equ 个方程，var 个变元。增广矩阵行数为 equ,分别为 0 到 equ-1,列数为 var+1,分别为 0 到 var.
int Gauss(int equ,int var)
{
    int i,j,k;
    int max_r;// 当前这列绝对值最大的行.
    int col;//当前处理的列
    int ta,tb;
    int LCM;
    int temp;
    int free_x_num;
    int free_index;
    for(int i = 0;i <= var;i++)
    {
        x[i]=0;
        free_x[i]=true;
    }
    col=0;
    for(k = 0;k < equ && col < var;k++,col++)
    {
        max_r=k;
        for(i = k+1;i < equ;i++)
        {
            if(abs(a[i][col]) > abs(a[max_r][col])) max_r=i;
        }
        if(max_r != k)
        {
            for(j = k;j < var+1;j++)    swap(a[k][j],a[max_r][j]);
        }
        if(a[k][col] == 0)
        {
            k--;
            continue;
        }
        for(i = k+1;i < equ;i++)
        {
            if(a[i][col] != 0)
            {
                LCM = lcm(abs(a[i][col]),abs(a[k][col]));
                ta = LCM/abs(a[i][col]);
                tb = LCM/abs(a[k][col]);
                if(a[i][col]*a[k][col] < 0) tb = -tb;
                for(j = col;j < var+1;j++)
                {
                    a[i][j] = ((a[i][j]*ta-a[k][j]*tb)%7+7)%7;
                }
            }
        }
    }
    for(i = k; i < equ; i++)
    {
        if (a[i][col] != 0) return -1;
```

```
    }
    if(k < var)
    {
        for(i = k-1;i >= 0;i--)
        {
            free_x_num = 0;
            for(j = 0;j < var;j++)
            {
                if (a[i][j] != 0 && free_x[j])  free_x_num++,free_index = j;
            }
            if(free_x_num > 1)  continue;
            temp = a[i][var];
            for(j = 0;j < var;j++)
            {
                if(a[i][j] != 0 && j != free_index) temp -= a[i][j]*x[j]%7;
                temp = (temp%7+7)%7;
            }
            x[free_index] = (temp/a[i][free_index])%7;
            free_x[free_index] = 0;
        }
        return var-k;
    }
    for(i = var-1;i >= 0;i--)
    {
        temp = a[i][var];
        for (j = i + 1;j < var;j++)
        {
            if(a[i][j] != 0)    temp -= a[i][j]*x[j];
            temp = (temp%7+7)%7;
        }
        while(temp%a[i][i] != 0)    temp += 7;
        x[i] =(temp/a[i][i])%7;
    }
    return 0;
}
```

**FFT 递归**

```
typedef complex<double> C;
int n,m;
C a[N],b[N];

int gi()
{
    int res = 0,fh = 1;
    char ch = getchar();
    while((ch > '9' || ch < '0') && ch != '-')  ch = getchar();
    if(ch == '-')
    {
        fh=-1;
        ch=getchar();
    }
    while(ch >= '0' && ch <= '9')
    {
        res = res*10+ch-'0';
        ch=getchar();
    }
    return fh*res;
}

void fft(C *a,int n,int f)
{
    if(n == 1)  return;
    C wn(cos(2.0*PI/n),sin(f*2.0*PI/n)),w(1,0),t,a0[n>>1],a1[n>>1];
    for(int i = 0;i < n>>1;i++)
```

```
    {
        a0[i] = a[i<<1];
        a1[i] = a[i<<1|1];
    }
    fft(a0,n>>1,f);
    fft(a1,n>>1,f);
    for(int i = 0;i < n>>1;i++,w *= wn)
    {
        t = w*a1[i];
        a[i] = a0[i]+t;
        a[i+(n>>1)] = a0[i]-t;
    }
}
int main()
{
    n = gi();
    m = gi();
    for(int i = 0;i <= n;i++)   a[i]=gi();
    for(int i = 0;i <= m;i++)   b[i]=gi();
    m += n;
    for(n = 1;n <= m;n *= 2);
    fft(a,n,1);
    fft(b,n,1);
    for(int i = 0;i <= n;i++)   a[i] *= b[i];
    fft(a,n,-1);
    for(int i = 0;i <= m;i++)   printf("%d ",int(a[i].real()/n+0.5));
    return 0;
}
```

**FFT 非递归**

```
typedef complex<double> C;
int n,m,L,R[N];
C a[N],b[N];

int gi()
{
    int res = 0,fh = 1;
    char ch = getchar();
    while((ch > '9' || ch < '0') && ch != '-')  ch = getchar();
    if(ch == '-')
    {
        fh=-1;
        ch=getchar();
    }
    while(ch >= '0' && ch <= '9')
    {
        res = res*10+ch-'0';
        ch=getchar();
    }
    return fh*res;
}


void fft(C *a,int f)
{
    for(int i = 0;i < n;i++)
    {
        if(i < R[i])    swap(a[i],a[R[i]]);
    }
    for(int i = 1;i < n;i *= 2)
    {
        C wn(cos(PI/i),sin(f*PI/i)),x,y;
        for(int j = 0;j < n;j += i<<1)
        {
```

```
            C w(1,0);
            for(int k = 0;k < i;k++,w *= wn)
            {
                x = a[j+k];
                y = w*a[j+i+k];
                a[j+k] = x+y;
                a[j+i+k] = x-y;
            }
        }
    }
}

int main()
{
    n = gi();
    m = gi();
    for(int i = 0;i <= n;i++)   a[i] = gi();
    for(int i = 0;i <= m;i++)   b[i] = gi();
    m += n;
    for(n = 1;n <= m;n *= 2)    L++;
    for(int i = 0;i < n;i++)    R[i] = (R[i>>1]>>1)|((i&1)<<(L-1));
    fft(a,1);
    fft(b,1);
    for(int i = 0;i <= n;i++)   a[i] *= b[i];
    fft(a,-1);
    for(int i = 0;i <= m;i++)   printf("%d ",int(a[i].real()/n+0.5));
    return 0;
}
```

**约数和定理**

```
//需要 getprime,qpower,getfac,先调用 getprime
//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%2) return (1+qpower(p,n/2+1,MOD))%MOD*sum(p,n/2)%MOD;
    else    return ((1+qpower(p,n/2+1,MOD))%MOD*sum(p,n/2-
1)+qpower(p,n/2,MOD)%MOD)%MOD;
}

//求 a^b 的约数和对 MOD 取模
long long solve(long long a,long long b)
{
    int cnt = getfac(a);
    long long ans = 1;
    for(int i = 1;i <= cnt;i++)
    {
        ans *= sum(factor[i][0],b*factor[i][1])%MOD;
        ans %= MOD;
    }
    return ans;
}
```

**莫比乌斯反演**

$$F(n)=\sum_{d|n}f(d) => f(n)=\sum_{d|n}\mu(d)F(\frac{n}{d})$$

$$F(n)=\sum_{n|d}f(d) => f(n)=\sum_{n|d}\mu(\frac{d}{n})F(d)$$

```
const int MAXN = 100000;
bool check[MAXN+5];
```

```
int prime[MAXN+5],mu[MAXN+5],sum[MAXN+5];

//求莫比乌斯函数
void Moblus()
{
    memset(check,0,sizeof(check));
    mu[1] = 1;
    int cnt = 0;
    for(int i = 2;i <= MAXN;i++)
    {
        if(!check[i])
        {
            prime[cnt++] = i;
            mu[i] = -1;
        }
        for(int j = 0;j < cnt && i*prime[j] <= MAXN;j++)
        {
            check[i*prime[j]] = 1;
            if(i%prime[j] == 0)
            {
                mu[i*prime[j]] = 0;
                break;
            }
            else    mu[i*prime[j]] = -mu[i];
        }
    }
    sum[0] = 0;
    for(int i = 1;i <= MAXN;i++)    sum[i] = sum[i-1]+mu[i];
}

//求[1,n],[1,m]内互质的数的对数
int solve(int n,int m)
{
    int ans = 0;
    if(n > m)   swap(n,m);
    for(int i = 1,last;i <= n;i = last+1)
    {
        last = min(n/(n/i),m/(m/i));
        ans += (sum[last]-sum[i-1])*(n/i)*(m/i);
    }
    return ans;
}

int main()
{
    Moblus();
    int a,b,c,d,k,T;
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d%d%d%d%d",&a,&b,&c,&d,&k);
        a--;
        c--;
        a /= k;
        b /= k;
        c /= k;
        d /= k;
        printf("%d\n",solve(b,d)-solve(a,d)-solve(b,c)+solve(a,c));
    }

}
```

**BSGS**

```
int Inval(int a,int b,int n)
{
    long long x, y, e;
    e_gcd(a,n,x,y);
    e = (long long)x*b%n;
    return e < 0?e+n:e;
}

//求 A^x = B(mod C)(C 可以为非素数)
int BSGS(int A,int B,int C)
{
    map<int,int> H;
    long long buf = 1%C,D = buf,K;
    int d = 0,tmp;
    for (int i = 0;i <= 100;buf = buf*A%C,i++)
    if(buf == B)    return i;
    while((tmp = gcd(A,C)) != 1)
    {
        if(B % gcd(A,C)!= 0)    return -1;
        d++;
        C /= tmp;
        B /= tmp;
        D =D*A/tmp%C;
    }
    H.clear();
    int M = (int)ceil(sqrt(C));
    buf = 1%C;
    for(int i = 0;i <= M;buf = buf*A%C,i++)
    {
        if(H.find((int)buf) == H.end()) H[(int)buf] = i;
    }
    K = qpower(A,M,C);
    for(int i = 0;i <= M;D = D*K%C,i++)
    {
        tmp = Inval((int)D,B,C);
        if(tmp >= 0 && H.find(tmp) != H.end())  return i*M+H[tmp]+d;
    }
    return -1;
}
```

**自适应 simpson 积分**

```
const double eps = 1e-6;
// 三点 simpson 法。这里要求 F 是一个全局函数
double simpson(double a,double b)
{
    double c = a+(b-a)/2;
    return (F(a)+4*F(c)+F(b))*(b-a)/6;
}
// 自适应 Simpson 公式（递归过程）。已知整个区间[a,b]上的三点 simpson 值 A
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(A-L-R) <= 15*eps)   return L+R+(A-L-R)/15;
    return asr(a,c,eps/2,L)+asr(c,b,eps/2,R);
}
// 自适应 Simpson 公式（主过程）
double asr(double a,double b,double eps)
{
    return asr(a,b,eps,simpson(a,b));
}
```

## 康托展开

X = an*(n-1)!+an-1*(n-2)!+...+ai*(i-1)!+...+a2*1!+a1*0!

其中，ai 为当前未出现的元素中是排在第几个(从 0 开始)。

```
int f(vector<int> v)
{
    int sum = 0;
    for(int i = 0;i < 8;i++)
    {
        int t = 0;
        for(int j = i+1;j < 8;j++)
        {
            if(v[i] > v[j]) t++;
        }
        sum += t*h[7-i];
    }
    return sum;
}
```

## sg 函数

```
//f[]：可以取走的石子个数
//sg[]:0~n 的 SG 函数值
//hash[]:mex{}
int n,m,p,f[N],sg[N],hashh[N];

void getSG(int n)
{
    memset(sg,0,sizeof(sg));
    for(int i = 1;i <= n;i++)
    {
        memset(hashh,0,sizeof(hashh));
        for(int j = 1;f[j] <= i;j++)    hashh[sg[i-f[j]]] = 1;
        for(int j = 0;j <= n;j++)        //求 mes{}中未出现的最小的非负整数
        {
            if(hashh[j] == 0)
            {
                sg[i] = j;
                break;
            }
        }
    }
}
```

## 整数划分分类

1. 若划分的多个整数可以相同

设 dp[i][j]为将 i 划分为不大于 j 的划分数。

(1) 当 i<j 时，i 不能划分为大于 i 的数，dp[i][j]=dp[i][i]。

(2) 当 i>j 时，可以根据划分中是否含有 j 分为两种情况。

若划分中含有 j，划分方案数为 dp[i-j][j]。

若划分数中不含 j，相当于将 i 划分为不大于 j-1 的划分数，为 dp[i][j-1]。

所以当 i>j 时，dp[i][j]=dp[i-j][j]+dp[i][j-1]。

(3) 当 i=j 时，若划分中含有 j 只有一种情况，若划分中不含 j 相当于将 i 划分为不大于 j-1 的划分数。此时 dp[i][j]=1+dp[i][j-1]。

2.将 n 划分为 k 个整数的划分数

dp[i][j]表示将 i 划分成 j 个正整数的划分数。

(1) 当 i<j 时，i 不能划分成 j 个正整数，dp[i][j]=0。

(2) 当 i=j 时，只有一种情况，dp[i][j]=1。

(3) 当 i>j 时， 此时根据含 1 和不含 1，故 dp[i][j]=dp[i-1][j-1]+dp[i-j][j]。

3. 若划分的正整数必须不同

设 dp[i][j]为将 i 划分为不超过 j 的不同整数的划分数。

(1) 当 i<j 时，i 不能划分为大于 i 的数，所以 dp[i][j]=dp[i][i]。

(2) 当 i>j 时，可以根据划分中是否含有 j 分为两种情况。

若划分中含有 j，则其余的划分中最大只能是 j-1，方案数为 dp[i-j][j-1]。

若划分中不含 j，相当于将 i 划分为不大于 j-1 的划分数，为 dp[i][j-1]。

所以当 i>j 时 dp[i][j]=dp[i-j][j-1]+dp[i][j-1]。

(3) 当 i=j 时，若划分中含有 j 只有一种情况，若划分中不含 j 相当于将 i 划分为不大于 j-1 的划分数。此时 dp[i][j]=1+dp[i][j-1]。

4.将 n 划分为若干正奇数之和的划分数

设 f[i][j]为将 i 划分为 j 个奇数之和的划分数，g[i][j]为将 i 划分为 j 个偶数之和的划分数。

使用截边法，将 g[i][j]的 j 个划分都去掉 1，可得 g[i][j] = f[i-j][j]。

f[i][j]中有包含 1 的划分方案和不包含 1 的划分方案。

对于包含 1 的划分方案，可以将 1 的划分除去，转化为"将 i-1 划分为 j-1 个奇数之和的划分数"，即 f[i-1][j-1]。

对于不包含 1 的划分方案，可以使用截边法对 j 个划分每一个都去掉一个 1，转化为"将 i-j 划分为 j 个偶数之和的划分数"，即 g[i-j][j]。

所以 f[i][j]=f[i-1][j-1]+g[i-j][j]。

f[n][0]+f[n][1]+……+f[n][n]为将 n 划分为若干奇数的划分数。

## Polya 定理

设 G = {a1，a2，…，ag}是 N 个对象的置换群，用 M 种颜色给这 N 个对象着色，

则不同的着色 方案数为：|G|^(-1) * {M^c(a1) + M^c(a2) + … + M^c(ag)}。其中 c(ai)为置换 ai 的循环节数，( i = 1，2，…，g )。

## 原根

设 m 是正整数，a 是整数，若 a 模 m 的阶等于 φ(m)，则称 a 为模 m 的一个原根。

如果 g 是 P 的原根，那么 g 的（1...P-1）次幂 mod P 的结果一定互不相同。

如果 g 是 P 的原根，就是 g^(P-1) = 1 (mod P)当且仅当指数为 P-1 的时候成立.(这里 P 是素数).

```
int n,a[35000],cnt = 0;

int main()
{
    scanf("%d",&n);
    int endd = sqrt(n-1);
    for(int i = 2;i <= endd;i++)
    {
        if((n-1)%i == 0)    a[++cnt] = i;
    }
    for(int i = 2;i <= n-1;i++)
    {
        int j;
```

```
        for(j = 1;j <= cnt;j++)
        {
            if(qmod(i,a[j],n) == 1 || qmod(i,(n-1)/a[j],n) == 1)    break;
        }
        if(j == cnt+1)
        {
            printf("%d\n",i);
            return 0;
        }
    }
}
```

**卡特兰数**

1, 2, 5, 14, 42

h(0) = 1,h(1) = 1

h(n) = h(0)*h(n-1)+h(1)*h(n-2) + ... + h(n-1)h(0) (n>=2)

h(n) = h(n-1)*(4*n-2)/(n+1)

h(n) = C(2n,n)/(n+1)

h(n) = C(2n,n)-C(2n,n-1)

1.n 对括号正确匹配组成的字符串数。

2.一个栈(无穷大)的进栈序列为 1，2，3，…，n，有多少个不同的出栈序列。

3.在一个凸多边形中，通过若干条互不相交的对角线，把这个多边形划分成了若干个三角形,求不同划分的方案数 f（n）。

4.一位大城市的律师在她住所以北 n 个街区和以东 n 个街区处工作。

5.长度为 2n 的 Dyck words 的数量。 Dyck words 是由 n 个 X 和 n 个 Y 组成的字符串，并且从左往右数， Y 的数量不超过 X。

6.拥有 n+1 个叶子节点的二叉树的数量。例如 4 个叶子节点的所有二叉树形态。

7.圆桌握手问题： 圆桌周围有 2n 个人，他们两两握手，但没有交叉的方案数。

---

**调和级数**

f(n)≈ln(n)+C+1/2*n （n 很大时）

C≈0.57721566490153286060651209

# 数学结论

**排列组合**

$$A_n^m = n(n-1)\cdots(n-m+1) = \frac{n!}{(n-m)!}$$

$$C_n^m = \frac{A_n^m}{m!} = \frac{n!}{m!(n-m)!} = C_n^{n-m}$$

圆排列 $\frac{P_n^m}{m} = \frac{n!}{(n-m)! \times m}, (1 \le m \le n)$

有限多重集的排列 n！/(n1!*n2!*...*nk!)

n 元无限集可重-r 组合 C(n+r-1,r)种。

n 元无限集取 r 个,n 中每个至少出现一次 C(r-1,n-1)(r≥n)

**直线分平面**

f(1) = 2

f(n) = f(n-1)+n = n(n+1)/2+1

**折线分平面**

f(1) = 2

f(n) = f(n-1)+4(n-1)+1 = 2n^2-n+1

**圆分平面**

f(1) = 2

f(n) = f(n-1)+2(n-1) = n^2-n+2

**三角形分平面**

f(1) = 2

f(n) = f(n-1)+6*(n-1)

**平面分空间**

f(1) = 2

g(n) = n(n+1)/2+1

f(n) = f(n-1)+g(n-1) = (n^3+5n)/6+1

**求 Ax+By+C**

已知 p1(x1,x1),p2(x2,y2)，求 Ax+By+C = 0

A = y2 - y1

B = x1 - x2

C = x2*y1-x1*y2

**海伦公式**

$S = \sqrt{p(p-a)(p-b)(p-c)}$，p 为半周长

**四边形最大面积**

$S = \sqrt{(p-a)(p-b)(p-c)(p-d)}$，p 为半周长

## 斯特林公式

n 很大时, $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$.

## 笛卡尔定理

4 个圆相切，外切 k = 1/r，内切 k = -1/r，圆退化成直线 k = 0。

$$(k_1+k_2+k_3+k_4)^2 = 2(k_1^2+k_2^2+k_3^2+k_4^2).$$

$$k_4 = k_1+k_2+k_3 \pm 2\sqrt{k_1k_2+k_2k_3+k_3k_1}.$$

## Purfer 序列

一个长度为 n-2 的 Purfer 序列唯一对应一个 n 个点的树，且 Purfer 序列中 i 出现的次数就是节点 i 的度数减一。

## 圆锥最大面积

全面积为 πa2 的圆锥最大面积为 sqrt(2)/12*πa^3。

## 正弦、余弦、正切

sinA / a = sinB / b = sinC/c

a² = b² + c²- 2bc·cosA

(a+b)/(a-b) = tan[(A+B)/2]/tan[(A-B)/2]

## 多边形面积

如果逆时针给出点坐标，值为正，

$$\frac{1}{2}\left|\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1}y_i)\right|$$

如果顺时针给出点坐标，值为负。

## Bash 游戏

有一堆石子共有 N 个。A B 两个人轮流拿，A 先拿。每次最少拿 1 颗，最多拿 K 颗，拿到最后 1 颗石子的人获胜。

若 n 不足 k+1，则 A 第一次取完，A 胜。

若 n 是 k+1 的倍数，每次 A 取 x，B 都能取 k+1-x，B 胜。否则，A 胜。

## 威佐夫游戏

有 2 堆石子。A B 两个人轮流拿，A 先拿。每次可以从一堆中取任意个或从 2 堆中取相同数量的石子，但不可不取。

两堆石头 x < y，若(y-x)*(sqrt(5)+1) / 2+1) == x，则 B 胜。否则 A 胜。

## 1/a 循环节长度

a 先约去 2 和 5 的因子->b，然后球欧拉函数值 φ，求一个最小的 x，使得 x|φ 且 10^x%b == 1。

# 计算几何

**计算几何基本函数**

```
#define eps 1e-8
#define PI acos(-1)

int sgn(double x)
{
    if(fabs(x) < eps)   return 0;
    if(x < 0)   return -1;
    return 1;
}

struct point
{
    double x,y;
    point(){};
    point(double a,double b):x(a),y(b){};
    friend point operator+(point a,point b)      //向量加法
    {
        return point(a.x+b.x,a.y+b.y);
    }
    friend point operator-(point a,point b)      //向量减法
    {
        return point(a.x-b.x,a.y-b.y);
    }
    friend double operator*(point a,point b)     //点积
    {
        return a.x*b.x+a.y*b.y;
    }
    friend double operator^(point a,point b)     //叉积
    {
        return a.x*b.y-a.y*b.x;
    }
    void trans(double B)     //绕原点旋转弧度 B
    {
        double tx = x,ty = y;
        x = tx*cos(B)-ty*sin(B);
        y = tx*sin(B)+ty*cos(B);
    }
};

struct line
{
    point s,e;
    line(){};
    line(point a,point b):s(a),e(b){};
    friend pair<int,point> operator&(line a,line b) //重合 0，平行 1，相交 2
    {
        point ans = a.s;
        if(sgn((a.s-a.e)^(b.s-b.e)) == 0)
        {
            if(sgn((a.s-b.e)^(b.s-b.e)) == 0)   return make_pair(0,ans);     //重合
            return make_pair(1,ans);     //平行
        }
        double t = ((a.s-b.s)^(b.s-b.e))/((a.s-a.e)^(b.s-b.e));
        ans.x += (a.e.x-a.s.x)*t;
        ans.y += (a.e.y-a.s.y)*t;
        return make_pair(2,ans);

    }
};
```

**两点距离**

```
double dis(point a,point b)     //两点距离
{
    return sqrt((a-b)*(a-b));
}
```

**点到直线距离**

```
point dis2(point p,line l)      //点到直线距离，返回垂点
{
    point ans;
    double t = ((p-l.s)*(l.e-l.s))/((l.e-l.s)*(l.e-l.s));
    ans.x = l.s.x+(l.e.x-l.s.x)*t;
    ans.y = l.s.y+(l.e.y-l.s.y)*t;
    return ans;
}
```

**点到线段距离**

```
point dis3(point p,line l)      //点到线段距离，返回线段上最近的点
{
    point ans;
    double t = ((p-l.s)*(l.e-l.s))/((l.e-l.s)*(l.e-l.s));
    if(t >= 0 && t <= 1)
    {
        ans.x = l.s.x+(l.e.x-l.s.x)*t;
        ans.y = l.s.y+(l.e.y-l.s.y)*t;
    }
    else if(dis(p,l.s) < dis(p,l.e))    ans = l.s;
    else    ans = l.e;
    return ans;
}
```

**判断线段相交**

```
bool inter(line a,line b)       //判断线段相交
{
    double x1 = a.s.x,y1 = a.s.y,x2 = a.e.x,y2 = a.e.y,x3 = b.s.x,y3 = b.s.y,x4 =
b.e.x,y4 = b.e.y;
    double t1 = (x2-x1)*(y3-y2)-(x3-x2)*(y2-y1);
    double t2 = (x2-x1)*(y4-y2)-(x4-x2)*(y2-y1);
    double t3 = (x4-x3)*(y1-y4)-(x1-x4)*(y4-y3);
    double t4 = (x4-x3)*(y2-y4)-(x2-x4)*(y4-y3);
    return t1*t2 <= 0 && t3*t4 <= 0;
}
```

**直线和线段相交**

```
bool inter2(line a,line b)      //判断直线 a 和线段 b 相交
{
    return sgn((b.s-a.e)^(a.s-a.e))*sgn((b.e-a.e)^(a.s-a.e)) <= 0;
}
```

**判断点在直线上**

```
bool online(point p,line l)     //判断点在直线上
{
    return sgn((l.s-p)^(l.e-p)) == 0;
}
```

**判断点在线段上**

```
bool onseg(point p,line l)      //判断点在线段上
{
    return sgn((l.s-p)^(l.e-p)) == 0 && sgn((p.x-l.s.x)*(p.x-l.e.x)) <= 0 && sgn((p.y-
l.s.y)*(p.y-l.e.y)) <= 0;
}
```

**判断点在凸多边形**

```
bool onconvex(point a,point *p,int n)   //判断点在凸多边形（凸包，点逆时针，若顺时针改
为>0）内，-1 在外，0 在边上，1 在内
{
    for(int i = 1;i <= n;i++)
    {
```

```
        int j = i+1;
        if(j > n)    j = 1;
        if(sgn((p[i]-a)^(p[j]-a)) < 0)  return -1;
        if(onseg(a,line(p[i],p[j])))    return 0;
    }
    return 1;
}
```

**判断点在多边形内**

```
bool onpoly(point a,point *p,int n)      //判断点在多边形内，-1 在外，0 在边上，1 在内
{
    int cnt = 0;
    line ray = line(a,point(-1e9,a.y));
    for(int i = 1;i <= n;i++)
    {
        int j = i+1;
        if(j > n)    j = 1;
        line side = line(p[i],p[j]);
        if(onseg(a,side))    return 0;
        if(sgn(side.s.y-side.e.y) == 0) continue;
        if(onseg(side.s,ray))
        {
            if(sgn(side.s.y-side.e.y) > 0)  cnt++;
        }
        else if(onseg(side.e,ray))
        {
            if(sgn(side.e.y-side.s.y) > 0)  cnt++;
        }
        else if(inter(ray,side))    cnt++;
    }
    if(cnt%2)    return 1;
    return -1;
}
```

**计算多边形面积**

```
double calcarea(point *p,int n)      //计算多边形面积，可顺时针或逆时针
{
    double ans = 0;
    for(int i = 1;i <= n;i++)
    {
        int j = i+1;
        if(j > n)    j = 1;
        ans += (p[i]^p[j])/2;
    }
    return fabs(ans);
}
```

**判断是否凸多边形**

```
bool isconvex(point *p,int n)        //判断是否凸多边形,允许共线边，可顺时针或逆时针
{
    bool s[3] = {0};
    for(int i = 1;i <= n;i++)
    {
        int j = i+1,k = j+2;
        if(j > n)    j -= n;
        if(k > n)    k -= n;
        s[sgn((p[j]-p[i])^(p[k]-p[i]))+1] = 1;
        if(s[0] && s[2])    return 0;
    }
    return 1;
}
```

**判断四点共面**

```
bool isface(int x1,int y1,int z1,int x2,int y2,int z2,int x3,int y3,int z3,int x4,int
y4,int z4)
{
    int a11 = x1-x2,a12 = x2-x3,a13 = x3-x4;
```

```
    int a21 = y1-y2,a22 = y2-y3,a23 = y3-y4;
    int a31 = z1-z2,a32 = z2-z3,a33 = z3-z4;
    return a11*a22*a33+a12*a23*a31+a13*a21*a32-a13*a22*a31-a11*a23*a32-a12*a21*a33 ==
0;
```

**判断三角形和圆相交**

```
bool trianglecircle(int XX,int YY,int R,int X1,int Y1,int X2,int Y2,int X3,int Y3)
{
    double a1 = (X1-XX)*(X1-XX)+(Y1-YY)*(Y1-YY);
    double a2 = (X2-XX)*(X2-XX)+(Y2-YY)*(Y2-YY);
    double a3 = (X3-XX)*(X3-XX)+(Y3-YY)*(Y3-YY);
    double rr = R*R;
    if(a1 < rr-1e-6 && a2 < rr-1e-6 && a3 < rr-1e-6)    return 1;
    if(a1 > rr+1e-6 && a2 > rr+1e-6 && a3 > rr+1e-6)
    {
        double t = ((XX-X1)*(X2-X1)+(YY-Y1)*(Y2-Y1))*((XX-X2)*(X1-X2)+(YY-Y2)*(Y1-Y2));
        if(t > 1e-6)
        {
            t = abs((X1-XX)*(Y2-YY)-(X2-XX)*(Y1-YY))/sqrt((X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-
Y2));
            if(t*t < rr+1e-6)   return 0;
        }
        t = ((XX-X3)*(X2-X3)+(YY-Y3)*(Y2-Y3))*((XX-X2)*(X3-X2)+(YY-Y2)*(Y3-Y2));
        if(t > 1e-6)
        {
            t = abs((X3-XX)*(Y2-YY)-(X2-XX)*(Y3-YY))/sqrt((X3-X2)*(X3-X2)+(Y3-Y2)*(Y3-
Y2));
            if(t*t < rr+1e-6)   return 0;
        }
        t = ((XX-X3)*(X1-X3)+(YY-Y3)*(Y1-Y3))*((XX-X1)*(X3-X1)+(YY-Y1)*(Y3-Y1));
        if(t > 1e-6)
        {
            t = abs((X3-XX)*(Y1-YY)-(X1-XX)*(Y3-YY))/sqrt((X3-X1)*(X3-X1)+(Y3-Y1)*(Y3-
Y1));
            if(t*t < rr+1e-6)   return 0;
        }
        return 1;
    }
    return 0;
}
```

**求凸包**

```
struct node
{
    int x,y;
    friend bool operator <(node a,node b)
    {
        if(a.x == b.x)  return a.y<b.y;
        return a.x<b.x;
    }
}a[1005],ans[1005];
int n,m;

int cross(node a,node b,node c)//向量积
{
    return (a.x-c.x)*(b.y-c.y)-(b.x-c.x)*(a.y-c.y);
}

int convex(int n)//求凸包上的点
{
    sort(a+1,a+n+1);
    int m = 0;
    for(int i = 1;i <= n;i++)
    {
```

```
        while(m > 1 && cross(ans[m],a[i],ans[m-1])<=0)  m--;
        ans[++m] = a[i];
    }
    int k = m;
    //求得上凸包
    for(int i = n-1;i >= 1;i--)
    {
        while(m > k && cross(ans[m],a[i],ans[m-1])<=0)    m--;
        ans[++m] = a[i];
    }
    if(n > 2)   m--;//起始点重复。
    return m;
}
```

**求三角形外心**

```
point waixin(point a,point b,point c)   //求三角形外心
{
    double a1 = b.x-a.x,b1 = b.y-a.y,c1 = (a1*a1+b1*b1)/2;
    double a2 = c.x-a.x,b2 = c.y-a.y,c2 = (a2*a2+b2*b2)/2;
    double d = a1*b2-a2*b1;
    return point(a.x+(c1*b2-c2*b1)/d,a.y+(a1*c2-a2*c1)/d);
}
```

**计算两圆相交面积**

```
double circlrarea(point a1,int r1,point a2,int r2)      //计算两圆相交面积
{
    double d = dis(a1,a2);
    if(r1+r2 < d+eps)   return 0;
    if(d < fabs(r1-r2)+eps)
    {
        double r = min(r1,r2);
        return PI*r*r;
    }
    double x = (d*d+r1*r1-r2*r2)/(2*d);
    double t1 = acos(x/r1),t2 = acos((d-x)/r2);
    return r1*r1*t1+r2*r2*t2-d*r1*sin(t1);
}
```

**平面最近点对**

```
int n,b[100005];
struct xx
{
    double x,y;
    friend bool operator<(xx a,xx b)
    {
        return a.x < b.x;
    }
}a[100005];

bool cmp(int x,int y)
{
    return a[x].y < a[y].y;
}

double dis(xx a,xx b)
{
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}

double f(int l,int r)
{
    if(l == r)  return 0;
    if(r-l == 1)    return dis(a[l],a[r]);
    if(r-l == 2)
    {
        double minn = 1e18;
```

```
        minn = min(minn,dis(a[l],a[l+1]));
        minn = min(minn,dis(a[l],a[r]));
        minn = min(minn,dis(a[l+1],a[r]));
        return minn;
    }
    int mid = (l+r)/2;
    double minn = min(f(l,mid),f(mid+1,r));
    int cnt = 0;
    for(int i = l;i <= r;i++)
    {
        if(a[i].x >= a[mid].x-minn && a[i].x <= a[mid].x+minn)  b[++cnt] = i;
    }
    sort(b+1,b+1+cnt,cmp);
    for(int i = 1;i <= cnt;i++)
    {
        for(int j = i+1;j <= cnt;j++)
        {
            if(a[b[j]].y-a[b[i]].y > minn)  break;
            minn = min(minn,dis(a[b[i]],a[b[j]]));
        }
    }
    return minn;
}

int main()
{
    while(scanf("%d",&n) && n)
    {
        for(int i = 1;i <= n;i++)   scanf("%lf%lf",&a[i].x,&a[i].y);
        sort(a+1,a+1+n);
        printf("%.2f\n",f(1,n)/2);
    }
    return 0;
}
```

**半平面交**

```
int sgn(double x)
{
    if(fabs(x) < eps)   return 0;
    if(x < 0)   return -1;
    return 1;
}

struct point
{
    double x,y;
    point(){};
    point(double a,double b):x(a),y(b){};
    friend point operator+(point a,point b)     //向量加法
    {
        return point(a.x+b.x,a.y+b.y);
    }
    friend point operator-(point a,point b)     //向量减法
    {
        return point(a.x-b.x,a.y-b.y);
    }
    friend double operator*(point a,point b)    //点积
    {
        return a.x*b.x+a.y*b.y;
    }
    friend double operator^(point a,point b)    //叉积
    {
        return a.x*b.y-a.y*b.x;
    }
```

```
}ans[20005];

struct line
{
    point s,e;
    double k;
    line(){};
    line(point a,point b):s(a),e(b)
    {
        k = atan2(e.y-s.y,e.x-s.x);
    };
    friend point operator&(line a,line b)
    {
        point ans = a.s;

        double t = ((a.s-b.s)^(b.s-b.e))/((a.s-a.e)^(b.s-b.e));
        ans.x += (a.e.x-a.s.x)*t;
        ans.y += (a.e.y-a.s.y)*t;
        return ans;
    }
}a[20005],q[20005];

bool HPIcmp(line a,line b)    //直线左边
{
    if(fabs(a.k - b.k) > eps)   return a.k < b.k;
    return ((a.s - b.s)^(b.e - b.s)) < 0;
}

//返回核的凸包点
void HPI(line *a,int n,point *ans,int &cnt)
{
    sort(a+1,a+n+1,HPIcmp);
    int tot = 0;
    for(int i = 1;i <= n;i++)
    {
        if(fabs(a[i].k-a[i-1].k) > eps)   a[++tot] = a[i];
    }
    int head = 0,tail = 1;
    q[0] = a[1];
    q[1] = a[2];
    cnt = 0;
    for(int i = 3;i <= tot;i++)
    {
        if(fabs((q[tail].e-q[tail].s)^(q[tail-1].e-q[tail-1].s)) < eps ||
fabs((q[head].e-q[head].s)^(q[head+1].e-q[head+1].s)) < eps)  return;
        while(head < tail && (((q[tail]&q[tail-1])-a[i].s)^(a[i].e-a[i].s)) > eps)
tail--;
        while(head < tail && (((q[head]&q[head+1])-a[i].s)^(a[i].e-a[i].s)) > eps)
head++;
        q[++tail] = a[i];
    }
    while(head < tail && (((q[tail]&q[tail-1])-q[head].s)^(q[head].e-q[head].s)) > eps)
tail--;
    while(head < tail && (((q[head]&q[head-1])-q[tail].s)^(q[tail].e-q[tail].e)) > eps)
head++;
    if(tail <= head+1)  return;
    for(int i = head;i < tail;i++)  ans[++cnt] = q[i]&q[i+1];
    if(head < tail-1)   ans[++cnt] = q[head]&q[tail];
}
```

## 旋转卡壳求最远点对

```
point a[50005],ans[50005];
int n,m;

int cross(point a,point b,point c)//向量积
{
    return (a.x-c.x)*(b.y-c.y)-(b.x-c.x)*(a.y-c.y);
}

int dis2(point a,point b)
{
    return (a-b)*(a-b);
}
int convex(int n)//求凸包上的点
{
    sort(a+1,a+n+1);
    int m = 0;
    for(int i = 1;i <= n;i++)
    {
        while(m > 1 && cross(ans[m],a[i],ans[m-1])<=0)  m--;
        ans[++m] = a[i];
    }
    int k = m;
    //求得上凸包
    for(int i = n-1;i >= 1;i--)
    {
        while(m > k && cross(ans[m],a[i],ans[m-1])<=0)    m--;
        ans[++m] = a[i];
    }
    if(n > 2)   m--;//起始点重复。
    return m;
}

int rotatingcalipers(point *a,int n)
{
    int ans = 0,now = 2,ne = 3;
    if(ne > n)  ne = 1;
    for(int i = 1;i <= n;i++)
    {
        int j = i+1;
        if(j  > n)  j = 1;
        point t = a[i]-a[j];
        while((t^(a[ne]-a[now])) < 0)
        {
            now = ne;
            ne = now+1;
            if(ne > n)  ne = 1;
        }
        ans = max(ans,max(dis2(a[i],a[now]),dis2(a[j],a[ne])));
    }
    return ans;
}

int main()
{
    while(~scanf("%d",&n))
    {
        for(int i = 1;i <= n;i++)   scanf("%d%d",&a[i].x,&a[i].y);
        m = convex(n);
        printf("%d\n",rotatingcalipers(ans,m));
    }
}
```

## 旋转卡壳球平面点最大三角形面积

```
//已求得凸包
int rotatingcalipers(point *a,int n)
{
    int ans = 0;
    for(int i = 1;i <= n;i++)
    {
        int j = i+1;
        if(j  > n)  j = 1;
        int k = j+1;
        if(k > n)   k = 1;
        int kk = k+1;
        if(kk > n)  kk = 1;
        while(j != i && k != i)
        {
            ans = max(ans,abs((a[j]-a[i]^(a[k]-a[i]))));
            while(((a[i]-a[j])^(a[kk]-a[k])) < 0)
            {
                k = kk;
                kk++;
                if(kk > n)  kk = 1;
            }
        }
    }
    return ans;
}
```

## 旋转卡壳求两凸包最小距离

```
double dis4(point a,point b,point c)
{
    return dis(a,dis3(a,line(b,c)));
}

double dis5(point a,point b,point c,point d)
{
    double ans1 = min(dis4(a,c,d),dis4(b,c,d));
    double ans2 = min(dis4(c,a,b),dis4(d,a,b));
    return min(ans1,ans2);
}

double getangel(point a,point b,point c,point d)
{
    return (b-a)^(d-c);
}

int rotatingcalipers(point *a,int n,point *b,int m)
{
    int sa = 1,sb = 1;
    for(int i = 1;i <= n;i++)
    {
        if(sgn(a[i].y-a[sa].y) < 0) sa = i;
    }
    for(int i = 1;i <= m;i++)
    {
        if(sgn(b[i].y-b[sb].y) < 0) sb = i;
    }
    double t,ans = dis(a[sa],b[sb]);
    int na = sa+1,nb = sb+1;
    if(na > n)  na = 1;
    if(nb > m)  nb = 1;
    for(int i = 1;i <= n;i++)
    {
        while(sgn(t = getangel(a[sa],a[na],b[sb],b[nb])) < 0)
        {
```

```
            sb = nb;
            nb++;
            if(nb > m)  nb = 1;
        }
        if(sgn(t) == 0) ans = min(ans,dis5(a[sa],a[na],b[sb],b[nb]));
        else    ans = min(ans,dis4(b[sb],a[sa],a[na]));
        sa = na;
        na++;
        if(na > n)  na = 1;
    }
    return ans;
}

//已求得凸包
double solve(point *a,int n,point *b,int m)
{
    return min(rotatingcalipers(a,n,b,m),rotatingcalipers(b,m,a,n));
}
```

**高精度**

```
//输入的两个数要求为正。
int compare(string str1,string str2)
{
    if(str1.length() > str2.length())    return 1;
    else if(str1.length() < str2.length())    return -1;
    else    return str1.compare(str2);
}

string add(string str1,string str2)//加法
{
    string str;
    int len1 = str1.length(),len2 = str2.length();
    if(len1 < len2)
    {
        for(int i = 1;i <= len2-len1;i++)    str1 = "0"+str1;
    }
    else
    {
        for(int i = 1;i <= len1-len2;i++)    str2 = "0"+str2;
    }
    int cf = 0,temp;
    for(int i = str1.length()-1;i >= 0;i--)
    {
        temp = str1[i]-'0'+str2[i]-'0'+cf;
        cf = temp/10;
        temp %= 10;
        str = char(temp+'0')+str;
    }
    if(cf != 0)    str = char(cf+'0')+str;
    return str;
}

string sub(string str1,string str2)//减法
{
    string str;
    int flag = 0;
    if(compare(str1,str2) < 0)
    {
        flag = 1;
        swap(str1,str2);
    }
    int tmp = str1.length()-str2.length(),cf = 0;
    for(int i = str2.length()-1;i >= 0;i--)
    {
```

```
        if(str1[tmp+i] < str2[i]+cf)
        {
            str = char(str1[tmp+i]-str2[i]-cf+'0'+10)+str;
            cf = 1;
        }
        else
        {
            str = char(str1[tmp+i]-str2[i]-cf+'0')+str;
            cf = 0;
        }
    }
    for(int i = tmp-1;i >= 0;i--)
    {
        if(str1[i]-cf >= '0')
        {
            str = char(str1[i]-cf)+str;
            cf = 0;
        }
        else
        {
            str = char(str1[i]-cf+10)+str;
            cf = 1;
        }
    }
    str.erase(0,str.find_first_not_of('0'));
    if(str.empty())   str = "0";
    if(flag)    str = "-"+str;
    return str;
}

string mul(string str1,string str2)//乘法
{
    string str;
    int len1 = str1.length();
    int len2 = str2.length();
    string tempstr;
    for(int i = len2-1;i >= 0;i--)
    {
        tempstr = "";
        int temp = str2[i]-'0',t = 0,cf = 0;
        if(temp != 0)
        {
            for(int j = 1;j <= len2-1-i;j++)    tempstr += "0";
            for(int j = len1-1;j >= 0;j--)
            {
                cf = (temp*(str1[j]-'0')+cf);
                t = cf%10;
                cf /= 10;
                tempstr = char(t+'0')+tempstr;
            }
            if(cf != 0) tempstr = char(cf+'0')+tempstr;
            str=add(str,tempstr);
        }
    }
    str.erase(0,str.find_first_not_of('0'));
    if(str.empty())   str = "0";
    return str;
}


void div(string str1,string str2,string &quotient,string &residue)//除法取余
{
    quotient = "";
    residue = "";
```

```
    if(str2 == "0")
    {
        quotient = "ERROR";
        residue = "ERROR";
        return;
    }
    if(str1 == "0")
    {
        quotient = "0";
        residue = "0";
        return;
    }
    int res = compare(str1,str2);
    if(res < 0)
    {
        quotient = "0";
        residue = str1;
        return;
    }
    else
    {
        int len1 = str1.length();
        int len2 = str2.length();
        string tempstr;
        tempstr.append(str1,0,len2-1);
        for(int i = len2-1;i < len1;i++)
        {
            tempstr = tempstr+str1[i];
            tempstr.erase(0,tempstr.find_first_not_of('0'));
            if(tempstr.empty())    tempstr = "0";
            for(char ch = '9';ch >= '0';ch--)
            {
                string str,tmp;
                str = str+ch;
                tmp = mul(str2,str);
                if(compare(tmp,tempstr) <= 0)
                {
                    quotient = quotient+ch;
                    tempstr = sub(tempstr,tmp);
                    break;
                }
            }
        }
        residue = tempstr;
    }
    quotient.erase(0,quotient.find_first_not_of('0'));
    if(quotient.empty()) quotient="0";
}
```

# 其他

**希尔排序**

```
void shell_sort(int arr[],int n)//希尔排序接口
{
    for(int gap = n/2;gap > 0;gap /= 2)
    {
        for(int i =gap;i < n;i++)
        {
            int t = arr[i],j;
            for(j = i-gap;j >= 0 && arr[j] > t;j -= gap)    arr[j+gap] = arr[j];
            arr[j+gap] = t;
        }
    }
}
```

**归并排序**

```
void merge(int a[],int first,int mid,int last,int temp[])//合并
{
    int i = first,j = mid+1,k = 0;
    while (i <= mid && j <= last)
    {
        if(a[i] < a[j])    temp[k++] = a[i++];
        else    temp[k++] = a[j++];
    }
    while (i <= mid)    temp[k++] = a[i++];
    while (j <= last)    temp[k++] = a[j++];
    for(i = 0;i < k;i++)    a[first+i] = temp[i];
}

void _merge_sort(int arr[],int temp[],int first,int last)//归并
{
    if(first < last)
    {
        int mid = (first+last)/2;
        _merge_sort(arr,temp,first,mid);
        _merge_sort(arr,temp,mid+1,last);
        merge(arr,first,mid,last,temp);
    }
}

void merge_sort(int arr[],int n)//归并排序接口
{
    int* temp = (int*)malloc(n*sizeof(int));
    _merge_sort(arr,temp,0,n-1);
    free(temp);
}
```

**堆排序**

```
void percdown(int arr[],int p,int n)//调整最大堆
{
    int Parent,Child,x = arr[p];
    for(Parent = p;Parent*2+1 < n;Parent = Child)
    {
        Child = Parent*2+1;
        if(Child != n-1 && arr[Child] < arr[Child+1])    Child++;
        if(x >= arr[Child])    break;
        else    arr[Parent] = arr[Child];
    }
    arr[Parent] = x;
}

void heap_sort(int arr[],int n)//堆排序接口
{
    for(int i = n/2-1;i >= 0;i--)    percdown(arr,i,n);
    for(int i = n-1;i > 0;i--)
    {
        swap(arr[0],arr[i]);
        percdown(arr,0,i);
    }
}
```

**快速排序**

```
void _quick_sort(int arr[],int l,int r)
{
    if(l < r)
    {
        int i = l,j = r,x = arr[l];
        while(i < j)
        {
            while(i < j && arr[j] >= x)    j--;
```

```
            if(i < j)    arr[i++] = arr[j];
            while(i < j && arr[i] < x)    i++;
            if(i < j)    arr[j--] = arr[i];

        }
        arr[i] = x;
        _quick_sort(arr,l,i-1);
        _quick_sort(arr,i+1,r);
    }
}
```

**头文件**

```
#pragma comment(linker, "/STACK:102400000,102400000")
#include<cstdio>
#include<iostream>
#include<iomanip>
#include<algorithm>
#include<cmath>
#include<cstring>
#include<string>
#include<vector>
#include<stack>
#include<queue>
#include<set>
#include<map>
#include<list>
#include<deque>
#include<bitset>
#include<cstdlib>
#define LL long long
#define PI acos(-1)
#define eps 1e-8
#define lowbit(x) (x&-x)
#define MOD 1000000007
#define INF 0x3f3f3f3f
#define MEM(a,x) memset(a,x,sizeof(a))
using namespace std;
```

**输入外挂**

```
void read(int &ans)
{
    ans = 0;
    int flag = 0,ch;
    if((ch = getchar()) == '-') flag = 1;
    else if(ch >= '0' && ch <= '9') ans = ch-'0';
    while((ch = getchar()) >= '0' && ch <= '9' )    ans = ans*10+ch-'0';
    if(flag)   ans *= -1;
}
```

**随机数**

```
#include<random>
std::random_device rd;
rd();

std::mt19937 mt(rd());
mt();
```