Standard Source code Library

AcFast

February 28, 2014

Contents

1 算法

1.1 网络流算法

1.1.1 网络流模型

1、最大权闭合图：定义一个有向图$G=(V,E)$的闭合图是该有向图的一个点集，且该点集的所有出边都还指向该点集。即闭合图内的任一点的任意后继也一定在闭合图中。更形式化地说，闭合图是这样一个点集$V'\in V$，满足对于$\forall u\in V'$引出的$\forall\langle u,v\rangle\in E$，必有$v\in V'$成立。还有一种等价定义为：满足对于$\forall\langle u,v\rangle\in E$，若有$u\in V'$成立，必有$v\in V'$成立。闭合图允许超过一个连通块。

给每个点v分配一个点权w_v（任意实数，可正可负）。最大权闭合图，是一个点权之和最大的闭合图，即最大化$\sum_{v\in V'}w_v$。

在许多实际应用中，给出的有向图常常是一个有向无环图（DAG），闭合图的性质恰好反映了事件间的必要条件的关系：一个时间的发生，它所需要的所有前提也要发生。

最大权闭合图转化成最小割模型：在原图点集的基础上增加源s和汇t；将原图每条有向边$\langle u,v\rangle\in E$替换为容量为$c(u,v)=+\infty$的有向边$\langle u,v\rangle\in E_N$；增加连接源$s$到原图每个正权点$v(w_v>0)$的有向边$\langle s,v\rangle\in E_N$，容量为$c(s,v)=w_v$；增加连接原图的每个负权点$v(w_v<0)$到汇$t$的有向边$\langle v,t\rangle\in E_N$，容量为$c(v,t)=-w_v$。其中，正无限$\infty$定义为任意一个大于$\sum_{v\in V}|w_v|$的整数。

2、最大密度子图：定义一个无向图$G=(V,E)$的密度D为该图的边数$|E|$与该图的点数$|V|$的比值$D=\frac{|E|}{|V|}$。给出一个无向图$G=(V,E)$，其具有最大密度的子图$G'=(V',E')$称为最大密度子图，即最大化$D'=\frac{|E'|}{|V'|}$。

性质：无向图中，任意两个具有不同密度的子图G_1,G_2，它们的密度差不小于$\frac{1}{n^2}$。

构图：在原图点集V的基础上增加源s和汇t；将每条原无向边(u,v)替换为两条容量为1的有向边$\langle v,u\rangle$和$\langle u,v\rangle$；增加连接源s到原图每个点v的有向边$\langle s,v\rangle$，容量为U；增加连接原图每个点v到汇t的有向边$\langle v,t\rangle$，容量为$(U+2g-d_v)$。其中U是图总的边数，g是二分的答案，d_v表示顶点v的度。

1.1.2 SAP

```
#include <queue>
#include <cstring>
#include <algorithm>
using namespace std;

const int MAX_N = 102401;
const int MAX_M = 256000;
const int INF = 0x3f3f3f3f;

struct SAP{
    //S -> source, T -> sink
    //n -> |V|, cnt -> 2 * |E|
    int S, T, n, cnt;
    int f[MAX_M], adj[MAX_M], next[MAX_M];
    int g[MAX_N], last[MAX_N], h[MAX_N], vh[MAX_N];
```

```
void init(int _n){
    n = _n;
    cnt = 0;
    for (int i = 0; i < n; i++) g[i] = -1;
}

void addedge(int x, int y, int limit){
    f[cnt] = limit;
    adj[cnt] = y;
    next[cnt] = g[x];
    g[x] = cnt++;

    f[cnt] = 0;
    adj[cnt] = x;
    next[cnt] = g[y];
    g[y] = cnt++;
}

void BFS(int T){
    for (int i = 0; i < n; i++){
        h[i] = -1;
        vh[i] = 0;
    }
    vh[h[T] = 0] = 1;
    queue<int> q;
    q.push(T);
    while (!q.empty()){
        int node=q.front(); q.pop();
        for (int p = g[node]; p != -1; p = next[p]){
            /* p % 2 == 1 indicate reverse edge
             * h[adj[p]] == -1 indicate the node isn't label
            */
            if ((p&1) && h[adj[p]] == -1){
                ++vh[h[adj[p]] = h[node] + 1];
                q.push(adj[p]);
            }
        }
    }
}

int dfs(int node, int add){
    if (node == T) return add;
    int minh = n, p = last[node];
    do{
        if (f[p] > 0){
            int y = adj[p];
            if (h[node] == h[y] + 1){
                int temp = dfs(y, min(add, f[p]));
```

```
                if (temp > 0){
                    f[p] -= temp;
                    f[p ^ 1] += temp;
                    last[node] = p;
                    return temp;
                }
            }
            if (h[S] >= n) return 0;
            minh = min(minh, h[y] + 1);
        }
        p = next[p];
        if (p == -1) p = g[node];
    } while (p != last[node]);

    if (--vh[h[node]] == 0) h[S] = n;
    ++vh[h[node] = minh];
    return 0;
}

int maxflow(int _S, int _T){
    S = _S, T = _T;
    if (g[S] == -1){
        //the graph not connected with source
        return -1;
    }

    /* make label with BFS, sometime the effect is good
    BFS(T);
    for (int i = 0; i < n; i++){
        last[i] = g[i];
    }
    */

    //don't make label with BFS
    for (int i = 0; i < n; i++){
        h[i] = vh[i] = 0;
        last[i] = g[i];
    }
    //end

    int flow = 0;
    while (h[S] < n) flow += dfs(S, INF);
    return flow;
}
} network_flow;
```

1.1.3 有上下界的最大流

```
#include <algorithm>
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;

const int INF=0x7FFFFFFF;

const int maxn=1001;
const int maxm=100001;

int n,m,s,t,ca,P,tot,S,T,NT,flow,maxtot;
int g[maxn],last[maxn],h[maxn],vh[maxn];
int a[maxm][4],f[maxm],adj[maxm],next[maxm];

void insert(int x, int y, int limit){
    f[tot]=limit;
    adj[tot]=y;
    next[tot]=g[x];
    g[x]=tot++;

    f[tot]=0;
    adj[tot]=x;
    next[tot]=g[y];
    g[y]=tot++;
}

int dfs(int now, int add){
    if (now==T) return add;
    int y, tmp, minh=NT+1, p=last[now];
    do{
        y=adj[p];
        if (f[p]>0 && p<maxtot){
            if (h[now]==h[y]+1){
                tmp=dfs(y,min(f[p],add));
                if (tmp!=0){
                    f[p]-=tmp;
                    f[p^1]+=tmp;
                    last[now]=p;
                    return tmp;
                }
            }
            minh=min(minh,h[y]+1);
            if (h[S]>NT) return 0;
        }
        p=next[p];
```

```
        if (p==-1) p=g[now];
    } while (p!=last[now]);

    if (--vh[h[now]]==0) h[S]=NT+1;
    h[now]=minh; ++vh[minh];
    return 0;
}

int getflow(){
    //a[i][0],a[i][1],a[i][2],a[i][3]表示(a[i][0],a[i][1])
    //这条边以及下界流量为a[i][2]，上界流量为a[i][3]
    //tot是边的总数
    tot=0;
    memset(g,255,sizeof(g));
    for (int i=1; i<=m; i++)
    insert(a[i][0],a[i][1],a[i][3]-a[i][2]);
    int tmp=tot;
    //前面tot条边都是上界流量-下界流量
    //做第二次最大流的时候只能用这tot条边
    //所以在这里要用tmp纪录tot

    //S和T在第一次网络流中是超级源和超级汇。
    //对于一条下界边(u,v,lower)，连(u,T,lower)和(S,v,lower)
    S=n+1; T=S+1;
    for (int i=1; i<=m; i++)
    {
        insert(a[i][0],T,a[i][2]);
        insert(S,a[i][1],a[i][2]);
    }
    //s和t是原图中的源和汇，第一次最大流要连一条(t,s,无穷大)的边
    insert(t,s,INF);

    memset(h,0,sizeof(h));
    memset(vh,0,sizeof(vh));
    for (int i=1; i<=T; i++) last[i]=g[i];
    //maxtot表示做网络流时所能经过的编号最大的边，NT表示有多少个顶点
    maxtot=tot; vh[0]=T; flow=0; NT=n+2;
    while (h[S]<=NT) flow+=dfs(S,INF);
    //第一次最大流，如果流量不等于所有边的下界总和，则方案不合法
    if (flow!=m*lower) return -1;

    //第二次最大流在残余图上做
    //且只能使用前tmp条边，源和汇是原图的源汇
    //f[tot-1]表示从原图的源流出的流量，加入到最后的流量中
    S=s; T=t; flow=f[tot-1];
    memset(h,0,sizeof(h));
    memset(vh,0,sizeof(vh));
    for (int i=1; i<=n; i++) last[i]=g[i];
```

```
    //若T比定点个数小，则设NT=n，所有的h[S]<=NT而不是<=T
    maxtot=tmp; vh[0]=n; NT=n;
    while (h[S]<=NT) flow+=dfs(S,INF);
    //flow就是这个有上下界流量的最大流
    return flow;
}
```

1.1.4 费用流

1、求解可行流：给定一个网络流图, 初始时每个节点不一定平衡 (每个节点可以有盈余或不足), 每条边的流量可以有上下界, 每条边的当前流量可以不满足上下界约束. 可行流求解中没有源和汇的概念, 算法的目的是寻找一个可以使所有节点都能平衡, 所有边都能满足流量约束的方案, 同时可能附加有最小费用的条件 (最小费用可行流).

2、求解最大流：给定一个网络流图, 其中有两个特殊的节点称为源和汇. 除源和汇之外, 给定的每个节点一定平衡. 源可以产生无限大的流量, 汇可以吸收无限大的流量. 标准的最大流模型, 初始解一定是可行的 (例如, 所有边流量均为零), 因此边上不能有下界. 算法的目的是寻找一个从源到汇流量最大的方案, 同时不破坏可行约束, 并可能附加有最小费用的条件 (最小费用最大流).

3、扩展的最大流：在有上下界或有节点盈余的网络流图中求解最大流. 实际上包括两部分, 先是消除下界, 消除盈余, 可能还需要消除不满足最优条件的流量 (最小费用流), 找到一个可行流, 再进一步得到最大流. 因此这里我们的转化似乎是从最大流转化为可行流再变回最大流, 但其实质是将一个过程 (扩展的最大流) 变为了两个过程 (可行流 + 最大流).

4、最小费用流的各种转化

- 1. 最小费用 (可行) 流 $\rightarrow$ 最小费用最大流
  - 建立超级源 s' 和超级汇 t'，对顶点 i，若 $e_i > 0$ 添加便 $s' \rightarrow i, c = 0, u = e_i$, 若 $e_i < 0$ 添加边 $i \rightarrow t', c = 0, u = -e_i$, 之后求从 s' 到 t' 的最小费用最大流, 如果流量等于 $\sum e_i$, 就存在可行流，残量网络已在原图上求出.
- 2. 最小费用最大流 $\rightarrow$ 最小费用 (可行) 流
  - 连边 $t \rightarrow s, c = -\infty, u = +\infty$, 所有点 i 有 $e_i = 0$, 然后直接球解最小费用最大流.
- 3. 最小费用 (可行) 流中负权边的消除
  - 直接将负权边满流
- 4. 最小费用最大流中负权边的消除
  - 先连边 $t \rightarrow s, c = 0, u = +\infty$, 使用 (3.) 中的方法消除负权边，使用 (1.) 中的方法求出最小费用 (可行) 流，之后距离标号不变, 再球最小费用最大流; 注意此时增广费用不能机械使用源点的标号，而应该是源点会点标号之差。

```
pair<int,int> operator +(const pair<int,int> &a, const pair<int,int> &b){
    return make_pair(a.first + b.first, a.second + b.second);
}

struct netwrok{
    static const int MAX_N = 256;
    static const int MAX_M = 102400;
    static const int INF = 0x3f3f3f3f;

    bool vis[MAX_N];
    int n, cnt, S, T, flow;
    int g[MAX_N], dist[MAX_N], pre[MAX_N];
    int f[MAX_M], adj[MAX_M], next[MAX_M], cost[MAX_M];

    void ins(int x, int y, int limit, int c){
        f[cnt] = limit;
        cost[cnt] = c;
        adj[cnt] = y;
        next[cnt] = g[x];
        g[x] = cnt++;

        f[cnt] = 0;
        cost[cnt] = -c;
        adj[cnt] = x;
        next[cnt] = g[y];
        g[y] = cnt++;
    }

    void init(int _n, int _S, int _T){
        n = _n, S = _S, T = _T;
        cnt = 0;
        memset(g, 255, sizeof(g));
    }

    bool spfa(){
        memset(vis, 0, sizeof(vis));
        memset(dist, 0x3f, sizeof(dist));
        queue<int> q;
        q.push(S);
        vis[S] = true;
        dist[S] = 0;
        while (!q.empty()){
            int node = q.front(); q.pop();
            for (int p = g[node]; p != -1; p = next[p]){
                if (f[p] > 0 && dist[node] + cost[p] < dist[adj[p]]){
                    dist[adj[p]] = dist[node] + cost[p];
                    pre[adj[p]] = p;
                    if (!vis[adj[p]]){
```

```
                    q.push(adj[p]);
                    vis[adj[p]] = true;
                }
            }
        }
        vis[node] = false;
    }
    return dist[T] != INF;
}

pair<int,int> augPath(){
    int C = 0, F = INF;
    for (int i = T; i != S; i = adj[pre[i] ^ 1]) F = min(F, f[pre[i]]);
    for (int i = T; i != S; i = adj[pre[i] ^ 1]){
        C += F * cost[pre[i]];
        f[pre[i]] -= F;
        f[pre[i] ^ 1] += F;
    }
    return make_pair(F, C);
}

//ret.first -> flow, ret.second -> cosst
pair<int,int> maxflow(){
    pair<int,int> ret;
    while (spfa()) ret = ret + augPath();
    return ret;
}
} g;
```

1.1.5 KM算法

```
//可以解决最大权完全匹配，下标从0开始
struct km{
    static const int MAX_N = 128;
    static const int INF = 0x3f3f3f3f;
    int n, m;
    bool sx[MAX_N], sy[MAX_N];
    int cx[MAX_N], cy[MAX_N], lx[MAX_N], ly[MAX_N], g[MAX_N][MAX_N];

    void init(int _n, int _m){
        n = _n, m = _m;
        for (int i = 0; i < n; i++){
            for (int j = 0; j < m; j++) g[i][j] = -INF;
        }
    }

    void ins(int x, int y, int c){
        g[x][y] = c;
```

```
}

int find(int k){
    sx[k] = true;
    for (int i = 0; i < m; i++) if (!sy[i] && lx[k] + ly[i] == g[k][i]){
        sy[i] = true;
        if (cy[i] == -1 || find(cy[i])){
            cx[k] = i;
            cy[i] = k;
            return true;
        }
    }
    return false;
}


int match(){
    for (int i = 0; i < n; i++){
        cx[i] = cy[i] = -1;
        ly[i] = 0;
        lx[i] = -INF;
    }
    for (int i = 0; i < n; i++){
        for (int j = 0; j < m; j++) lx[i] = max(lx[i], g[i][j]);
    }
    for (int k = 0; k < n; k++) if (cx[k] == -1){
        while (true){
            memset(sx, 0, sizeof(sx));
            memset(sy, 0, sizeof(sy));
            if (find(k)) break;
            int temp = INF;
            for (int i = 0; i < n; i++) if (sx[i]){
                for (int j = 0; j < m; j++) if (!sy[j]){
                    temp = min(temp, lx[i] + ly[j] - g[i][j]);
                }
            }
            for (int i = 0; i < n; i++) if (sx[i]){
                lx[i] -= temp;
            }
            for (int i = 0; i < m; i++) if (sy[i]){
                ly[i] += temp;
            }
        }
    }
    int ret = 0;
    for (int i = 0; i < n; i++) ret += g[i][cx[i]];
    return ret;
}
```

```
} g;
```

1.1.6 网络流割点

网络流求割点：将一个点拆成两个点，两个点之间连一条边。求出最大流后，从源点开始 DFS 遍历，如果一条边没有满流就可以通过，记录源点 S 可以到达的点。如果一条边是割边，则这条边的两个点分别属于 S 集合和 T 集合。

1.2 生成树算法

1.2.1 生成树计数

一个完全图 K_n 有 n^{n-2} 棵生成树，即 n 个节点的带标号无根树有 n^{n-2} 个。

证明用到 **prufercode**：一棵 n 无根树的 **prufercode** 是这样转化的：每次选一个编号最小的叶节点，删除它，并把它所连的父亲节点的编号写下，直到这棵树剩下 2 个节点为止。那么生成的这 $n-2$ 个数组成的序列就是这棵树的 **prufercode**。**prufercode** 和树是一一对应的，而一个长度为 $n-2$，每个数字的范围为 $[1,n]$ 的序列一共有 n^{n-2} 种可能，所以 n 个节点的带标号无根树一共有 n^{n-2} 个。

以上是 Cayley 公式，它的一个应用：n 个节点，每个节点的度分别为 $d_1, d_2, \ldots, d_n$，那么生成树的个数为 $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}$。因为顶点 i 在序列中出现了 d_i-1 次。

对于完全二分图，两边的顶点分别为 n, m，那么生成树的个数为 $n^{m-1} * m^{n-1}$。

1.2.2 无向图的生成数计数-MatrixTree 定理

给出一个无向图 $G=(V,E)$，求生成树个数。做法是构造一个 $n*n$ 的 Kichhoff 矩阵。矩阵的对角线 (i,i) 的位置填的是第 i 个顶点的度，对于 G 的边 (v_i,v_j) 在矩阵 (i,j) 和 (j,i) 的位置填 -1(若 (i,j) 有 k 条重边，那么矩阵 (i,j) 和 (j,i) 的位置填 $-k$)，然后生成树的个数就是 $n*n$ 的矩阵的 $n-1$ 阶的行列式。具体做法就是删除任意的第 r 行 r 列，然后求矩阵的行列式。

1.2.3 无向带权图最小生成树计数

把所有的边按照边权从小到大排序，然后做 Kruskal。

假设已经处理了边权 $w_i < w$ 的边，形成一个森林 T，现在考虑所有边权为 w 的边。

1、若一条边权为 w 的边 (u,v) 所连接的两个顶点在森林 T 中属于统一个块，那么 (u,v) 这条边是不可能存在于最小生成树的方案中，否则 (u,v) 可以存在于最小生成树的方案中。

2、把所有边权为 w，且根据森林 T 判断出可以存在于最小生成树方案中的边找出来，假设这些边集为 E。我们可以把森林 T 中的每一个块缩成一个点，那么用 E 中的边去连接 T，就形成了一些连通块。对于每一个连通块的方案数就是对这个连通块做一个生成树计数就可以了，然后把这些连通块各自的方案数相乘就是选择边权为 w 的边的方案数。

3、算完边权为 w 的方案之后，就把这些边加入到 T 中，形成新的森林。

1.2.4 带限制的最小生成树问题

- Problem

  无向带权连通图，每条边是黑色或白色。让你求一棵最小权的恰好有 K 条白色边的生成树。

- Solution

  1、对于一个图，如果存在一棵生成树，它的白边数量为 x，那么就称 x 是合法白边数。所有的合法白边数组成一个区间 $[l,r]$。

  2、对于一个图，如果存在一棵最小生成树，它的白边数量为 x，那么就称 x 是最小合法白边数。所有的最小合法白边数组成一个区间 $[l,r]$。

  3、将所有白边追加权值 x 所得到的最小生成树，如果该树有 a 条白边。那么这可树就是 a 条白边最小生成树的一个最优解。

  所以可以二分得到一个最大的 x 使得所求的最小生成树的白边的最小值和最大值所组成的一个区间 $[l,r]$，若 $K \in [l,r]$，则该最小生成树就是最优解。其实只要求出最大的一个 x 使得最小生成树中最大白边数量不小于 K 即可。记录答案的时候，必须把枚举的 x 加上，然后在最后减去 $K*x$，如果直接在计算的时候加原来白边的长度的话，有可能超过 K 条边。

  对于黑、白不同的边，他们内部的顺序是一样的，所以一开始将黑白边分别排序，这样在二分判断的时候只需要 $O(M)$ 的时间复杂度去合并排序的边了，其中 M 是边数。总的时间复杂度为 $O(MlogW+NlogN)$ 其中 N 是顶点数，W 是边权。

- Expansion

  1、限制某个节点 $node$ 的度数恰好为 K 的最小生成树。解法就是把和 $node$ 关联的边标记为白边，其余的边为黑边，然后就转化为上面的经典问题了。

1.3 图算法

1.3.1 无向图割点

```
/*
 * dfn[]表示访问时的编号，lowlink[]表示不经过父子边能访问到的dfn的最小值
 * 对于v的子节点u，若lowlink[u]>=dfn[v],则v是一个割点。不过对于根节点要特
 * 判，若根节点的子节点个数>1，那么根节点就是割点。
 *
 * 以下算法不能处理重边，如果要处理重边，则需要加一个判断边的id是否被访问
 * 程序默认1为根节点
*/

void dfs(int node, int fa){
    dfn[node]=lowlink[node]=++total;
    for (int i=0; i<g[node].size(); i++)
        if (g[node][i]!=fa){
            if (!dfn[g[node][i]]){
                dfs(g[node][i],node);
                lowlink[node]=min(lowlink[node],lowlink[g[node][i]]);
                if (lowlink[g[node][i]]>=dfn[node]){
                    if (node==1) ++son; else cutPoint[node]=true;
```

```
                }
            } else lowlink[node]=min(lowlink[node],dfn[g[node][i]]);
        }
}
```

1.3.2 无向图割边

dfn[node]表示节点node访问的时间，lowlink[node]表示节点node不通过父子边能访问到的访问时间最早的节点的访问时间
对于u->v这条边，若lowlink[v]>dfn[u]，那么这就是一条割边

```
void dfs(int node){
    ++sign;
    dfn[node]=lowlink[node]=sign;
    for (int i=0; i<g[node].size(); i++)
        if (!vis[g[node][i].id]){
            vis[g[node][i].id]=true;
            if (dfn[g[node][i].y]==0){
                 dfs(g[node][i].y);
                 lowlink[node]=min(lowlink[node],lowlink[g[node][i].y]);
                 if (lowlink[g[node][i].y]>dfn[node])
                     cout<<g[node][i].id<<endl;
            } else lowlink[node]=min(lowlink[node],dfn[g[node][i].y]);
        }
}
```

1.3.3 树的同构

【题目】给出一棵n个节点的树，用m种颜色对其节点染色，问有多少种不同的染色方案。
【思路】首先要处理子树的同构问题，对于任意自同构，质心是不动点（质心：直径的中点，如果直径长度是奇数，则质心在边上）。
设$f(T)$表示以树T本质不同的染色数，设r的儿子中共有k个同构等价类，分别为$T_1, T_2, \cdots, T_k$，数量分别为$c_1, c_2, \cdots, c_k$。对于任意等价类分别计算$f(T_i)$，$f(T)$等于不同构的子树染色数的乘积，对于每一个等价类T_i，一共有c_i个，那么对于这c_i个等价类的染色数等价于用$f(T_i)$种颜色对c_i个物品染色，这个问题等价于方程$x_1 + x_2 + \cdots + x_{f(T_i)} = c_i$的方案数$C(f(T_i) + c_i - 1, c_i)$。
对于子树的同构判断，我们可以计算每一棵子树T_i的hash值。
设H_i为树T_i的hash值，树T有k棵子树，分别为$T_1, T_2, \cdots, T_k$，那么树T的hash值$H(T) = (((((a * p\mathbf{xor}H_1) modq) * p\mathbf{xor}H_2) modq) \cdots * p\mathbf{xor}H_k) modq) * b modq$，其中$a, b, p, q$为常量，$H_i$需要从小到大排序。

1.3.4 树的分治

```
/*
* active[]表示当前子树的节点
* nodeList存储当前子树的节点
```

```
* visit[]表示已经用作根个节点(即已经被计算过)
* data存储当前子树到根的路径的长度以及该路径属于哪一棵子树
* 算法的具体思想就是找当前子树的中心，以中心为根节点，统计子树通过根节点
* 连接起来的长度<=limit的路径。然后再统计删除该根节点后个棵子树的情况，一直递归。
* 时间复杂度O(NlogN)，空间复杂度O(n)
*/

#include <vector>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;

struct node_data{
    int node,len;
    node_data(int node=0, int len=0):node(node),len(len){}
};

const int MAXN=100001;

int n,limit;
bool active[MAXN],visit[MAXN];
int total[MAXN];
vector<int> nodeList;
vector<node_data> g[MAXN];
vector< pair<int,int> > data;

void init(){
    int x,y,len;
    memset(g,0,sizeof(g));
    for (int i=1; i<n; i++){
        scanf("%d%d%d",&x,&y,&len);
        --x,--y;
        g[x].push_back(node_data(y,len));
        g[y].push_back(node_data(x,len));
    }
}

int newRoot(int node, int fa, int &root, int &nowMax, int totalNode){
    int maxSize=0,sum=1;
    for (int i=0; i<g[node].size(); i++)
    if (g[node][i].node!=fa && active[g[node][i].node]){
        int temp=newRoot(g[node][i].node,node,root,nowMax,totalNode);
        maxSize=max(maxSize,temp);
        sum+=temp;
    }
    maxSize=max(maxSize,totalNode-sum);
```

```
    if (maxSize<nowMax){
        root=node;
        nowMax=maxSize;
    }
    return sum;
}

void getSubtree(int node, int fa, int len, int &part){
    for (int i=0; i<g[node].size(); i++){
        if (active[g[node][i].node] && g[node][i].node!=fa){
            if (g[node][i].len+len>limit) continue;
            data.push_back(make_pair(g[node][i].len+len,part));
            getSubtree(g[node][i].node,node,g[node][i].len+len,part);
        }
    }
}

void getNodeList(int node, int fa){
    active[node]=true;
    nodeList.push_back(node);
    for (int i=0; i<g[node].size(); i++)
    if (g[node][i].node!=fa && !visit[g[node][i].node])
    getNodeList(g[node][i].node,node);
}

long long getAnswer(int node, int fa){
    nodeList.clear();
    getNodeList(node,fa);
    int root,nowMax=n;
    newRoot(node,fa,root,nowMax,nodeList.size());

    int part=0;
    data.clear();
    for (int i=0; i<g[root].size(); i++)
    if (active[g[root][i].node] && g[root][i].len<=limit){
        data.push_back(make_pair(g[root][i].len,part));
        getSubtree(g[root][i].node,root,g[root][i].len,part);
        ++part;
    }
    sort(data.begin(),data.end());

    long long ret=data.size();
    int le=-1,ri=data.size()-1;
    while (ri>0){
        while (le+1<ri && data[le+1].first+data[ri].first<=limit){
            ++le;
            ++total[data[le].second];
        }
```

```
        while (le>=ri) --total[data[le--].second];
        ret+=le+1-total[data[ri].second];;
        --ri;
    }
    for (int i=0; i<part; i++) total[i]=0;
    for (int i=0; i<nodeList.size(); i++) active[nodeList[i]]=false;

    visit[root]=true;
    for (int i=0; i<g[root].size(); i++)
    if (!visit[g[root][i].node])
    ret+=getAnswer(g[root][i].node,root);
    return ret;
}

void solve(){
    memset(visit,0,sizeof(visit));
    cout<<getAnswer(0,-1)+n<<endl;
}

int main(){
    int SIZE=1<<23;
    char *p=(char*)malloc(SIZE)+SIZE;
    __asm__("movl %0, %%esp\n" :: "r"(p));
    while (scanf("%d%d",&n,&limit)!=EOF){
        init();
        solve();
    }
    return 0;
}
```

1.4 哈密尔顿回路

求一个图的哈密尔顿回路是一个NP问题，只能用搜索解决。当定点数n比较小的时候，可以用状态压缩Dp解决。当n比较大的时候，只能用搜索解决。但是，关于哈密尔顿回路，有一个性质：

Ore性质：对所有不邻接的不同顶点对x和y，有

$$deg(x) + deg(y) \geq n$$

那么这个图一定存在哈密尔顿回路，且可以用一下方法求回路，时间复杂度接近$O(n^2)$

1)从任意一个顶点开始，在它的任意一端邻接一个顶点，构造一条越来越长的路径，直到不能再加长为止。设路径为

$$\gamma : y_1 - y_2 - \cdots - y_m$$

2)检查y_1和y_m是否邻接。

. a)如果y_1和y_m不邻接，则转到3，否则，y_1和y_m是邻接的，转到b。

. b) 如果$m = n$，则停止构造并输出哈密尔顿回路$y_1 - y_2 - \cdots - y_m - y_1$，否则，转到c。

. c) 找出一个不在γ上的顶点z和在γ上的顶点y_k，满足z和y_k是邻接的，将γ用下面的长度为$m + 1$的路径来替代

$$z - y_k - \cdots - y_m - y_1 - \cdots - y_{k-1}$$

. 转到2)

3) 找出一个顶点$y_k(1 < k < m)$，满足y_1和y_k是邻接的，且y_{k-1}和y_m也是邻接的，将γ用下面的路径来替代

$$y_1 - \cdots - y_{k-1} - y_m - \cdots - y_k$$

. 转到2)

1.5 最近公共祖先算法

1.5.1 Tarjan

```
/*
 * the node is 1-base
*/
#include <vector>
#include <cstdio>
#include <cstring>
using namespace std;

const int MAXN=100001;
const int MAXM=100001;

struct query{
    int node,id;
    query(int node=0, int id=0):node(node),id(id){}
};

int n,m;
bool visit[MAXN];
int lca[MAXM];
int father[MAXN];
vector<int> g[MAXN];
vector<query> q[MAXN];

void init(){
    int x,y;
    memset(g,0,sizeof(g));
    scanf("%d",&n);
    for (int i=1; i<n; i++){
        scanf("%d%d",&x,&y);
        g[x].push_back(y);
        g[y].push_back(x);
    }
```

```
}

int find(int k){
    return father[k]!=k?father[k]=find(father[k]):k;
}

void dfs(int node, int fa){
    visit[node]=true;
    for (int i=0; i<g[node].size(); i++)
        if (g[node][i]!=fa){
            dfs(g[node][i],node);
            father[g[node][i]]=node;
        }

    for (int i=0; i<q[node].size(); i++)
        if (visit[q[node][i].node]){
            lca[q[node][i].id]=find(q[node][i].node);
        }
}

void solve(){
    int x,y;
    scanf("%d",&m);
    memset(q,0,sizeof(q));
    for (int i=0; i<m; i++){
        scanf("%d%d",&x,&y);
        q[x].push_back(query(y,i));
        q[y].push_back(query(x,i));
    }
    memset(visit,0,sizeof(visit));
    for (int i=1; i<=n; i++) father[i]=i;
    dfs(1,-1);
    for (int i=0; i<m; i++) printf("%d\n",lca[i]);
}

int main(){
    init();
    solve();
    return 0;
}
```

1.5.2 祖先树

```
#include <vector>
#include <cstdio>
#include <cstring>
#include <iostream>
using namespace std;
```

```
const int MAXN=100001;
const int maxDeep=17;


/*
 * The ancestor-tree can use to calc LCA with O(logN) Complexity
 * also maintain some information from node i to i's ancestor
 * deep[node] is the distance from root to node
 * father[node][i] is the 2^i ancestor of node
*/

int n,m;
int deep[MAXN],father[MAXN][maxDeep];
vector<int> g[MAXN];

void init(){
    scanf("%d",&n);
    memset(g,0,sizeof(g));
    int x,y;
    for (int i=0; i<n-1; i++){
        scanf("%d%d",&x,&y);
        g[x].push_back(y);
        g[y].push_back(x);
    }
}

void dfs(int node, int fa){
    //update father[node][] use the information calc before
    for (int i=1; i<maxDeep; i++)
        father[node][i]=father[father[node][i-1]][i-1];

    for (int i=0; i<g[node].size(); i++)
        if (g[node][i]!=fa){
            deep[g[node][i]]=deep[node]+1;
            father[g[node][i]][0]=node;
            dfs(g[node][i],node);
        }
}

int LCA(int x, int y){
    //choose the farther one from root and jump up until x,y has
    //the same distance from root.
    if (deep[x]<deep[y]) swap(x,y);
    int delta=deep[x]-deep[y];
    for (int i=0; i<maxDeep; i++)
        if (delta & (1<<i)) x=father[x][i];
```

```
    //when the x's,y's (2^i)-th father is different,both of them
    //should jump up for 2^i step
    for (int i=maxDeep-1; i>=0; i--)
        if (father[x][i]!=father[y][i]){
            x=father[x][i],y=father[y][i];
        }

    //there are two situtation at last
    return x!=y?father[x][0]:x;
}

void solve(){
    memset(father,0,sizeof(father));
    deep[1]=0;
    dfs(1,0);

    scanf("%d",&m);
    int x,y;
    for (int i=0; i<m; i++){
        scanf("%d%d",&x,&y);
        printf("%d\n",LCA(x,y));
    }
}

int main(){
    init();
    solve();
    return 0;
}
```

1.6 字符串算法

1.6.1 字符串循环节

对于一个长度为n的字符串，枚举n的因子d，然后判断d是否为循环节。只要字符串$\mathrm{st[1..n-d]}$和$\mathrm{st[d+1..n]}$相等，则d为循环节，这一个判断可以用hash来完成，时间复杂度为$\mathrm{O(1)}$。根据循环节判断的性质，求出KMP的next数组之后(假设下标从1开始)，那么这个字符串的最小循环长度为$n - next[n]$。

对于字符串的循环节还有这样一个规律: 如果$\frac{n}{i}$和$\frac{n}{j}$是循环节，且i, j互质，那么$\frac{n}{i*j}$也是循环节。

1.6.2 KMP

```
/*
str为主串,mode为模式串,返回匹配的位置从0开始计数。
next[i]表示匹配到模式串第i个位置失败后应该尝试匹配next[i]这个位置
*/
void calc_next(string &st){
    int i=0,j=-1; next[0]=-1;
```

```
    while (i<st.length()){
        if (j<0 || st[i]==st[j]){
            ++i;
            ++j;
            next[i]=j;
        } else j=next[j];
    }
}

void KMP(string &str, string &mode){
    calc_next(mode);
    int i=0,j=0,modeLen=mode.size();
    while (i<str.length()){
        if (j<0 || str[i]==mode[j]){
            ++i;
            ++j;
        } else j=next[j];

        if (j>=modeLen){
            cout<<i-mode.length()<<endl;
            j=next[j];
        }
    }
}
```

1.6.3 扩展KMP

```
/*
str是主串,mode是模式串。
next[i]表示以模式串第i个位置开始的后缀和模式串的最长公共前缀的长度。
extend[i]表示主串第i个位置开始的后缀和模式串的最长公共前缀。
*/

void calc_next(string &st){
    int prev,pos,j=-1;
    next[0]=st.length();
    for (int i=1; i<st.length(); i++,j--){
        if (j<0 || i+next[i-prev]>=pos){
            if (j<0) j=0,pos=i;
            while (pos<st.length() && st[pos]==st[j]){
                ++pos,++j;
            }
            next[i]=j,prev=i;
        } else next[i]=next[i-prev];
    }
}

void getExtend(string &str, string &mode){
```

```
    calc_next(mode);
    int prev,pos,j=-1;
    for (int i=0; i<str.length(); i++,--j){
        if (j<0 || i+next[i-prev]>=pos){
            if (j<0) j=0,pos=i;
            while (pos<str.length() && j<mode.length() && str[pos]==mode[j]){
                ++pos,++j;
            }
            extend[i]=j,prev=i;
        } else extend[i]=next[i-prev];
    }
}
```

1.6.4 回文串 Manacher 算法

Manacher算法的主要思想就是利用对称关系，记录一个对称轴以及回文串长度+对称轴的最大位置，然后算出以当前字符为中心的最长回文串。

```
string longestPalindrome(string s) {
    string str = "#";
    for (int i = 0; i < s.size(); i++){
        str += s[i];
        str += '#';
    }
    int maxLen[str.size()];
    int maxPos = 0, index = 0;
    for (int i = 0; i < str.size(); i++){
        if (maxPos > i){
            maxLen[i] = min(maxLen[2 * index - i], maxPos - i);
        } else {
            maxLen[i] = 0;
        }
        while (i - maxLen[i] - 1 >= 0 && i + maxLen[i] + 1 < str.size() &&\
        str[i - maxLen[i] - 1] == str[i + maxLen[i] + 1]) ++maxLen[i];
        if (i + maxLen[i] > maxPos){
            maxPos = i + maxLen[i];
            index = i;
        }
    }
    int maxPalindromeLen = 1;
    index = 0;
    for (int i = 0; i < str.size(); i++) if (maxLen[i] >= maxPalindromeLen){
        maxPalindromeLen = maxLen[i];
        index = i;
    }
    string palindromeString = "";
   for (int i = index - maxPalindromeLen; i <= index + maxPalindromeLen; i++) if (str[i] != '#')
    return palindromeString;
}
```

1.6.5 最小表示法

```
#include <string>
#include <cstring>
#include <iostream>
using namespace std;

string st;

void init(){
    cin>>st;
}

int calc_min_pos(string &st){
    int i=0,j=1,k=0,diff,len=st.size();
    while (i<len && j<len && k<len){
        diff=st[(i+k)%len]-st[(j+k)%len];
        if (diff==0) ++k; else{
            if (diff>0) i+=k+1; else j+=k+1;
            if (i==j) ++j;
            k=0;
        }
    }
    return min(i,j);
}

void solve(){
    cout<<calc_min_pos(st)<<endl;
}

int main(){
    init();
    solve();
    return 0;
}
```

1.6.6 后缀数组

Suffix基本步骤:
1、对字符串出现的字符进行排序，如果字符串出现的不同字母个数相对较少就用基数排序，在字符串的最后加上一个字典序最小的字母，这样可以防止例如AAAAA的时候，Suffix无法计算出Rank。如果遇到多个字符串，一般用不同且未出现过的字符隔开。

2、计算SA、Rank以及height数组。

3、后缀数组最常用到height数组的性质，height[i]表示排名为i和i-1的后缀的最长公共前缀。

Suffix题目类型:
1、求两个串后缀的最长公共前缀：对height做一次RMQ处理，然后就可以在O(1)时间内回答。

2、可重叠最长重复字串：由于可重复，所以直接求height数组中的最大值即可。

3、不可重叠最长重复字串(pku1743)：二分字串长度len，把height中连续>=len的分在一个组，然后在这个组找到最大、最小的Sa[i],Sa[j]如果Sa[i]-Sa[j]>=len就有解。

4、可重叠的k次最长重复字串(pku3261)：同样二分字串长度len，把height中连续>=len的分在一个组，然后判断这个组元素个数是否不小于k。

5、不同字串的个数(spoj705)：每个子串一定是某个后缀的前缀，那么原问题等价于求所有后缀之间的不相同的前缀的个数。如果所有的后缀按照suffix(sa[1])，suffix(sa[2]),...，suffix(sa[n])的顺序计算，不难发现，对于每一次新加进来的后缀suffix(sa[k]),它将产生n-sa[k]+1个新的前缀。但是其中有height[k]个是和前面的字符串的前缀是相同的。所以suffix(sa[k])将“贡献”出n-sa[k]+1-height[k]个不同的子串。累加后便是原问题的答案。这个做法的时间复杂度为O(n)。

6、一个串的最长回文字串(pku3774)：把原串反过来加在原串后，用未出现字符隔开，然后对height做RMQ处理，然后问题就变成求两个串某两个后缀的最长公共字串。

7、两个串的最长连续公共字串：用一个未出现的符号连接两串，然后求满足Sa[i],Sa[i-1]属于不同串的height[i]的最大值。

8、长度不小于k 的公共子串的个数(pku3415):二分答案len,然后把height分组，看是否存在一组height值>=len且出现在每个串。

```
/*
 * sa[i]表示排名为i的后缀在原串的起始位置为sa[i](从1开始计数)
 * rank[i]表示以第i个位置开始的后缀在所有后缀中的排名(从1开始计数)
 * sa[rank[i]]==i、rank[sa[i]]==i
 * height[i]表示排名为i的后缀与排名为i-1的后缀的最长公共前缀
*/

#include <string>
#include <cstring>
#include <iostream>
using namespace std;

const int MAXN=100001;
const int maxChar=256;

int rank[MAXN],rank1[MAXN],rank2[MAXN],sa[MAXN],
    tmpsa[MAXN],height[MAXN],countrank[MAXN];

void suffixArray(string &st){
    int tot=0,len=st.size();
    int number[maxChar];
```

```
bool appear[maxChar];
memset(appear,0,sizeof(appear));
memset(number,0,sizeof(number));
for (int i=0; i<len; i++) appear[st[i]]=true;
for (int i=0; i<maxChar; i++)
    if (appear[i]) number[i]=++tot;

memset(countrank,0,sizeof(countrank));
for (int i=1; i<=len; i++){
    rank[i]=number[st[i-1]];
    ++countrank[rank[i]];
}
for (int i=1; i<=maxChar; i++) countrank[i]+=countrank[i-1];
for (int i=len; i>=1; i--) sa[countrank[rank[i]]--]=i;

int l=1;
while (l<=len){
    for (int i=1; i<=len; i++){
        rank1[i]=rank[i];
        if (i+l<=len) rank2[i]=rank[i+l]; else
            rank2[i]=0;
    }
    memset(countrank,0,sizeof(countrank));
    for (int i=1; i<=len; i++) ++countrank[rank2[i]];
    for (int i=1; i<=len; i++) countrank[i]+=countrank[i-1];
    for (int i=len; i>=1; i--) tmpsa[countrank[rank2[i]]--]=i;

    memset(countrank,0,sizeof(countrank));
    for (int i=1; i<=len; i++) ++countrank[rank1[i]];
    for (int i=1; i<=len; i++) countrank[i]+=countrank[i-1];
    for (int i=len; i>=1; i--) sa[countrank[rank1[tmpsa[i]]]--]=tmpsa[i];

    rank[sa[1]]=1;
    for (int i=2; i<=len; i++){
        rank[sa[i]]=rank[sa[i-1]];
        if (!(rank1[sa[i]]==rank1[sa[i-1]]
              && rank2[sa[i]]==rank2[sa[i-1]]))
            ++rank[sa[i]];
    }
    l+=l;
}

l=0; height[1]=0;
for (int i=1; i<=len; i++)
    if (rank[i]>1){
        int j=sa[rank[i]-1];
        while (i+l<=len && j+l<=len && st[i+l-1]==st[j+l-1]) ++l;
        height[rank[i]]=l;
```

```
            if (l) --l;
        } else height[rank[i]]=0;
}
```

1.6.7 后缀自动机

1、后缀自动机的的节点分为接受态和非接受态两种，接受态表示这个节点可以接收当前字符串的后缀，即从出发点走到接受态的节点的路径 (字符串) 是当前字符串的后缀从。从结束态 (最后一个字符) 沿着 parent 一直走到初始状态，所经过的节点就是接受态的节点。

2、构造字符串 S 的后缀自动机，从起始点出发，到达任意一个接受态的路径是字符串 S 的后缀，从起始点到一个接受态 a，最多不会超过 n 条路径 (表示最多有 n 个后缀)，对于两个不同的接受态 a 和 b，从起始点出发到 a，b 的路径集合要么是包含关系，要么是不相交。

3、从起始点出发，到达任意一个顶点 a 的路径 p 是字符串 S 的一个子串，后缀自动机保证了从起始点出发到达任意点的路径都是不同的。假设从起始点出发，到达一个节点 a_i 的路径数 n_i，那么字符串 S 的所有不同的子串的个数为 $\sum_{a_i} n_i$。

4、假设从出发点到一个节点 a 的所有路径为 $\{p_i\}$，那么对于任意 $len(p_i) < len(p_j)$，p_i 是 p_j 的后缀。

5、构造完后缀自动机之后，要解决问题一般可以 dfs 遍历自动机，或者用桶排序根据自动机的每个节点的 maxL 构造出拓扑图，然后在拓扑图上求解。

题目：

- 1. 给出字符串 S 以及 q 个询问，每个询问给出一个字符串 sub，问字符串 sub 在 S 出现的次数。
  - 从起始点出发，按照 sub 的字母顺序走到状态 a，假设状态 a 有 m 条不同的路径能走到某个接受态，那么 sub 在 S 中出现的次数为 m。
- 2. 给出字符串 S_1,S_2，求这两个串的最长公共子串。
  - 首先用 S_1 构造后缀自动机，然后按照 S_2 的字母顺序在自动机上遍历。
  - 假设当前自动机上的节点为 a，对于 $S_2[i]$，如果 a 有 $S_2[i]$ 这个儿子，那么当前的长度 maxL + 1，将 a 转移到 $a->ch[S_2[i]]$。如果 a 没有 $S_2[i]$ 这个儿子，那么沿着 a 的 parent 往走，走到 a 为空或者 a 存在一个 $S_2[i]$ 的儿子节点为止，当 a 存在一个 $S_2[i]$ 的儿子，将 maxL 设为 $a->ch[S_2[i]]->maxL$，否则 $maxL = 0$。
  - 每走一步都尝试更新最长公共字串的答案。
- 3. 求多个字符串 $S_1, S_2, \cdots, S_n$ 的最长公共子串。
  - 首先选一个字符串 S_1 构造后缀自动机。
  - 对于其他 S_i，按照字母顺序在自动机上遍历，走到自动机上的一个状态 a，更新这个状态的 $maxL$ 得到最大值，设为 $maxL_i[a]$。
  - 因为从起始点出发，到达同一个节点 a 的路径两两之间的有后缀关系，所以对于某个某个状态 a，$S_2, \cdots, S_n$ 走到这个状态的 $maxL$ 的最小值就是走到这个状态的所有串的最长公共子串。所以 n 个串的最长公共字串的值就是 $max(min(maxL_i[a])), (2 \le i \le n)$。

以下程序为给出n个串，求出他们的最长公共子串的长度。

```
#include <stdio.h>
#include <string.h>

const int MAX_N = 100001;

struct SAMnode{
    int maxL;
    SAMnode *ch[26], *parent;
};

struct SAM{
    int cnt;
    SAMnode *start, *end;
    SAMnode pool[MAX_N * 2];

    void init(){
        cnt = 0;
        start = end = &pool[cnt++];
    }

    SAMnode &operator [](const int &id){
        return pool[id];
    }

    void push_back(int c, int maxL){
        SAMnode *np = &pool[cnt++], *p = end;
        np->maxL = maxL;
        for (; p && !p->ch[c]; p = p->parent) p->ch[c] = np;
        end = np;
        if (!p){
            np->parent = start;
        } else{
            if (p->ch[c]->maxL == p->maxL + 1){
                np->parent = p->ch[c];
            } else{
                SAMnode *q = &pool[cnt++], *r = p->ch[c];
                *q = *r;
                q->maxL = p->maxL + 1;
                np->parent = r->parent = q;
                for (; p && p->ch[c] == r; p = p->parent) p->ch[c] = q;
            }
        }
    }
} sam;

char str[MAX_N];
```

```
int n;
int f[MAX_N * 2], g[MAX_N * 2], v[MAX_N], q[MAX_N * 2];

void init(){
    scanf("%s", str);
    sam.init();
    n = strlen(str);
    for (int i = 0; i < n; i++) sam.push_back(str[i] - 'a', i + 1);
}

void updata(int &x, int y){
    if (x < y) x = y;
}

void getans(char *str){
    memset(g, 0, sizeof(g));
    int n = strlen(str);
    SAMnode *p = sam.start;
    int maxL = 0;
    for (int i = 0; i < n; i++){
        int c = str[i] - 'a';
        if (p->ch[c]){
            p = p->ch[c];
            ++maxL;
            updata(g[p - sam.pool], maxL);
            SAMnode *pp = p->parent;
        } else{
            while (p && !p->ch[c]) p = p->parent;
            if (!p) maxL = 0; else maxL = p->maxL + 1;
            if (!p) p = sam.start; else p = p->ch[c];
            updata(g[p - sam.pool], maxL);
            SAMnode *pp = p->parent;
        }
    }

    for (int i = sam.cnt; i >= 1; i--){
        SAMnode *p = &sam[i];
        if (p->parent) updata(g[p->parent - sam.pool], g[p - sam.pool]);
    }
}

int min(int a, int b){
    return a < b?a:b;
}

void solve(){
    for (int i = 0; i < sam.cnt; i++) v[sam[i].maxL]++;
    for (int i = 1; i <= n; i++) v[i] += v[i - 1];
```

```
    for (int i = 0; i <= sam.cnt; i++) q[v[sam[i].maxL]--] = i;
    int ret = 0;
    for (int i = 0; i < sam.cnt; i++) f[i] = sam[i].maxL;
    while (~scanf("%s", str)){
        getans(str);
        for (int i = 0; i < sam.cnt; i++) f[i] = min(f[i], g[i]);
    }
    for (int i = 0; i < sam.cnt; i++) updata(ret, f[i]);
    printf("%d\n", ret);
}

int main(){
    init();
    solve();
    return 0;
}
```

2 数据结构

2.1 Splay

```
/*
* Splay 多数用来维护一段序列，通常这一段序列又需要 reverse 操作，或者删除一段子序列等
* Splay 任意节点 x 提升为节点 y 的儿子，一般可以在这个操作的基础上完成其他操作
* 例如要删除一段范围 [x,y] 的序列，那么可以把 x-1 这个节点提升为根，再把 y+1 这个节点提升
* 为 x-1 的儿子（右儿子），然后 y+1 这个节点的左儿子就是 [x,y] 这一段序列
*
* Splay 的操作通常都是把一段范围的序列变为某个节点的一棵子树，然后就可以对这棵子树
* 进行操作
*
* 因为空序列不好处理，所以 Splay 一般都会加入两个不影响结果的辅助节点（头、尾），至于
* 怎样不影响结果，要根据题目来设定。
*
* Splay 提升某个节点的时候，有两种情况。一种是先调用 select 得到节点 x，然后再提升 x，这样
* 在 splay 操作时，就不需要 pushdown 标签，因为在 select 过程中已经 pushdown 了。
* 但如果是已知某个节点 x，需要 splay 提升，这是在 splay 提升的过程中，就要注意 pushdown 标
签了
* 在对 Splay 操作时，要时刻注意是否要进行 pushdown 操作
*
*
* root ： SplayTree 的根节点，要注意时刻维护!!!
* build(data,l,r,fa) ： 如果初始时给出序列，那么可以用 build 来建立一棵平衡的 SplayTree
* rotate(x) ： 将节点 x 往上旋转（左右旋转都没问题）
* splay(x,fa) ： 将节点 x 提升为 fa 的儿子（当 fa=NULL 即将 x 提升为根）
* select(x,k) ： 一般调用为 select(root,k)，即获得第排名为 k 的节点，在这个过程
* 中需要先 pushdown 下传标签
* join(x,y) ： 合并 x，y 两棵 SplayTree，其中 y 在 x 的右边。首先将 x 最右的节点提升为
```

* x的根，此时x无右子树，所以直接将y连接为x的右子树
* del(k)：删除排名为k的节点。首先将第k个节点提升为根，然后直接合并根的两棵
* 子树，注意任意时刻Splay都最少保持有2个节点（开始加入的2个节点）
* insert(key,k)：在第k个节点前插入一个key，首先将节点k-1提升为根，然后将节点k提升
* 为k-1的儿子，此时节点k无左子树，然后将key插入到k的左子树中。
* 要在第k个节点之前插入一段连续的数的操作和一个数差不多。
*/

题目：reginoal 2012 HangZhou A problem

1: add x

Starting from the arrow pointed element, add x to the number on the clockwise first k2 elements.

2: reverse

Starting from the arrow pointed element, reverse the first k1 clockwise elements.

3: insert x

Insert a new element with number x to the right (along clockwise) of the arrow pointed element.

4: delete

Delete the element the arrow pointed and then move the arrow to the right element.

5: move x

x can only be 1 or 2. If x = 1 , move the arrow to the left(along the counterclockwise) element, if x = 2 move the arrow to the right element.

6: query

Output the number on the arrow pointed element in one line.

```
#include <stdio.h>
#include <string.h>
#include <algorithm>
using namespace std;

#define REP(i,st,ed) for (int i=st; i<ed; i++)

const int MAXN = 102400;

struct splayNode{
    splayNode *fa,*ch[2];
    bool rev;
    int size,key,add;
    splayNode(int key=0){
        this->key=key;
        size=1,add=0,rev=false;
        fa=ch[0]=ch[1]=NULL;
    }
    void clear(){
        if (ch[0]) ch[0]->clear();
        if (ch[1]) ch[1]->clear();
        delete this;
    }
    void update(){
```

```
        size=(ch[0]?ch[0]->size:0)+(ch[1]?ch[1]->size:0)+1;
    }
    void down(){
        if (add){
            if (ch[0]) ch[0]->key+=add,ch[0]->add+=add;
            if (ch[1]) ch[1]->key+=add,ch[1]->add+=add;
            add=0;
        }
        if (rev){
            swap(ch[0],ch[1]);
            if (ch[0]) ch[0]->rev^=1;
            if (ch[1]) ch[1]->rev^=1;
            rev=false;
        }
    }
    splayNode* rightmost(){
        down();
        if (ch[1]) return ch[1]->rightmost();
        return this;
    }
} *root,*temp;

typedef splayNode* ptr;

int n,m,k1,k2,data[MAXN];

ptr build(int data[], int l, int r, ptr fa=NULL){
    int mid=(l+r)/2;
    ptr node=new splayNode(data[mid]);
    node->fa=fa;
    if (l<mid) node->ch[0]=build(data,l,mid-1,node);
    if (mid<r) node->ch[1]=build(data,mid+1,r,node);
    node->update();
    return node;
}

void rotate(ptr x){
    ptr y=x->fa;
    if (x->fa=y->fa) y->fa->ch[y->fa->ch[1]==y]=x;
    int o=y->ch[0]==x;
    if (y->ch[!o]=x->ch[o]) x->ch[o]->fa=y;
    x->ch[o]=y;
    y->fa=x;
    y->update();
    x->update();
}

void splay(ptr x, ptr fa){
```

```
    while (x->fa!=fa){
        /* this condition are need sometime
         * if (x->fa->fa) x->fa->fa->down();
         * if (x->fa) x->fa->down();
         * x->down();
        */
        if (x->fa->fa==fa) rotate(x); else{
            int o1=x->fa->ch[0]==x,o2=x->fa->fa->ch[0]==x->fa;
            if (o1==o2) rotate(x->fa),rotate(x); else rotate(x),rotate(x);
        }
    }
}

ptr select(ptr x, int k){
    x->down();
    if (x->ch[0]){
        if (x->ch[0]->size>=k) return select(x->ch[0],k);
        k-=x->ch[0]->size;
    }
    if (k==1) return x;
    return select(x->ch[1],k-1);
}

ptr join(ptr x, ptr y){
    ptr temp=x->rightmost();
    splay(temp,NULL);
    temp->ch[1]=y;
    y->fa=temp;
    temp->update();
    return temp;
}

void del(int k){
    root=select(root,k);
    splay(root,NULL);
    root->ch[0]->fa=root->ch[1]->fa=NULL;
    root=join(root->ch[0],root->ch[1]);
}

void insert(int key, int k){
    root=select(root,k-1);
    splay(root,NULL);
    temp=select(root,k);
    splay(temp,root);
    temp->ch[0]=new splayNode(key);
    temp->ch[0]->fa=temp;
    temp->update();
    root->update();
```

```
}

void init(){
    REP(i,1,n+1) scanf("%d",&data[i]);
    root=build(data,0,n+1);
}

void solve(){
    char st[10];
    int number;
    REP(i,0,m){
        scanf("%s",st);
        if (st[0]=='a'){
            scanf("%d",&number);
            root=select(root,1);
            splay(root,NULL);
            temp=select(root,k2+2);
            splay(temp,root);
            temp->ch[0]->key+=number;
            temp->ch[0]->add+=number;
        } else if (st[0]=='r'){
            root=select(root,1);
            splay(root,NULL);
            temp=select(root,k1+2);
            splay(temp,root);
            root->ch[1]->ch[0]->rev^=1;
        } else if (st[0]=='i'){
            scanf("%d",&number);
            insert(number,3);
        } else if (st[0]=='d'){
            del(2);
        } else if (st[0]=='m'){
            scanf("%d",&number);
            if (number==1){
                temp=select(root,root->size-1);
                int key=temp->key;
                del(root->size-1);
                insert(key,2);
            } else{
                temp=select(root,2);
                int key=temp->key;
                del(2);
                insert(key,root->size);
            }
        } else if (st[0]=='q'){
            root=select(root,2);
            splay(root,NULL);
            printf("%d\n",root->key);
```

```
        }
    }
    root->clear();
}

int main(){
    int ca=0;
    while (scanf("%d%d%d%d",&n,&m,&k1,&k2)){
        if (n==0 && m==0 && k1==0 && k2==0) return 0;
        printf("Case #%d:\n",++ca);
        init();
        solve();
    }
    return 0;
}
```

2.2 左偏树

```
/*
 * 左偏树应用快速合并两个堆
 * 左偏树的节点有4个信息，分别是key，到最近的外部节点的距离dist
 * 左右子树的指针left，right
 *
 * 左偏树的性质:
 * 1、节点的key小于等于左右子节点的key
 * 2、节点的左子树的dist不小于右子树的dist
 * 3、节点的dist等于其右子树的dist+1，特别地空子树的dist=-1
 * 4、一棵N个节点的左偏树的距离最多为log(N+1)-1
 *
 * 左偏树的操作是建立在merge之上，合并两棵左偏树A,B，以最小堆为例:
 * 1、若A为空树，则返回B，若B为空树，返回A
 * 2、如果A的key大于B的key，则交换A，B，然后继续合并
 * 3、将B和A的右子树合并，如果合并之后A的右子树的dist大于左子树的
 * dist， 则交换A的左右子树。若合并后A的右子树为空，额A的dist为0，
 * 否则A的dist等于右子树的dist+1
 *
 * 左偏树的操作:
 * 1、插入：相当于合并两棵左偏树
 * 2、删除根节点：合并根节点的左又子树
 * 3、查询最值：根节点
 *
 * 给出N个节点，构建一棵N个节点的左偏树。
 * 1、首先把所有节点加入到队列里面，每一个节点都可以看作一棵左偏树
 * 2、每次选队列最前端的两个节点合并，把合并后的左偏树放到队尾。
 * 3、这样合并N-1次就可以得到一棵左偏树，时间复杂度为O(N)
 *
 * 用指针实现的左偏树，在实现清除操作时，要注意指针所指的数据不能被
 * delete多次，否则会出现运行错误
```

```
 *
 * 左偏树的操作的时间复杂度:
 * 构建         : O(N)
 * 插入         : O(logN)
 * 查询最值        : O(1)
 * 合并         : O(logN)
*/

//以下是用指针实现的根的key为最小值的左偏树
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;

struct leftistNode{
    int key,dist;
    leftistNode *left,*right;
    leftistNode(int key=0):key(key){
        dist=0;
        left=right=NULL;
    }
};

leftistNode *mergeNode(leftistNode *a, leftistNode *b){
    if (a==NULL) return b;
    if (b==NULL) return a;
    if (a->key>b->key) swap(a,b);
    a->right=mergeNode(a->right,b);
    if (a->right!=NULL){
        if (a->left==NULL || a->left->dist<a->right->dist)
            swap(a->left,a->right);
    }
    if (a->right==NULL){
        a->dist=0;
    } else{
        a->dist=a->right->dist+1;
    }
    return a;
}

struct leftistTree{
    leftistNode *root;

leftistTree():root(NULL){}

    void insert(leftistNode *rhs){
        root=mergeNode(root,rhs);
    }
```

```
    void insert(int key){
        leftistNode *temp=new leftistNode(key);
        insert(temp);
    }

    void del(){
        if (root==NULL) return;
        leftistNode *temp=mergeNode(root->left,root->right);
        delete root;
        root=temp;
    }

    void _clear(leftistNode *root){
        if (root==NULL) return;
        if (root->left!=NULL) _clear(root->left);
        if (root->right!=NULL) _clear(root->right);
        delete root;
    }

    void clear(){
        _clear(root);
        root=NULL;
    }

    leftistNode *askMax(){
        return root;
    }
};

leftistTree merge(leftistTree a, leftistTree b){
    leftistTree ret;
    ret.root=mergeNode(a.root,b.root);
    return ret;
}

/*
 * 非指针版的实现就是事先开一个buffer存储数据
 * 以下是zoj2334的程序
*/

#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;

const int MAXN=100001;
```

```
struct leftistNode{
    int key,dist,left,right;

    leftistNode(int key=0):key(key){
        dist=0;
        left=right=0;
    }
};

leftistNode buffer[MAXN];

int mergeNode(int A, int B){
    if (!A) return B;
    if (!B) return A;
    if (buffer[A].key<buffer[B].key) swap(A,B);
    buffer[A].right=mergeNode(buffer[A].right,B);
    if (!buffer[B].left ||
        buffer[buffer[A].left].dist<buffer[buffer[A].right].dist)
            swap(buffer[A].left,buffer[A].right);
    if (!buffer[A].right){
        buffer[A].dist=0;
    } else buffer[A].dist=buffer[B].dist+1;
    return A;
}

struct leftistTree{
    int root;
    leftistTree(int root=0):root(root){}

    void insert(int pos){
        root=mergeNode(root,pos);
    }

    int del(){
        int temp=root;
        root=mergeNode(buffer[root].left,buffer[root].right);
        return temp;
    }

    int top(){
        return root;
    }
};

leftistTree merge(leftistTree &a, leftistTree &b){
    return leftistTree(mergeNode(a.root,b.root));
}
```

```
int n;
int number[MAXN],father[MAXN];
leftistTree heap[MAXN];

void init(){
    for (int i=1; i<=n; i++) scanf("%d",&number[i]);
}

int find(int k){
    int x,y=k;
    while (father[y]!=y) y=father[y];
    while (k!=y){
        x=father[k];
        father[k]=y;
        k=x;
    }
    return y;
}

void solve(){
    int x,y,m,total=0;
    for (int i=1; i<=n; i++){
        father[i]=i;
        buffer[++total]=leftistNode(number[i]);
        heap[i].root=i;
    }
    scanf("%d",&m);
    for (int i=0; i<m; i++){
        scanf("%d%d",&x,&y);
        x=find(x),y=find(y);
        if (x==y){
            printf("%d\n",-1);
        } else{
            father[y]=x;
            heap[x]=merge(heap[x],heap[y]);
            int pos=heap[x].del();
            buffer[pos]=leftistNode(buffer[pos].key/2);
            printf("%d\n",buffer[pos].key);
            heap[x].insert(pos);
        }
    }
}

int main(){
    while (scanf("%d",&n)!=EOF){
        init();
        solve();
    }
```

```
    return 0;
}
```

2.3 树链剖分

```
/*
 * hson[i]表示i这个节点所连接的那条重边对应的子节点编号
 *
 * top[i]表示i这个节点所属的那条链的链头节点编号
 *
 * idx[i]表示编号唯i的节点在线段树中的编号，一条链在线段树上的编号是连续的
 *
 * 树链剖分的步骤:
 * 1、遍历整棵树，找子树节点个数最大的子节点作为主链
 * 2、遍历轻重边，把原树上的节点与其在线段树的节点对应
 * 记录每条链在线段树中的位置以及链头所链接的父亲节点
 * 3、在询问路径时，每次两个节点深度较大的网上跳，直到两个节点在同一条链上为止。
 *
 * 每次询问的时间复杂度为O((logN)^2)
 *
 * 本程序的例题：询问路径中节点的最大值。把某个节点的值
 * 增加一个值。
 *
*/
#pragma comment(linker, "/STACK:16777216")

#include <cstdio>
#include <cstring>
#include <vector>
#include <algorithm>
using namespace std;

const int MAX_N = 100001;

int n,totalSize;
int size[MAX_N],top[MAX_N],depth[MAX_N],idx[MAX_N],hson[MAX_N],parent[MAX_N];
int tree[MAX_N * 4];
vector<int> g[MAX_N];

void init(){
    int x,y;
    scanf("%d",&n);
    for (int i = 1; i <= n; i++) g[i].clear();
    for (int i = 1; i < n; i++){
        scanf("%d%d",&x,&y);
        g[x].push_back(y);
```

```
        g[y].push_back(x);
    }
}

void dfs(int node){
    size[node] = 1, hson[node] = -1;
    for (int i = 0; i < g[node].size(); i++) if (g[node][i] != parent[node]){
        int son = g[node][i];
        depth[son] = depth[node] + 1;
        parent[son] = node;
        dfs(son);
        size[node] += size[son];
        if (hson[node] == -1 || size[son] > size[hson[node]]) hson[node] = son;
    }
}

void build(int node, int tp){
    idx[node] = ++totalSize;
    top[node] = tp;
    if (hson[node] != -1) build(hson[node], tp);
    for (int i = 0; i < g[node].size(); i++){
        int son = g[node][i];
        if (son != hson[node] && son != parent[node]) build(son,son);
    }
}

void change(int x, int y, int state, int L, int R, int value){
    if (L > y || x > R) return;
    if (L <= x && y <= R){
        tree[state] += value;
        return;
    }

    int mid = (x + y) / 2;
    change(x, mid, state * 2, L, R, value);
    change(mid + 1, y, state * 2 + 1, L, R, value);
    tree[state] = max(tree[state * 2], tree[state * 2 + 1]);
}

int ask(int x, int y, int state, int L, int R){
    if (L > y || x > R) return 0;
    if (L <= x && y <= R) return tree[state];

    int mid = (x + y) / 2;
    return max(ask(x, mid, state * 2, L, R), ask(mid + 1, y, state * 2 + 1, L, R));
}

int query_max(int x, int y){
```

```
    int tpx = top[x], tpy = top[y], ret = 0;
    while (tpx != tpy){
        if (depth[tpx] < depth[tpy]) swap(tpx,tpy), swap(x,y);
        ret = max(ret, ask(1, n, 1, idx[tpx], idx[x]));
        x = parent[tpx], tpx = top[x];
    }
    if (depth[x] > depth[y]) swap(x,y);
    ret = max(ret, ask(1, n, 1, idx[x], idx[y]));
    return ret;
}

void solve(){
    dfs(1);
    totalSize = 0;
    memset(tree,0,sizeof(tree));
    build(1,1);

    char st[3];
    int m,x,y,value;
    scanf("%d",&m);
    for (int i = 0; i < m; i++){
        scanf("%s",st);
        if (st[0] == 'I'){
            scanf("%d%d",&x,&value);
            change(1, n, 1, idx[x], idx[x], value);
        } else{
            scanf("%d%d",&x,&y);
            printf("%d\n",query_max(x,y));
        }
    }
}

int main(){
    init();
    solve();
    return 0;
}
```

2.4 可持久化数据结构

2.4.1 可持久化线段树

持久化数据结构最重要的思想是无论询问还是插入操作，都不修改原来的数据，而是新建立节点。

题目描述

1. C l r d: Adding a constant d for every $A_i(l \leq i \leq r)$, and increase the time t add by 1, this is the only operation that will cause the time increase.

2. Q l r: Querying the current sum of $A_i(l \leq i \leq r)$.

3. H l r t: Querying a history sum of $A_i(l \leq i \leq r)$ in time t.

4. B t: Back to time t. And once you decide return to a past, you can never be access to a forward edition anymore.

程序

```
#include <cmath>
#include <cstdio>
#include <cstring>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;

const int MAXN=100001;
const int MAXLIMIT=500000;

struct data{
    long long add;
    long long sum;
    data *lt,*rt;
    data(){}
    data(data *lt, data *rt, long long sum, int add=0):lt(lt),rt(rt),sum(sum),add(add){}
    inline long long getSum(int len){
        if (!this) return 0;
        return sum+add*len;
    }
};

data buffer[MAXLIMIT];
data* tree[MAXN];
int n,m,total,number;
int record[MAXN];

data *build(int l, int r){
    if (l==r){
        scanf("%d",&number);
        data *node=&buffer[total++];
        node->lt=node->rt=NULL;
        node->sum=number; node->add=0;
        return node;
    }

    int mid=(l+r)/2;
    data *node=&buffer[total++];
    node->lt=build(l,mid);
    node->rt=build(mid+1,r);
    node->sum=node->lt->getSum(mid-l+1)+
        node->rt->getSum(r-mid);
```

```
    node->add=0;
    return node;
}

data *insert(int l, int r, data *node, int L, int R, long long Add, int add){
    if (L>r || l>R){
        if (Add!=0){
            buffer[total]=data(*node);
            buffer[total].add+=Add;
            return &buffer[total++];
        }
        return node;
    }
    if (L<=l && r<=R){
        buffer[total]=data(*node);
        buffer[total].add+=Add+add;
        return &buffer[total++];
    }

    int mid=(l+r)/2;
    data *lt=insert(l,mid,node->lt,L,R,
        Add+node->add,add);

    data *rt=insert(mid+1,r,node->rt,L,R,
        Add+node->add,add);

    long long sum=lt->getSum(mid-l+1)+
        rt->getSum(r-mid);

    buffer[total++]=data(lt,rt,sum,0);
    return &buffer[total-1];
}

long long ask(int l, int r, data *node, int L, int R, long long Add){
    if (L>r || l>R) return 0;
    if (L<=l && r<=R){
        return node->getSum(r-l+1)+Add*(r-l+1);
    }

    int mid=(l+r)/2;
    long long lt=ask(l,mid,node->lt,L,R,
        Add+node->add);

    long long rt=ask(mid+1,r,node->rt,L,R,
        Add+node->add);

    return lt+rt;
}
```

```
void init(){
    total=0;
    tree[0]=build(1,n);
}

void solve(){
    char opt[2];
    int L,R,add;
    int cur=0,Time;
    record[cur]=total;
    for (int i=0; i<m; i++){
        scanf("%s",opt);
        if (opt[0]=='C'){
            scanf("%d%d%d",&L,&R,&add);
            tree[cur+1]=insert(1,n,tree[cur],L,R,0,add);
            record[++cur]=total;
        } else
        if (opt[0]=='H'){
            scanf("%d%d%d",&L,&R,&Time);
            long long temp=ask(1,n,tree[Time],L,R,0);
            printf("%I64d\n",temp);
        } else
        if (opt[0]=='Q'){
            scanf("%d%d",&L,&R);
            long long temp=ask(1,n,tree[cur],L,R,0);
            printf("%I64d\n",temp);
        } else{
            scanf("%d",&cur);
            total=record[cur];
        }
    }
}

int main(){
    while (scanf("%d%d",&n,&m)!=EOF){
        init();
        solve();
    }
    return 0;
}
```

2.5 字典树

2.5.1 Trie树

```
struct Trie{
    int cnt, head;
    char ch[MAX_M];
    int g[MAX_M], next[MAX_M];

    void init(){
        head = 1;
```

```
        cnt = 1;
    }

    inline int getnext(int p){
        return next[p];
    }

    int &operator[](const int &idx){
        return g[idx];
    }

    int getson(int node, char c){
        int p = g[node];
        while (p && ch[p] != c) p = next[p];
        return p;
    }

    int insert(int node, char c){
        ++cnt;
        g[cnt] = 0;
        ch[cnt] = c;
        next[cnt] = g[node];
        g[node] = cnt;
        return cnt;
    }
} trie;
```

2.5.2 Trie图

```
/*
trie图用于多模式串匹配,一般的题目都是要求生成的字符串中不包含一些字符串
建立trie图,然后从根节点开始走,只要不走到危险节点就可以了
注意: trie图的2个性质: 1、一个节点的危险性和它后缀节点相同 2、根节点的直接儿子的后缀是根节
点
*/
#include <string>
#include <queue>
#include <cstdio>
#include <cstring>
#include <iostream>
using namespace std;

class Trie{
    public:
        int tot;
        const static int MAXN=100;

    public:
        char ch[MAXN];
        bool danger[MAXN];
```

```
        int g[MAXN],next[MAXN],suffix[MAXN];

    public:
        Trie(){
            tot=1;
            memset(g,255,sizeof(g));
            memset(danger,0,sizeof(danger));
        }
        int getson(int node, char c, bool flag);
        void insert(string st);
        void makegraph();
};

int Trie::getson(int node, char c, bool flag){
    while (true){
        int now=g[node];
        while (now!=-1 && ch[now]!=c) now=next[now];
        if (now!=-1 || !flag) return now;
        if (node==1) return node;
        node=suffix[node];
    }
}

void Trie::insert(string st){
    int len=st.size();
    int now=1,son;
    for (int i=0; i<len; i++){
        son=getson(now,st[i],false);
        if (son==-1){
            ++tot;
            ch[tot]=st[i];
            next[tot]=g[now];
            g[now]=tot;
            now=tot;
        } else now=son;
    }
    danger[now]=true;
}

void Trie::makegraph(){
    queue<int> q;
    q.push(1);
    while (!q.empty()){
        int node=q.front(); q.pop();
        for (int p=g[node]; p!=-1; p=next[p]){
            q.push(p);
            if (node==1){
                suffix[p]=1;
            } else{
                suffix[p]=getson(suffix[node],ch[p],true);
            }
```

```
            danger[p]|=danger[suffix[p]];
        }
    }
}

int main(){
    string st;
    Trie graph;
    cin>>st;
    graph.insert(st);
    cin>>st;
    graph.insert(st);
    return 0;
}
```

2.6 划分树

```
{
划分树支持静态询问,询问方式为区间[le,ri]中的第k大/小的数
}
Const  maxn=100000;

var
  n,m:longint;
  a:array[0..maxn] of longint;
  toLeft,value:array[0..trunc(ln(maxn)/ln(2))+1,1..maxn] of longint;

procedure qsort(l,r:longint);
var
  i,j,m:longint;
begin
  i:=l; j:=r; m:=a[(i+j) shr 1];
  repeat
    while a[i]<m do inc(i);
    while a[j]>m do dec(j);
    if i<=j then
    begin
      a[0]:=a[i];
      a[i]:=a[j];
      a[j]:=a[0];
      inc(i); dec(j);
    end;
  until i>j;

  if l<j then qsort(l,j);
  if i<r then qsort(i,r);
end;

procedure init;
var
```

```
  i:longint;
begin
  readln(n,m);
  for i:=1 to n do read(a[i]);
        for i:=1 to n do value[0,i]:=a[i];
  qsort(1,n);
end;

procedure BuildTree(x,y,deep:longint);
var
  i,mid,le,ri,same,lsame:longint;
begin
  if x=y then exit;
  mid:=(x+y) shr 1; lsame:=mid-x+1;
  for i:=x to y do
    if value[deep][i]<a[mid] then dec(lsame);
  le:=x; ri:=mid+1; same:=0;
  for i:=x to y do
  begin
    if i=x then toLeft[deep,i]:=0 else toLeft[deep,i]:=toLeft[deep,i-1];
    if value[deep][i]<a[mid] then
    begin
      inc(toLeft[deep,i]);
      value[deep+1,le]:=value[deep,i];
      inc(le);
    end else
    if value[deep][i]>a[mid] then
    begin
      value[deep+1,ri]:=value[deep][i];
      inc(ri);
    end else
    begin
      if same<lsame then
      begin
        inc(toLeft[deep,i]);
        value[deep+1,le]:=value[deep,i];
        inc(le); inc(same);
      end else
      begin
        value[deep+1,ri]:=value[deep,i];
        inc(ri);
      end;
    end;
  end;

  BuildTree(x,mid,deep+1);
  BuildTree(mid+1,y,deep+1);
end;

function ask(le,ri,x,y,deep,kth:longint):longint;
var
```

```
  s1,s2,mid:longint;
begin
  if x=y then exit(value[deep,x]);
  mid:=(x+y) shr 1;
  if x=le then
  begin
    s1:=0;
    s2:=toLeft[deep,ri];
  end else
  begin
    s1:=toLeft[deep,le-1];
    s2:=toLeft[deep,ri];
  end;
  if s2-s1>=kth then
  begin
    le:=x+s1; ri:=le+s2-s1-1;
    exit(ask(le,ri,x,mid,deep+1,kth));
  end else
  begin
    kth:=kth-(s2-s1);  s2:=ri-le+1-(s2-s1);
    le:=mid+le-x+1-s1; ri:=le+s2-1;
    exit(ask(le,ri,mid+1,y,deep+1,kth));
  end;
end;

procedure main;
var
  i,kth,le,ri:longint;
begin
  BuildTree(1,n,0);
  for i:=1 to m do
  begin
    readln(le,ri,kth);
    writeln(ask(le,ri,1,n,0,kth));
  end;
end;

begin
  init;
  main;
end.
```

3 计算几何

3.1 计算几何基础

3.1.1 基本定理

1、$\triangle ABC$ 的外接圆半径 $R = abc/4S$，其中 a, b, c, S 分别表示三角形的三条边的边长以及三角形的面积。这个等式可以用正弦定理证明。

2、圆内接四边形的对角互补，设四条边的长度分别为 a, b, c, d，半周长 $p = (a + b + c + d)/2$，则内接四边

形的面积 $S=\sqrt{(p-a)(p-b)(p-c)(p-d)}$。其外接圆半径 $R=\frac{\sqrt{(ac+bd)(ad+bc)(ab+cd)}}{4S}$。在等边长的四边形中，圆内接四边形的面积最大。

3、所有的正多边形都有外接圆，外接圆的圆心和正多边形的中心重合。变长为 a 的正 n 边形外接圆的半径 $R_n=\frac{a}{2\mathbf{sin}(\frac{\pi}{n})}$，面积 $S=\pi R_n^2=\frac{\pi a^2}{4\mathbf{sin}(\frac{\pi}{n})^2}$。

3.1.2 坐标旋转

将 (x_1,y_1) 旋转 β，则得到的坐标 (x_2,y_2) 为 $(x_1*cos\beta-y_1*sin\beta,x_1*sin\beta+y_1*cos\beta)$

3.1.3 pick 定理

(整数格子点简单多边形的面积、内部点数、边界点数关系)：
设 F 为平面上以格子点为定点的单纯多边形，则其面积为：S=b/2+i-1。
b 为多边形边上点格点的个数，i 为多边形内部格点的个数。
可用其计算多边形的面积，边界格点数或内部格点数。

3.1.4 点到线段之间的距离

首先利用三角形底边与面积关系求出高(点到直线距离)，然后用点乘求出向量 $\overrightarrow{AB}$ 与向量 $\overrightarrow{BC}$ 的夹角，如果为钝角，则高在三角形内，否则在三角形外，如果在三角形外的话，那么点到线段的距离必然是到两端点的距离最短者。

```
double PointLineDist(point C, point A, point B){
        double ret=(B-A)*(C-A)/sqrt((B-A)^(B-A));
        if (((C-B)^(B-A))>0) return sqrt((B-C)^(B-C));
        if (((C-A)^(A-B))>0) return sqrt((A-C)^(A-C));
        return abs(ret);
}
```

3.1.5 线段和直线方程

```
struct point{
    double x,y;
};

point operator -(const point &a, const point &b){
    point ret;
    ret.x=a.x-b.x; ret.y=a.y-b.y;
    return ret;
}

/* 叉乘 A*B=|A||B|cos(<A,B>)若A叉乘B<0则A在B的左手边 */
double operator *(const point &a, const point &b){
    return a.x*b.y-a.y*b.x;
}

/* 点乘 A^B=|A||B|sin(<A,B>) */
double operator ^(const point &a, const point &b){
```

```
    return a.x*b.x+a.y*b.y;
}

/* 获取线段ab的中点 */
point midpoint(point &a, point &b){
    point ret;
    ret.x=(a.x+b.x)/2; ret.y=(a.y+b.y)/2;
    return ret;
}

/* 计算经过点a和点b的直线的方程Ax+By=C */
void getABC(double &A, double &B, double &C, point &a, point &b){
    A=b.y-a.y;
    B=a.x-b.x;
    C=A*a.x+B*a.y;
}

/* 计算线段ab的中垂线，直线ab的方程为Ax+By=C */
void getmidline(double &A, double &B, double &C, point &a, point &b){
    swap(A,B);
    A=-A;
    point p=midpoint(a,b);
    C=A*p.x+B*p.y;
}

/* 计算直线A1x+B1y=C1与直线A2x+B2y=C2的交点 */
point getIntersection(double A1, double B1, double C1,
                      double A2, double B2, double C2){
    double det=A2*B1-A1*B2;
    if (abs(det)<eps){
        /* 两直线平行 */
    }

    point ret;
    ret.x=(B1*C2-B2*C1)/det;
    ret.y=(A2*C1-A1*C2)/det;
    return ret;
}
```

3.1.6 两圆交点

```
#include <cmath>
#include <cstdio>
#include <iostream>
using namespace std;

const double eps=1e-8;

struct point{
    double x,y;
};
```

```
struct Circle{
    point C;
    double R;
};

double sqr(double x){
    return x*x;
}

double Distance2(point &a, point &b){
    return sqr(a.x-b.x)+sqr(a.y-b.y);
}

void Circle_Intersection(Circle A, Circle B){
    double Dist=sqrt(Distance2(A.C,B.C));
    if (Dist-A.R-B.R>eps || fabs(A.R-B.R)-Dist>eps){
        //no Intersection
        cout<<"No Intersection"<<endl;
        return;
    }
    double a=2*A.R*(A.C.x-B.C.x);
    double b=2*A.R*(A.C.y-B.C.y);
    double c=B.R*B.R-A.R*A.R-Distance2(A.C,B.C);
    double p=a*a+b*b;
    double q=-2*a*c;
    double r=c*c-b*b;
    double cos1=(-q+sqrt(q*q-4*p*r))/(2*p);
    double cos2=(-q-sqrt(q*q-4*p*r))/(2*p);
    double sin1=sqrt(1-cos1*cos1);
    double sin2=sqrt(1-cos2*cos2);

    point ret1,ret2;
    ret1.x=cos1*A.R+A.C.x; ret1.y=sin1*A.R+A.C.y;
    ret2.x=cos2*A.R+A.C.x; ret2.y=sin2*A.R+A.C.y;
    if (fabs(Distance2(ret1,B.C)-B.R*B.R)>eps)
        ret1.y=-sin1*A.R+A.C.y;
    if (fabs(Distance2(ret2,B.C)-B.R*B.R)>eps)
        ret2.y=-sin2*A.R+A.C.y;
    //the same point
    if (fabs(Distance2(ret1,ret2))<eps){
        ret1=ret2;
    }
}
```

3.1.7 两圆公切线

```
int main(){
    /*
    double X1,Y1,X2,Y2,R1,R2;
    (X - X1)^2 + (Y - Y1)^2 = R1^2
```

```
    (X - X2)^2 + (Y - Y2)^2 = R2^2
    delta1 < 0 and delta2 < 0 : 内含，无公切线
    delta1 < 0 and delta2 = 0 : 内切，有一条切线L3=L4
    delta1 < 0 and delta2 > 0 : 相交，L3,L4
    delta1 = 0 and delta2 > 0 : 外切，L1=L2,L3,L4
    delta1 > 0 and delta2 > 0 : 外离，L1,L2,L3,L4
    圆在L1,L2切线两旁，在L3,L4切线同一旁
    输出的参数为a,b,c，直线是ax + by + c = 0
    */
    cin>>X1>>Y1>>R1>>X2>>Y2>>R2;
    double delta1 = sqr(X1 - X2) + sqr(Y1 - Y2) - sqr(R1 + R2);
    double delta2 = sqr(X1 - X2) + sqr(Y1 - Y2) - sqr(R1 - R2);
    double p1 = R1 * (X1 * X2 + Y1 * Y2 - sqr(X2) - sqr(Y2));
    double p2 = R2 * (sqr(X1) + sqr(Y1) - X1 * X2 - Y1 * Y2);
    double q = X1 * Y2 - X2 * Y1;
    //L1
    A1 = (X2 - X1) * (R1 + R2) + (Y1 - Y2) * sqrt(delta1);
    B1 = (Y2 - Y1) * (R1 + R2) + (X2 - X1) * sqrt(delta1);
    C1 = p1 + p2 + q * sqrt(delta1);

    //L2
    A2 = (X2 - X1) * (R1 + R2) - (Y1 - Y2) * sqrt(delta1);
    B2 = (Y2 - Y1) * (R1 + R2) - (X2 - X1) * sqrt(delta1);
    C2 = p1 + p2 - q * sqrt(delta1);

    //L3
    A3 = (X2 - X1) * (R1 - R2) + (Y1 - Y2) * sqrt(delta2);
    B3 = (Y2 - Y1) * (R1 - R2) + (X2 - X1) * sqrt(delta2);
    C3 = p1 - p2 + q * sqrt(delta2);

    //L4
    A4 = (X2 - X1) * (R1 - R2) - (Y1 - Y2) * sqrt(delta2);
    B4 = (Y2 - Y1) * (R1 - R2) - (X2 - X1) * sqrt(delta2);
    C4 = p1 - p2 + q * sqrt(delta2);
    return 0;
}
```

3.2 圆与简单多边形面积交

```
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <map>
#include <set>
#include <queue>
#include <stack>
#include <vector>
#include <sstream>
#include <iostream>
```

```
#include <algorithm>
using namespace std;

const int MAXN=128;
const double eps=1e-8;
const double pi=acos(-1.0);

struct point{
    double x,y;
    point(){
        x=0;
        y=0;
    }
    point(double x,double y):x(x),y(y){}
    bool operator ==(point b)const{
        return x==b.x && y==b.y;
    }
};

struct polygon{
    int n;
    point points[MAXN];
    int size(){
        return n;
    }
    point& operator [](int i){
        return points[i];
    }
    void resize(int x){
        n=x;
    }
};

inline double length(point a){
    return sqrt(a.x*a.x+a.y*a.y);
}

inline point operator +(point a,point b){
    return point(a.x+b.x,a.y+b.y);
}

inline point operator -(point a,point b){
    return point(a.x-b.x,a.y-b.y);
}

inline point operator *(point a,double t){
    return point(a.x*t,a.y*t);
}

inline point operator /(point a,double t){
    return point(a.x/t,a.y/t);
```

```
}

inline double operator *(point a,point b){
    return a.x*b.x+a.y*b.y;
}

inline double operator ^(point a,point b){
    return a.x*b.y-a.y*b.x;
}

inline double angle(point a,point b){
    double ans=fabs(atan2(a.y,a.x)-atan2(b.y,b.x));
    if (ans>pi) ans=pi*2-ans;
    return ans;
}

const point no_solution(0,0);

point intersection_to_circle(point p,point q,double r){
    point u=q-p;
    double A=u*u;
    double B=2*(p*u);
    double C=p*p-r*r;
    double delta=B*B-4*A*C;
    if (delta<-eps) return no_solution;
    else if (fabs(delta)<eps){
        double t=-B/(2*A);
        if (t<-eps || t>1.0+eps)
            return no_solution;
        return p+u*t;
    } else{
        double t1=(-B-sqrt(delta))/(2.0*A);
        double t2=(-B+sqrt(delta))/(2.0*A);
        double t;
        bool flag1=(t1>-eps && t1<1.0+eps);
        bool flag2=(t2>-eps && t2<1.0+eps);
        if (!flag1 && !flag2) return no_solution;
        else if (!flag1) t=t2;
        else t=t1;
        return p+u*t;
    }
}

point gravity_center(polygon p){
    point ans(0,0);
    double area=0;
    for (int i=0;i<p.size();i++){
        double now_area=p[i]^p[i+1];
        point now_center=(p[i]+p[i+1])/3.0;
        area+=now_area;
        ans=ans+now_center*now_area;
```

```
    }
    ans=ans/area;
    return ans;
}

double intersection_area(polygon p,point c,double r){
    double ans=0;
    for (int i=0;i<p.size();i++){
        point a=p[i]-c;
        point b=p[i+1]-c;
        double la=length(a);
        double lb=length(b);
        double now_area;
        int now_sign;
        if ((a^b)>0) now_sign=1;
        else now_sign=-1;
        double phi=angle(a,b);
        if (la<r+eps && lb<r+eps){
            now_area=fabs(a^b);
        } else if (la<r+eps){
            point c=intersection_to_circle(b,a,r);
            double alpha=angle(a,c);
            double beta=phi-alpha;
            now_area=la*r*sin(alpha)+r*r*beta;
        } else if (lb<r+eps){
            point c=intersection_to_circle(a,b,r);
            double alpha=angle(b,c);
            double beta=phi-alpha;
            now_area=lb*r*sin(alpha)+r*r*beta;
        } else{
            point c=intersection_to_circle(a,b,r);
            point d=intersection_to_circle(b,a,r);
            if (c==no_solution) now_area=r*r*phi;
            else{
                double alpha=angle(c,d);
                double beta=phi-alpha;
                now_area=r*r*sin(alpha)+r*r*beta;
            }
        }
        now_area*=now_sign;
        ans+=now_area;
    }
    return ans*0.5;
}

point center;
polygon p;
double v0,theta,t,g,R;

void init(){
    double x,y;
```

```
    cin>>p.n;
    for (int i=0; i<p.n; i++){
        cin>>x>>y;
        p.points[i]=point(x,y);
    }
    p.points[p.n]=p.points[0];
}

void solve(){
    printf("%.2lf\n",intersection_area(p,center,R));
}

int main(){
    while (cin>>center.x>>center.y>>R){
        if (!center.x && !center.y && !v0 && !theta && !t && !g && !R) break;
        init();
        solve();
    }
    return 0;
}
```

3.3 最小圆覆盖

```
/*
 * 最小圆覆盖的思想：若第i+1个点不在前i个点的最小圆内，则第i+1个点必然在
 * 前i+1个点的最小圆的边上。
 * 最坏情况下时间复杂度是O(n^3),期望是O(n)
*/

#include <cmath>
#include <cstdio>
#include <iostream>
#include <algorithm>
using namespace std;

const double eps=1e-8;
const int MAXN=1001;

struct point{
    double x,y;

    point(double x=0, double y=0):x(x),y(y){}
    double sqr(double x){ return x*x; }
    double dist2(const point &b){ return sqr(x-b.x)+sqr(y-b.y); }
    double dist(const point &b){ return sqrt(dist2(b)); }
    point operator -(const point &b){ return point(x-b.x,y-b.y); }
    double operator *(const point &b){ return x*b.y-y*b.x; }
};

struct circle{
    point center;
```

```
    double R;

    circle(){}
    circle(const point &center, double R):center(center),R(R){}

    bool inside(const point &p){
        return center.dist(p)<R+eps;
    }
};

int n;
point P[MAXN];

void init(){
    double x,y;
    for (int i=0; i<n; i++){
        scanf("%lf%lf",&x,&y);
        P[i]=point(x,y);
    }
    random_shuffle(P,P+n);
}

circle getCircleWith2Point(point &a, point &b){
    return circle(point((a.x+b.x)/2,(a.y+b.y)/2),a.dist(b)/2);
}

//calculate the area of triangle ABC
double Area(point &a, point &b, point &c){
    return fabs((b-a)*(c-a)/2.0);
}

//Rotate vector 1/2BA to vector BR
circle calc(point &a, point &b, double sinA, double cosA, double scale, double R){
    double x=(b.x-a.x)*scale,y=(b.y-a.y)*scale;
    return circle(point(a.x+cosA*x-sinA*y,a.y+cosA*y+sinA*x),R);
}

/*
 * R=abc/4S, where S is the area of triangle ABC.
 * calculate the vector 1/2BA and rotate it to BR.
 * when we know vector BR, we can calculate point R.
*/

circle getCircleWith3Point(point &a, point &b, point &c){
    double r=a.dist(b)*b.dist(c)*c.dist(a)/(4*Area(a,b,c));
    double halfAB=a.dist(b)/2.0;
    double lenBR=sqrt(r*r-halfAB*halfAB);
    double scale=r/(halfAB*2);
    circle Circle=calc(a,b,lenBR/r,halfAB/r,scale,r);
    if (Circle.inside(c)) return Circle;
    return calc(a,b,-lenBR/r,halfAB/r,scale,r);
```

```
}

/*
 * we know the minimalCircle of point 1..i, consider the (i+1)th point.
 * if the (i+1)th point is in the minimalCircle of point 1..i, consider (i+2)th point,
 * otherwise, the (i+1)th point is on the edge of the minimalCircle of 1..i+1.
 *
 * so we can add point to the minimalCircle one by one and maintain the minimalCircle.
 * when the point add by random order, the exception complexity of the algorithm is O(n)
*/

void solve(){
    circle Circle=getCircleWith2Point(P[0],P[1]);
    for (int i=2; i<n; i++){
        if (Circle.inside(P[i])) continue;
        Circle=getCircleWith2Point(P[0],P[i]);
        for (int j=0; j<i; j++){
            if (Circle.inside(P[j])) continue;
            Circle=getCircleWith2Point(P[i],P[j]);
            for (int k=0; k<j; k++){
                if (Circle.inside(P[k])) continue;
                Circle=getCircleWith3Point(P[i],P[j],P[k]);
            }
        }
    }
    printf("%.2lf %.2lf %.2lf\n",Circle.center.x,Circle.center.y,Circle.R);
}

int main(){
    while (scanf("%d",&n) && n!=0){
        init();
        solve();
    }
    return 0;
}
```

3.4 多边形的核

平面简单多边形的核是该多边形内部的一个点集，该点集中任意一点与多边形边界上一点连线都处于这个多边形内部。

求多边形的核的思想是用原多边形的边所在的直线不断切割多边形。

```
#include <iostream>
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
using namespace std;

const int MAXN=2000;
const double eps=1e-8;
```

```
struct point{
    double x,y;
};

int n,N,ca;
point a[MAXN],b[MAXN],c[MAXN];

void getABC(double &A, double &B, double &C, point &a, point &b){
    A=b.y-a.y;
    B=a.x-b.x;
    C=A*a.x+B*a.y;
}

point getIntersection(double A1, double B1, double C1,
                      double A2, double B2, double C2){
    double det=A2*B1-A1*B2;
    if (abs(det)<eps){
    }
    point ret;
    ret.x=(B1*C2-B2*C1)/det;
    ret.y=(A2*C1-A1*C2)/det;
    return ret;
}

void init(){
    cin>>n;
    for (int i=0; i<n; i++) cin>>a[i].x>>a[i].y;
}

void check(double A, double B, double C){
    int size=0;
    double A1,B1,C1;
    for (int i=1; i<=N; i++)
        if (A*b[i].x+B*b[i].y-C>=-eps){ //点是逆时针的话就<=eps
            c[++size]=b[i];
        } else{
            if (A*b[i-1].x+B*b[i-1].y-C>eps){ //点是逆时针的话就<-eps
                getABC(A1,B1,C1,b[i],b[i-1]);
                c[++size]=getIntersection(A,B,C,A1,B1,C1);
            }
            if (A*b[i+1].x+B*b[i+1].y-C>eps){ //点是逆时针的话就<-eps
                getABC(A1,B1,C1,b[i+1],b[i]);
                c[++size]=getIntersection(A,B,C,A1,B1,C1);
            }
        }
    N=size;
    for (int i=1; i<=N; i++) b[i]=c[i];
    b[0]=b[N]; b[N+1]=b[1];
}

void solve(){
```

```
    double A,B,C;
    N=n;
    for (int i=1; i<=N; i++) b[i]=a[i-1];
    b[0]=b[N]; b[N+1]=b[1]; a[n]=a[0];
    for (int i=0; i<n; i++){
        getABC(A,B,C,a[i],a[i+1]);
        check(A,B,C);
    }

//核的面积
    double ret=0;
    for (int i=1; i<=N; i++)
        ret+=b[i].x*b[i+1].y-b[i].y*b[i+1].x;
    printf("%.2lf\n",fabs(ret/2.0));

//如果N>0则存在核
if (N>0) cout<<"YES"<<endl; else cout<<"NO"<<endl;
}

int main(){
    init();
    solve();
    return 0;
}
```

3.5 矩形切割

```
/*
 * 有n个矩形，求这些矩形的面积并
 * 算法的主要思想是枚举第i个矩形，求出第i个矩形被i+1..n的矩形
 * 切割后剩下的面积是多少，加到最终的答案中
 *
 * 如果题目的每个矩形都有颜色，而且后面的矩形会覆盖前面的矩形，
 * 求每种颜色的面积。这样的题目也可以用矩形切割来做。
*/

//判断两个矩形是否相交，true代表不相交
bool check(int &x1, int &y1, int &x2, int &y2, int &pos){
    return x1>=v[pos].x2 || y1>=v[pos].y2 || x2<=v[pos].x1 || y2<=v[pos].y1;
}

void cut(int x1, int y1, int x2, int y2, int pos){
    while (pos<N && check(x1,y1,x2,y2,pos)) ++pos;

    //已经用v里面的N个矩形切割完
    if (pos==N){
        area+=(x2-x1)*(y2-y1);
        return;
    }

    if (x1<v[pos].x1){
```

```
        cut(x1,y1,v[pos].x1,y2,pos+1);
        x1=v[pos].x1;
    }

    if (y1<v[pos].y1){
        cut(x1,y1,x2,v[pos].y1,pos+1);
        y1=v[pos].y1;
    }

    if (x2>v[pos].x2){
        cut(v[pos].x2,y1,x2,y2,pos+1);
        x2=v[pos].x2;
    }

    if (y2>v[pos].y2){
        cut(x1,v[pos].y2,x2,y2,pos+1);
        y2=v[pos].y2;
    }
}

void solve(){
    area=0;
    for (int i=0; i<N; i++)
        cut(v[i].x1,v[i].y1,v[i].x2,v[i].y2,i+1);
    cout<<area<<endl;
}
```

3.6 矩形面积并

```
/*
 * 求n个矩形的面积并
 * 先离散化x轴坐标和y轴坐标
 * 用平行x轴的扫描线扫描，把x轴的线段插入到当前的线段树里面
 * 线段树记录x轴被覆盖的长度
 * 假设离散化后x坐标有m个，那么线段树记录的下标是[1..m-1]
 * 若下标从0开始计数，那么区间[l,r]的长度为x[r]-x[l-1]
*/

#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;

const int MAXN=100001;

struct segment{
    int x1,x2,y,flag;
    segment(){}
    segment(int x1, int x2, int y, int flag):x1(x1),x2(x2),y(y),flag(flag){}
};
```

```
struct data{
    int cover,len;
};

data tree[MAXN*8];
segment seg[MAXN*2];
int n,tot,totx,toty;
int vx[MAXN*2],vy[MAXN*2];

bool cmp(const segment &a, const segment &b){
    return a.y<b.y;
}

void init(){
    int x1,y1,x2,y2;
    tot=totx=toty=0;
    cin>>n;
    for (int i=0; i<n; i++){
        cin>>x1>>y1>>x2>>y2;
        vx[totx++]=x1,vx[totx++]=x2;
        vy[toty++]=y1,vy[toty++]=y2;
        seg[tot++]=segment(x1,x2,y1,1);
        seg[tot++]=segment(x1,x2,y2,-1);
    }
    sort(vx,vx+totx);
    sort(vy,vy+toty);
    totx=unique(vx,vx+totx)-vx;
    toty=unique(vy,vy+toty)-vy;
    sort(seg,seg+tot,cmp);
}

void update(int l, int r, int state){
    if (!tree[state].cover){
        tree[state].len=tree[state*2].len+tree[state*2+1].len;
    } else tree[state].len=vx[r]-vx[l-1];
}

void insert(int l, int r, int state, int L, int R, int flag){
    if (L>r || l>R) return;
    if (L<=l && r<=R){
        tree[state].cover+=flag;
        update(l,r,state);
        return;
    }

    int mid=(l+r)/2;
    insert(l,mid,state*2,L,R,flag);
    insert(mid+1,r,state*2+1,L,R,flag);
    update(l,r,state);
}
```

```
void solve(){
    memset(tree,0,sizeof(tree));
    long long area=0,t=0;
    for (int i=0; i<toty-1; i++){
        while (t<tot && seg[t].y==vy[i]){
        int L=lower_bound(vx,vx+totx,seg[t].x1)-vx+1,R=lower_bound(vx,vx+totx,seg[t].x2)-vx;
            insert(1,totx-1,1,L,R,seg[t].flag);
            ++t;
        }
        area+=(long long)(tree[1].len)*(vy[i+1]-vy[i]);
    }
    cout<<area<<endl;
}

int main(){
    int ca;
    cin>>ca;
    for (int i=0; i<ca; i++){
        init();
        solve();
    }
    return 0;
}
```

3.7 KD-Tree

3.7.1 程序

查找某个点距离最近的点，基本思想是每次分治把点分成两部分，建议按照坐标规模决定是垂直划分还是水平划分，查找时先往分到的那一部分查找，然后根据当前最优答案决定是否去另一个区间查找。

```
bool Div[MaxN];
void BuildKD(int deep,int l, int r, Point p[])\\`记得备份一下P`
{
    if (l > r) return;
    int mid = l + r >> 1;
    int minX, minY, maxX, maxY;
    minX = min_element(p + l, p + r + 1, cmpX)->x;
    minY = min_element(p + l, p + r + 1, cmpY)->y;
    maxX = max_element(p + l, p + r + 1, cmpX)->x;
    maxY = max_element(p + l, p + r + 1, cmpY)->y;
    Div[mid] = (maxX - minX >= maxY - minY);
    nth_element(p + l, p + mid, p + r + 1, Div[mid] ? cmpX : cmpY);
    BuildKD(l, mid - 1, p);
    BuildKD(mid + 1, r, p);
}

long long res;
void Find(int l, int r, Point a, Point p[])\\`查找`
{
```

```
    if (l > r)  return;
    int mid = l + r >> 1;
    long long dist = dist2(a, p[mid]);
    if (dist > 0)//如果有重点不能这样判断
        res = min(res, dist);
    long long d = Div[mid] ? (a.x - p[mid].x) : (a.y - p[mid].y);
    int l1, l2, r1, r2;
    l1 = l, l2 = mid + 1;
    r1 = mid - 1, r2 = r;
    if (d > 0)
        swap(l1, l2), swap(r1, r2);
    Find(l1, r1, a, p);
    if (d * d < res)
        Find(l2, r2, a, p);
}
```

3.7.2 例题

查询一个点为中心的给定正方形内所有点并删除（2012 金华网赛 A）

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cmath>
#include <queue>
using namespace std;

const int MaxN = 100000;
struct Point
{
    int x,y,r;
    int id;
    bool del;
};

int cmpTyp;
bool cmp(const Point& a,const Point& b)
{
    if (cmpTyp == 0)
        return a.x < b.x;
    else
        return a.y < b.y;
}

int cnt[MaxN];
bool Div[MaxN];
int minX[MaxN],minY[MaxN],maxX[MaxN],maxY[MaxN];
void BuildKD(int l,int r,Point p[])
{
    if (l > r)    return;
```

```
    int mid = l+r>>1;
    cmpTyp = 0;
    minX[mid] = min_element(p+l,p+r+1,cmp)->x;
    maxX[mid] = max_element(p+l,p+r+1,cmp)->x;
    cmpTyp = 1;
    minY[mid] = min_element(p+l,p+r+1,cmp)->y;
    maxY[mid] = max_element(p+l,p+r+1,cmp)->y;

    cnt[mid] = r-l+1;
    cmpTyp = Div[mid] = (maxX[mid]-minX[mid] < maxY[mid]-minY[mid]);
    nth_element(p+l,p+mid,p+r+1,cmp);
    BuildKD(l,mid-1,p);
    BuildKD(mid+1,r,p);
}

queue<int> Q;
int Find(int l,int r,Point a,Point p[])
{
    if (l > r)    return 0;
    int mid = l+r>>1;
    if (cnt[mid] == 0)    return 0;

    if (maxX[mid] < a.x-a.r ||
            minX[mid] > a.x+a.r ||
            maxY[mid] < a.y-a.r ||
            minY[mid] > a.y+a.r)
        return 0;

    int totdel = 0;

    if (p[mid].del == false)
        if (abs(p[mid].x-a.x) <= a.r && abs(p[mid].y-a.y) <= a.r)
        {
            p[mid].del = true;
            Q.push(p[mid].id);
            totdel++;
        }

    totdel += Find(l,mid-1,a,p);
    totdel += Find(mid+1,r,a,p);

    cnt[mid] -= totdel;

    return totdel;
}

Point p[MaxN],tp[MaxN];
int n;

int main()
{
```

```
    int cas = 1;
    while (true)
    {
        scanf("%d",&n);
        if (n == 0)    break;

        for (int i = 0;i < n;i++)
        {
            p[i].id = i;
            int tx,ty;
            scanf("%d%d%d",&tx,&ty,&p[i].r);
            p[i].x = tx-ty;
            p[i].y = tx+ty;
            p[i].del = false;
            tp[i] = p[i];
        }
        BuildKD(0,n-1,tp);

        printf("Case #%d:\n",cas++);
        int q;
        scanf("%d",&q);
        for (int i = 0;i < q;i++)
        {
            int id;
            scanf("%d",&id);
            int res = 0;
            id--;
            Q.push(id);
            while (!Q.empty())
            {
                int now = Q.front();
                Q.pop();
                if (p[now].del == true)    continue;
                p[now].del = true;
                res += Find(0,n-1,p[now],tp);
            }
            printf("%d\n",res);
        }
    }
    return 0;
}
```

4 数学

4.1 结论

1、费马欧拉素数定理：每个可表示为 $4n+1$ 形式的素数，只能用一种两数平方的形式来表示。

2、任意一个非负整数都能分解为 4 个平方数之和。设 $N = L * m^2$，如果 L 的质因子中没有 $4k+3$ 这样的形

式，那么 N 就能分解成两个平方数之和，如果 N 不是 $(8t+7)*4^k$ 的形式，则 N 能分解成 3 个平方数之和。

3、勾股数 a,b,c 满足 $a^2+b^2=c^2$，若 $(a,b)=1,(a,c)=1,(b,c)=1$ 则 (a,b,c) 为最原始的勾股数，对于任意一个正数系数 k，(ka,kb,kc) 也是勾股数。构造最原始的勾股数可以令 $a=m^2-n^2,b=2nm,c=n^2+m^2$，其中 $(n,m)=1$ 且 n,m 的奇偶性不同。

4.2 扩展 GCD

考虑这样一个不定方程:

$$ax+by=c$$

易知方程有解的充分必要条件是 $(a,b)\mid c$
根据 gcd 求解的特点，我们可以构造这样一个等式

$$ax+by=bx'+\left(a-b\left\lfloor\frac{a}{b}\right\rfloor\right)y'=c$$

于是有

$$a(x-y')+b\left(y-x'+\left\lfloor\frac{a}{b}\right\rfloor y'\right)=0$$

所以

$$x=y' \quad , \quad y=x'-\left\lfloor\frac{a}{b}\right\rfloor y'$$

是方程 $ax+by=c$ 的一个解

扩展 gcd 递归求解，当 $b=0$ 时，令 $x=\frac{c}{a},y=0$ 是 $ax+by=c$ 的一组特解，然后根据上述等式，我们可以求出方程的一般解。令 x,y 是方程 $ax+by=c$ 的一组特解，且 $d=gcd(a,b)$，则方程 $ax+by=c$ 的一般解为

$$X=x+\frac{b}{d}t,Y=y-\frac{a}{d}t \quad |t\in\mathbf{Z}$$

我们用 b^{-1} 表示 b 的逆元。因为 $\frac{1}{b}\equiv c(mod \quad p)\Rightarrow bx-py=c$ 于是我们可以利用扩展 gcd 可以求出 b 的逆元 b^{-1}，使得 $\frac{a}{b}\equiv c(mod \quad p)\Rightarrow ab^{-1}\equiv c(mod \quad p)$

程序:

```
#include<cstdio>
#include<cstring>
#include<iostream>
using namespace std;

int a,b,c,x,y;

void init(){
  cin>>a>>b>>c;
}

void extend_gcd(int a, int b, int &d, int &x, int &y){
    if (!b){
        d=a; x=1; y=0;
    } else{
        extend_gcd(b,a%b,d,y,x);
        y-=x*(a/b);
    }
```

```
}

void solve(){
  //求解ax+by=c
  //方程有解的充要条件是gcd(a,b) | c
  //求出方程的一个解x,y,然后对于任意正整数t,
  //方程的一般解可以表示为X=x+b/d*t , Y=y-a/d*t
  extend_gcd(a,b,d,x,y);
  if (c%d!=0) cout<<"No solution"<<endl; else
              cout<<"x="<<x<<" "<<"y="<<y<<endl;
}

int main(){
  init();
  solve();
  return 0;
}
```

4.3 中国剩余定理

考虑同余方程组

$$\begin{cases} x \equiv a_1 (mod \quad m_1) \\ x \equiv a_2 (mod \quad m_2) \\ \cdots\cdots \\ x \equiv a_k (mod \quad m_k) \end{cases}$$

其中

$$(m_i, m_j) = 1 \quad , \quad 1 \le i, j \le k, i \ne j$$

设

$$M = m_1 m_2 \cdots m_k \quad , \quad Mi = \frac{M}{m_i}$$

因为 $(Mi, m_i) = 1$，所以必然存在 Mi^{-1} 使得

$$MiMi^{-1} \equiv 1 (mod \quad m_i)$$

$$MiMi^{-1} \equiv 0 (mod \quad m_j), 1 \le j \le k, j \ne i$$

则

$$x = \sum_{i=1}^{k} a_i MiMi^{-1}$$

是同余方程组模 M 的唯一解，其中 Mi^{-1} 可用扩展 gcd 求得。

注：同余方程组有解的充要条件是 $a_i \equiv a_j (mod \quad (m_i, m_j)), 1 \le i, j \le k$

程序：

```
int main(){
  cin>>n;
  for (int i=1; i<=n; i++) cin>>a[i]>>m[i];
  int x , y , M=1 , ans=0;
  //要求m[i]两两互质,同余方程组解唯一
  for (int i=1; i<=n; i++) M*=m[i];
  for (int i=1; i<=n; i++){
    int Mi=M/m[i];
```

```
    //求逆元Mi的逆元Mi'
    extended_gcd(Mi,m[i],x,y);
    ans+=a[i]*Mi*x;
  }
  //ans+M*t是一般解
  ans=(ans%M+M)%M;
  cout<<ans<<endl<<endl;
  return 0;
}
```

4.4 组合数取模

如果模数为素数，直接求逆元就可以了。否则就把模数因式分解成 $\prod_{i=1}^{m} p_i^{k_i}$，求出组合数模 p^{k_i} 的答案，然后用中国剩余定理把这些解合并。如果要求 t 个组合数数 $C(n,m)$ 的对模 Mod 的模，那么总的时间复杂度是 $O(t(logn)^2)$ 级别的。

```
#include <cstring>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;

const int MAXN=100001;

struct data{
    int Pi,P;
    data(){}
    data(int Pi, int P):Pi(Pi),P(P){}
};

struct com{
    int n,m;
    com(){}
    com(int n, int m):n(n),m(m){}
};

int n,m,ca,Mod;
vector<com> ask;
long long factorial[MAXN],power[2];

void init(){
    int x,y;
    scanf("%d%d%d",&n,&m,&Mod);
    ask.clear();
    for (int i=0; i<m; i++){
        scanf("%d%d",&x,&y);
        ask.push_back(com(x,y));
    }
}

void extended_gcd(int a, int b, int &x, int &y){
```

```
    if (!b){
        x=1,y=0;
    } else{
        extended_gcd(b,a%b,y,x);
        y-=(a/b)*x;
    }
}

//calc 1/a mod b
int calcInv(int a, int b){
    int x,y;
    extended_gcd(a,b,x,y);
    return (x%b+b)%b;
}

vector<data> factorization(int Mod){
    vector<data> factor;
    int N=Mod;
    for (int i=2; i*i<=Mod; i++)
        if (N%i==0){
            int temp=1;
            while (N%i==0){
                temp*=i;
                N/=i;
            }
            factor.push_back(data(i,temp));
        }
    if (N!=1) factor.push_back(data(N,N));
    return factor;
}

void initialization(int Pi, int P){
    factorial[0]=1;
    for (int i=1; i<min(MAXN,P); i++){
        if (i%Pi!=0){
            factorial[i]=factorial[i-1]*i%P;
        } else factorial[i]=factorial[i-1];
    }
    if (P<MAXN){
        power[1]=factorial[P-1]; power[0]=power[1]*power[1]%P;
    }
}

//calc n! mod P
int calcFactorialMod(int n, int &cnt, int Pi, int P){
    cnt=0; long long ret=1;
    for (int i=n; i>=1; i/=Pi){
        ret=ret*power[(i/P)&1]*factorial[i%P]%P;
        cnt+=i/Pi;
    }
    return ret;
```

```
}

//calculate C(n,m)%P
int combination(int n, int m, int Pi, int P){
    if (m>n) return 0;
    int x,y,cnt1,cnt2,cnt3;
    long long ret1=calcFactorialMod(n,cnt1,Pi,P);
    long long ret2=calcFactorialMod(m,cnt2,Pi,P);
    long long ret3=calcFactorialMod(n-m,cnt3,Pi,P);

    cnt1-=cnt2+cnt3;
    ret1=ret1*calcInv(ret2*ret3%P,P)%P;
    for (int i=0; i<cnt1; i++) ret1=ret1*Pi%P;
    return ret1;
}

//merge
pair<int,int> Chinese_remainder(pair<int,int> a, pair<int,int> b){
    long long p1=a.second,p2=b.second,M=p1*p2;
    int t1=p2*calcInv(p2,p1)*a.first%M,t2=p1*calcInv(p1,p2)*b.first%M;
    return make_pair((t1+t2)%M,M);
}

void solve(){
    vector<data> factor=factorization(Mod);
    vector< pair<int,int> > final(ask.size());
    for (int i=0; i<final.size(); i++) final[i].first=final[i].second=-1;
    for (int i=0; i<factor.size(); i++){
        initialization(factor[i].Pi,factor[i].P);
        for (int j=0; j<ask.size(); j++){
            pair<int,int> p=make_pair(combination(ask[j].n,ask[j].m,
                        factor[i].Pi,factor[i].P),factor[i].P);
            if (final[j].first==-1){
                final[j]=p;
            } else{
                final[j]=Chinese_remainder(p,final[j]);
            }
        }
    }
    for (int i=0; i<final.size(); i++)
        printf("C(%d,%d) Mod %d = %d\n",ask[i].n,ask[i].m,Mod,final[i].first);
}

int main(){
    scanf("%d",&ca);
    for (int i=0; i<ca; i++){
        init();
        solve();
    }
    return 0;
}
```

4.5 欧拉函数

欧拉函数的性质：

$(1)\varphi(p^k) = p^k - p^{k-1}$ 其中 p 是质数
$(2)\varphi(nm) = \varphi(n)\varphi(m)$ 其中 $(n, m) = 1$

推论：

$(1)\sum_{1\le i\le n,(i,n)=1} i = \frac{1}{2}n\varphi(n)$
$(2)\sum_{d|n}\varphi(d) = n$
$(3)a^{\varphi(m)} \equiv 1(mod \quad m)$

线性求 $1\ N$ 的欧拉函数。

```
void calcphi(int n){
    phi[1] = 1; total = 0;
    memset(check,0,sizeof(check));
    for (int i = 2; i <= n; i++){
        if (!check[i]){
            phi[i] = i - 1;
            prime[total++] = i;
        }
        for (int j = 0; j < total; j++){
            if (i * prime[j] > n) break;
            check[i * prime[j]] = true;
            if (i % prime[j] == 0){
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            } else{
                phi[i * prime[j]] = phi[i] * (prime[j] - 1);
            }
        }
    }
}
```

4.6 莫比乌斯函数

当 $n = 1$ 时 $\mu(n) = 1$，当 $n = p_1p_2\cdots p_k$ 时，$\mu(n) = (-1)^k$，其余情况 $\mu(n) = 0.$

```
void calcmu(int n){
    memset(check,0,sizeof(check));
    mu[1] = 1; total = 0;
    for (int i = 2; i <= n; i++){
        if (!check[i]){
            prime[total++] = i;
            mu[i] = -1;
        }
        for (int j = 0; j < total; j++){
            if (i * prime[j] > n) break;
            check[i * prime[j]] = true;
            if (i % prime[j] == 0){
                mu[i * prime[j]] = 0;
```

```
                break;
            } else{
                mu[i * prime[j]] = -mu[i];
            }
        }
    }
    for (int i = 1; i <= n; i++) printf("%d %d\n",i,mu[i]);
}
```

4.7 Head算法

求$xy\ mod\ M$时，若$a*b$已经超出计算机整数表示范围，则会造成运算溢出。假设内置整数类型所能表示的最大数为w，则当$M<\frac{w}{2}$时，Head算法能在不造成运算溢出的情况下，计算$x*y\ mod\ M$。

令$T=\lfloor\sqrt{n}+\frac{1}{2}\rfloor$，$t=T^2-M$。其中有$|t|<T$且$t\equiv T^2\ mod\ M$

任意整数都能表示成$aT+b$的形式，其中$0\le a\le T, 0\le b<T$。

令$x=aT+b, y=cT+d$，则$xy\equiv(aT+b)(cT+d)\ (mod\ M)$。

$$(aT+b)(cT+d)\ (mod\ M)\equiv[hT+(g+f)t+bd]\ (mod\ M)$$

在以上等式中，将ac表示成$eT+f$的形式，将$ad+bc+et$表示成$gT+h$的形式。算出e,f,g,h，然后计算$(hT+(g+f)t+bd)\ mod\ M$就会在不造成运算溢出的情况下算出答案。

程序:

```
#include<cmath>

#define ll long long

ll mul_mod(ll x, ll y, ll n){
    ll T=floor(sqrt(n)+0.5);
    ll t=T*T-n;

    ll a=x/T, b=x%T;
    ll c=y/T, d=y%T;
    ll e=a*c/T, f=a*c%T;

    ll v=((a*d+b*c)%n+e*t)%n;
    ll g=v/T, h=v%T;

    ll ret=(((f+g)*t%n + b*d)%n+h*T)%n;
    ret=(ret%n+n)%n;
    return ret;
}
```

4.8 行列式取模

4.8.1 辗转相除法

```
/*
 * 1、若矩阵两行进行交换，行列式取反
```

```
 * 2、一行数加减另外一行数的倍数，行列式不变
 * 于是对于第i行和第j行，可以将第j行减去matrix[j][i]/matrix[i][i]倍的i行
 * 然后把两行交换，不断迭代至第j行的第i个元素变成0为止
 * 这个方法适用于任何模数，时间复杂度为O(n^3logC)，C就是矩阵的数的范围
*/
int gauss(int matrix[][MAXN], int n){
    int ret=1;
    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
            matrix[i][j]=(matrix[i][j]%Mod+Mod)%Mod;
    for (int i=0; i<n; i++){
        for (int j=i+1; j<n; j++){
            while (matrix[j][i]){
                int t=matrix[i][i]/matrix[j][i];
                for (int k=i; k<n; k++){
                    matrix[i][k]=(matrix[i][k]+(Mod-matrix[j][k])*t)%Mod;
                    swap(matrix[j][k],matrix[i][k]);
                }
                ret=Mod-ret;
            }
        }
        if (!matrix[i][i]) return 0;
        ret=ret*matrix[i][i]%Mod;
    }
    return ret;
}
```

4.8.2 高斯消元

```
/*
 * 把矩阵消成上三角矩阵，行列式就是对角线相乘
 * 适用于模数为质数的情况
 * 使用时注意范围溢出问题
 * 时间复杂度O(n^3)
*/

void extended_gcd(int a, int b, int &x, int &y){
    if (!b){
        x=1,y=0;
        return;
    }
    extended_gcd(b,a%b,y,x);
    y-=(a/b)*x;
}

int gauss(int matrix[][MAXN], int n){
    long long ret=1,inv=1;
    for (int i=0; i<n; i++)
```

```
        for (int j=0; j<n; j++)
            matrix[i][j]=(matrix[i][j]%Mod+Mod)%Mod;

    for (int i=0; i<n; i++){
        int absMax=matrix[i][i],pos=i;
        for (int j=i+1; j<n; j++){
            if (abs(matrix[j][i])>abs(absMax)){
                pos=j;
                absMax=matrix[j][i];
            }
        }
        if (i!=pos){
            for (int j=0; j<n; j++){
                swap(matrix[i][j],matrix[pos][j]);
            }
            ret=Mod-ret;
        }
        if (!matrix[i][i]) return 0;

        for (int k=i+1; k<n; k++){
            if (matrix[k][i]==0) continue;
            int d=gcd(abs(matrix[i][i]),abs(matrix[k][i]));
            int lcm=abs(matrix[i][i]*matrix[k][i])/d,
mul1=lcm/matrix[i][i],mul2=lcm/matrix[k][i];
            inv=inv*mul2%Mod;
            if (inv<0) inv+=Mod;
            for (int j=i; j<n; j++){
                matrix[k][j]=matrix[k][j]*mul2-matrix[i][j]*mul1;
                matrix[k][j]%=Mod;
                if (matrix[k][j]<0) matrix[k][j]+=Mod;
            }
        }
        ret=ret*matrix[i][i]%Mod;
    }

    int x,y;
    extended_gcd(inv,Mod,x,y);
    x=(x%Mod+Mod)%Mod;
    ret=ret*x%Mod;
    return ret;
}
```

4.9 特殊数列

4.9.1 斐波那契数

斐波那契数列:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$$

斐波那契数列的性质:

$(1): f(1)+f(2)+\cdots+f(n)=f(n+2)-1$
$(2): f(1)+f(3)+f(5)+\cdots+f(2n-1)=f(2n)$
$(3): f(2)+f(4)+f(6)+\cdots+f(2n)=f(2n+1)-1$
$(4): f(0)^2+f(1)^2+\cdots+f(n)^2=f(n)f(n+1)$
$(5): f(n+k)=f(m)f(n+1)+f(m-1)f(n)$
$(6): gcd(f(n),f(m))=f_{gcd(n,m)}$

4.9.2 卡特兰数

卡特兰序列:

$$1,2,5,14,42,132,429,1430,4862$$

第n项卡特兰数:

$$C_n=\frac{C_{2n}^n}{n+1}\quad ,n\geq 0$$

模型: 有n个$+1$和n个-1构成的$2n$项

$$a_1,a_2,\cdots,a_{2n}$$

其部分和满足

$$a_1+a_2+\cdots+a_k\geq 0\quad (k=1,2,\cdots,2n)$$

的数列的个数等于第n个Catalan数

$$C_n=\frac{C_{2n}^n}{n+1}$$

应用:
(1) 矩阵链乘，$P=a_1\times a_2\times\cdots\times a_n$ 根据乘法的结合律，不改变其顺序，只用括号表示成对的乘积，则一共有C_n种括号方案。
(2) 一个容量无强大的栈的进栈次序为$1,2,3,\cdots,n$则一共有C_n种出栈方式
(3) 具有$n+1$条边的凸多边形的三角剖分方案有C_{n-1}种
(4) 在圆上选择$2n$个点，将这些点成对连接起来使得所有得到的n条线段不相交的方案数有C_n种
(5) 给定n个节点，能构成C_n种不同的二叉树

4.9.3 Stirling数

第一类sitrling数：将n个数划分成k个循环，一共有$f(n,k)$总方案,

$$f(n,k)=f(n-1,k-1)+(n-1)f(n-1,k)$$

第二类stirling数：将n个数划分成k个集合，一共有$g(n,k)$种方案,

$$g(n,k)=kg(n-1,k)+g(n-1,k-1)$$

$$1^k+2^k+\cdots+n^k=g(k,0)C_{n+1}^1+g(k,1)C_{n+1}^2+\cdots+g(k,k)C_{n+1}^{k+1}$$

4.10 格子路径与 Schroder 数

1、从 $(0,0)$ 走到 (n,m)，只允许向上和向右走的矩形格子路径数为 $C(n+m,m)$。
2、从 $(0,0)$ 走到 (p,q)，只允许向上和向右走，且不得穿越对角线的方案数为 $\frac{p-q+1}{p+1} * C(p+q,q)$。
3、从 $(0,0)$ 走到 (p,q)，允许向上向右走，且允许 $D=(1,1)$ 的步进，不得穿越对角线的方案数为

$$K(p,q)=\sum_{r=0}^{min\{p,q\}}\frac{(p+q-r)!}{(p-r)!(q-r)!r!}$$

。

5 自定义类型及 C++ 算法

5.1 C++note

g++增栈，放在main函数第一句话

```
int SIZE = (1<<20) * 10; //10M
char *p=(char*)malloc(SIZE)+SIZE;
__asm__("movl %0, %%esp\n" :: "r"(p));
```

vc++增栈

```
#pragma comment(linker, "/STACK:16777216")
```

5.2 Vim 配置

```
#ts -> tabstop , sw -> shiftwidth si -> smartindent
set ts=4 sw=4 si nu
syntax enable
set makeprg=g++\ -o\ %<\ %\ -g
map <silent> mk :make <CR>
map <silent> mr :!./%< <CR>
map <silent> mw :!./%< <%<.in<CR>
map <silent> mn :cnext<CR>
map <silent> mp :cprev<CR>
map <F2> <ESC>ggyyVG"+y
map <C-H> gT
map <C-L> gt
```