生成树1

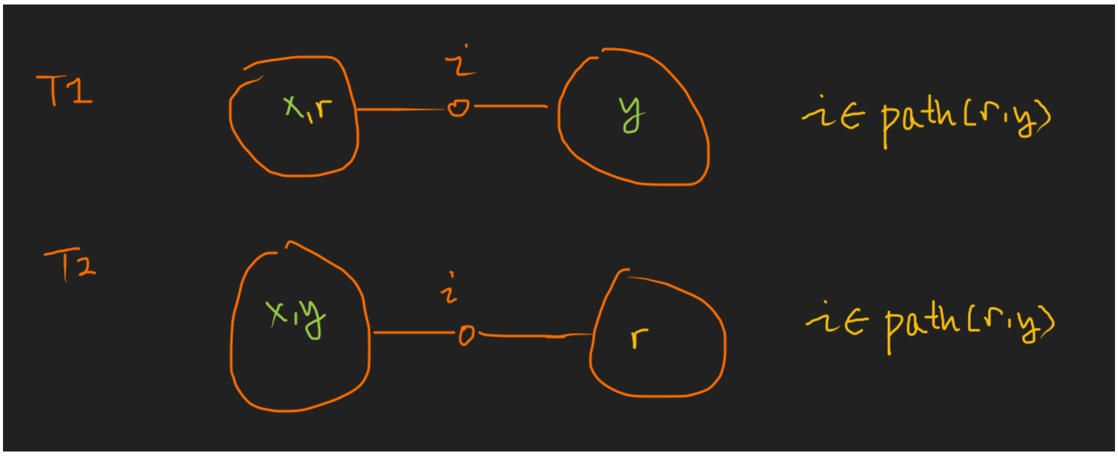
性质1用邻接矩阵可以直接判掉（注意这是无向图）。

对于性质2，不妨定住 u，以 u 为根跑 n 次DFS，然后枚举 v，把 k 条路径并起来（从每棵树上 v 一起向上跳），由于有点不重复的限制，若发现 k 条路径有重复点则非法，否则它们的并最多包含 n 个点，这样就在 $O(n)$ 时间内检查了一对 (u,v)。总复杂度 $O(n^3)$。

进一步地，性质2被满足当且仅当下列条件成立：

- 对于每个点 i，其至多在一棵生成树中度数大于 1（不是叶子）

> 假设点 i 在树 $T1,T2$ 中度数大于 1：在树 $T1$ 中有 $x-i-y$，在树 $T2$ 中 x,y 不能位于 i 异侧，不妨假设有 $x,y-l-i-r$；若 r 点在 $T1$ 中位于 x 一侧（即 $x,r-i-y$，另一侧同理），那么路径 (r,y) 在两棵树上存在公共点 i。



输入的时候统计每个点的度数，复杂度 $O(n^2)$。

```
#include<bits/stdc++.h>
using namespace std;
#define N 5005
int n,k,e[N][N],vis[N],in[N];
int main(){
    cin>>n>>k;
    for(int i=1;i<=k;++i){
        memset(in,0,sizeof in);
        for(int j=1,x,y;j<n;++j){
            cin>>x>>y;
            in[x]++,in[y]++;
            e[x][y]++,e[y][x]++;
            if(e[x][y]>1) cout<<"ERR1",exit(0);
        }
        for(int i=1;i<=n;++i){
            vis[i]+=(in[i]>1);
            if(vis[i]>1) cout<<"ERR2",exit(0);
        }
    }
    cout<<"OK";
```

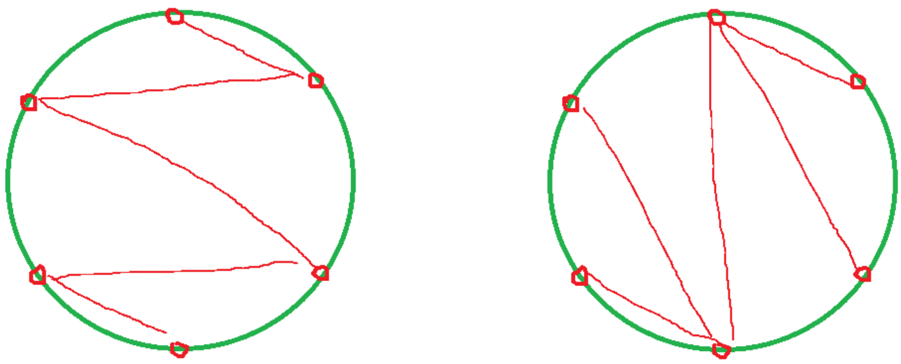
```
}
```

生成树2

将 n 个点放在圆上，先找到一棵满足条件的树，然后转 k 次。

Alice的问题：从 1 出发左右横跳，即 $1 \to 2 \to n \to 3 \to n-1 \to 4 \to ...$

Bob的问题：连 $1 \to (2,3,\ldots n/2-1,n/2), n/2 \to (n/2+1,n/2+2,\ldots,n-1,n)$



互质询问

原题：COCI

预处理值域 n 范围内每个数的质因子，对每个质因子 p 开 `set` 记录数集中所有 p 的倍数。对于一次查询 $[l,r]$，可以检查所有 $p \le r$ 的 `set`，看是否有一个 `set` 包含至少2个 $[l,r]$ 中的数。

为了加速查询，加入/删除 x 同时维护 `set` 中每个数的后继 $nxt[i]$ 位置，查询变为检查 $[l,r]$ 中 $min(nxt[i]) \le r, i \in [l,r]$，可以使用维护区间min的线段树（每个`set` 视为一条链信息，将所有链压缩在一棵线段树上），在叶子上开 `set` 支持单点修改。

1操作2个log，2操作1个log。总复杂度为大常数的2个log。

```
void change(node *p, int id, int x, int y);//单点修改id处的set，删除x、加入y
int query(node *p, int l, int r);//区间[l,r]查询最小值
vector<int> adjp[MAXN];//adjp[x]存储所有x的质因子
set<int> pos[MAXN];
bool vis[MAXN];
void pre(){
    for(int i=2;i<=n;i++){
```

```
        if(vis[i]) continue;
        adjp[i].push_back(i);
        for(int j=2*i;j<=n;j+=i){
            vis[j] = 1;
            adjp[j].push_back(i);
        }
    }
    rt = buildT(1, n);
}

int l, r;
void find_lr(int p, int x){//在p的倍数中查找x的前驱/后继
    l = -1, r = -1;
    it = pos[p].find(x);
    ++it;
    if(it != pos[p].end()) r = *it;
    --it;
    if(it != pos[p].begin()){
        --it;
        l = *it;
    }
}

void work(int x){
    if(!in[x]){//add
        in[x] = 1;
        for(auto p:adjp[x]){
            pos[p].insert(x);
            find_lr(p, x);
            if(l != -1) change(rt, l, r, x);
            if(r != -1) change(rt, x, -1, r);
        }
    } else {//del
        in[x] = 0;
        for(auto p:adjp[x]){
            find_lr(p, x);
            if(l != -1) change(rt, l, x, r);
            if(r != -1) change(rt, x, r, -1);
            pos[p].erase(x);
        }
    }
}

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin>>n>>q;
    pre();
    char op;
    int l,r,x;
    while(q--){
        cin>>op;
```

```
        if(op=='S'){
            cin>>x;
            work(x);
        }
        else{
            cin>>l>>r;
            if(query(rt, l, r) <= r) cout<<"yes"<<'\n';
            else cout<<"no"<<'\n';
        }
    }
    return 0;
}
```

数字串

原题：P2282 [HNOI2003]历史年份

记 $num(i,j) = s[i \sim j]$ 构成的数字值

$f[i] = s[1 \sim i]$ 的（最优）分割方案中，最后一段的起始位置最大值

$g[i] = s[i \sim n]$ 的（最优）分割方案中，保证最后一段最小的前提下，第一段的结尾位置最大值

算法1

首先正着DP求出 f 数组，转移为：

- $f[i] = max(j, 1 \le j < i \wedge num(f[j-1], j-1) < num(j,i))$

然后从 $g[f[n]] = n$ 出发，反着DP求出 g 数组，转移为：

- $g[i] = max(j, i \le j \le n \wedge num(i,j) < num(j+1, g[j+1]))$

最终答案从 $g[1]$ 出发输出答案。

暴力进行 num 比较，直接实现是 $O(n^3)$ 的，期望得分30分。

算法2

字符串比较可以二分 + hash，就是先比较字符串长度，长度相同的话就二分找最长公共前缀 lcp ，然后直接比较 $lcp+1$ 这个位置。

但是前缀 `0` 需要先处理掉，这里可以预处理 $L/R[i]$ 代表 $s[i]$ 前后最近的不是 `0` 位置，就可以找到最长的一段前缀 `0` 或者后缀 `0`。

这样复杂度是 $O(n^2 logn)$，期望得分55~70分。

算法3

优化DP转移，注意这两个DP都满足决策单调性（$f[i], g[i]$ 不降），可以直接用决策单调性的套路优化。

考虑我为人人，从 $f[i]$ 出发，发现满足 $num(f[i], i) < num(i+1, j)$ 的 j 形成一个后缀，可以使用一个支持区间和 x 取max、单点求值的线段树。

$g[i]$ 的转移类似。

这样复杂度是1个log，可以通过。

注意：

1. 线段树可以使用标记永久化减小常数
2. 线段树和数组清空不要 `memset`
3. 反着DP的时候，初始 $f[n]$ 以及之前的一段前导 `0` 位置直接赋初值 $g[i] = n$

```cpp
#include<bits/stdc++.h>
#define MAXN 2000005
using namespace std;
typedef unsigned long long ull;
struct node{
    int l,r,tag;
    node *ls, *rs;
} pool[2*MAXN], *rt;
int top = 0;
node* build(int l, int r){
    node *p = pool + (++top);
    p->l = l; p->r = r;
    p->ls = p->rs = 0;
    p->tag = 0;

    if(l==r) return p;
    int mid = (l+r)/2;
    p->ls = build(l, mid);
    p->rs = build(mid+1, r);
    return p;
}

void change(node *p, int ql, int qr, int val){
    if(qr <= 0 || ql > qr) return;
    if(p->l==ql && p->r==qr){
        p->tag = max(p->tag, val);
        return;
    }
    int mid = (p->l+p->r)/2;
    if(qr <= mid) change(p->ls, ql, qr, val);
    else if(ql >= mid+1) change(p->rs, ql, qr, val);
    else{
        change(p->ls, ql, mid, val);
        change(p->rs, mid+1, qr, val);
    }
}

```

```
int query(node *p, int x, int tag){
    tag = max(tag, p->tag);
    if(p->l==x && p->r==x) return tag;
    int mid = (p->l + p->r)/2;
    if(x <= mid) return query(p->ls, x, tag);
    else return query(p->rs, x, tag);
}

int n;
char s[MAXN];
ull h[MAXN], pw[MAXN];
ull base = 233;

ull calh(int l, int r){
    return h[r] - h[l-1]*pw[r-l+1];
}

int lcp(int i, int j, int len){//s[i,i+len-1]和s[j,j+len-1]的lcp
    if(s[i] != s[j]) return 0;
    int l = 1, r = len, mid;
    while(l<r){
        mid = (l+r+1)/2;
        if(calh(i,i+mid-1) == calh(j,j+mid-1)) l = mid;
        else r = mid-1;
    }
    return l;
}

int L[MAXN], R[MAXN], f[MAXN], g[MAXN];
void init(){
    R[n+1] = n+1;
    for(int i=n;i>=1;--i) R[i] = (s[i]!='0')?i:R[i+1];
    L[0] = 0;
    for(int i=1;i<=n;i++) L[i] = (s[i]!='0')?i:L[i-1];

    pw[0] = 1;
    for(int i=1;i<=n;i++){
        pw[i] = pw[i-1] * base;
        h[i] = h[i-1] * base + s[i];
    }
}

bool cmp(int l1, int r1, int l2, int r2){//s[l1,r1] < s[l2,r2]
    if(r1==0) return 1;
    int len1 = r1-R[l1]+1, len2 = r2-R[l2]+1;
    if(len2 <= 0) return 0;
    if(len1 != len2) return len1 < len2;

    int k = lcp(R[l1], R[l2], len1);
    if(k == len1) return 0;
    return s[R[l1]+k] < s[R[l2]+k];
}
```

```
void solve1(){
    for(int i=0;i<=n+1;i++) f[i] = 0;
    top = 0;
    rt = build(1, n);

    f[0] = 1;
    int len,j;
    for(int i=1;i<=n;i++){
        f[i] = query(rt, i, f[i-1]);
        len = R[i+1] - (i+1);
        j = i+1+len+(i-R[f[i]]);
        if(j > n) continue;
        if(!cmp(R[f[i]], i, i+1+len, j)) ++j;
        change(rt, j, n, i+1);
    }
}


void solve2(){
    for(int i=0;i<=n+1;i++) g[i] = 0;
    top = 0;
    rt = build(1, n);
    for(int i=L[f[n]-1]+1;i<=n;i++) g[i] = n;

    int len,j;
    for(int i=f[n];i>=1;i--){
        g[i] = query(rt, i, g[i]);
        len = g[i]-R[i]+1;
        j = i-1-len+1;

        if(!cmp(j, i-1, i, g[i])) j = R[j]+1;
        j = L[j-1]+1;
        j = max(j, 1);
        change(rt, j, i-1, i-1);
    }

    for(int i=1;i<=n;){
        for(int j=i;j<=g[i];j++) cout<<s[j];
        i = g[i]+1;
        if(i<=n) cout<<",";
    }
    cout<<'\n';
}

int main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    int tq = 0;
    cin>>tq;
    while(tq--){
        cin>>s+1;
```

```
        n = strlen(s+1);
        init();
        solve1();
        solve2();
    }
    return 0;
}
```