Interval Tree

```
#define MX (int)1e6
#define lson l,m,n<<1
#define rson m+1,r,n<<1|1
#define lc n<<1
#define rc n<<1|1
int num[MX], sum[MX << 2], ma[MX << 2], mi[MX << 2], add[MX << 2];
int L, R, N, x, v;
inline void up(int n)
{
    sum[n] = sum[lc] + sum[rc];
    ma[n] = max(ma[lc], ma[rc]);
    mi[n] = min(mi[lc], mi[rc]);
}
void B(int l = 1, int r = N, int n = 1)
{
    // add[n] = 0;
    if (l == r)
    {
        scanf("%d", &num[l]);
        sum[n] = ma[n] = num[l];
        return;
    }
    int m = l + r >> 1;
    B(lson), B(rson), up(n);
}
// Prepare: x, v
void U(int l = 1, int r = N, int n = 1)
{
    if (l == r)
```

```
    {
        // sum[n] = mi[n] = v;
        return;
    }
    int m = l + r >> 1;
    if (x <= m) U(lson);
    else U(rson);
    up(n);
}
// Prepare: L, R
int Q(int l = 1, int r = N, int n = 1)
{
    if (L <= l && r <= R)
    {
        // return sum[n];
        // return ma[n];
    }
    down(n, r - l + 1);
    int ans = 0, m = l + r >> 1;
    if (L <= m) ans += Q(lson);
    // ans = max(ans, Q(lson));
    if (m < R) ans += Q(rson);
    // ans = max(ans, Q(rson));
    return ans;
}
void down(int n, int m)
{
    if (add[n])
    {
        add[lc] += add[n];
        add[rc] += add[n];
        sum[lc] += add[n] * (m - (m >> 1));
        sum[rc] += add[n] * (m >> 1);
        add[n] = 0;
    }
}
// Prepare: L, R, v
void U(int l = 1, int r = N, int n = 1)
{
    if (L <= l && r <= R)
    {
        add[n] += v, sum[n] += v * (r - l + 1);
        return;
    }
    down(n, r - l + 1);
    int m = (l + r) >> 1;
    if (L <= m) U(lson);
    if (m < R) U(rson);
    up(n);
}
```

Interval Tree 2D

```
#define lson1 l,m,n<<1
#define rson1 m+1,r,n<<1|1
#define lc1 n<<1
```

```
#define rc1 n<<1|1
#define lson2 a,m,o<<1
#define rson2 m+1,b,o<<1|1
#define lc2 o<<1
#define rc2 o<<1|1
using namespace std;
int mi[MX << 2][MX << 2], N;
void B2(int l, int r, int n, int a = 1, int b = N, int o = 1)
{
    if (a == b)
    {
        if (l == r)
            scanf("%d", &mi[n][o]);
        else mi[n][o] = min(mi[lc1][o], mi[rc1][o]);
    }
    else
    {
        int m = a + b >> 1;
        B2(l, r, n, lson2), B2(l, r, n, rson2);
        mi[n][o] = min(mi[n][lc2], mi[n][rc2]);
    }
}
void B1(int l = 1, int r = N, int n = 1)
{
    if (l == r)
    {
        B2(l, r, n);
        return;
    }
    int m = l + r >> 1;
    B1(lson1), B1(rson1);
    B2(l, r, n);
}
int Q2(int l, int r, int n, int a, int b, int o)
{
    if (l == a && r == b)
        return mi[n][o];
    int m = l + r >> 1;
    if (b <= m)
        return Q2(l, m, n, a, b, lc2);
    if (m < a)
        return Q2(m + 1, r, n, a, b, rc2);
    return min(Q2(l, m, n, lson2), Q2(m + 1, r, n, rson2));
}
int Q1(int x1, int y1, int x2, int y2, int l = 1, int r = N, int n = 1)
{
    if (l == x1 && r == x2)
        return Q2(1, N, n, y1, y2, 1);
    int m = l + r >> 1;
    if (x2 <= m)
        return Q1(x1, y1, x2, y2, lson1);
    if (m < x1)
        return Q1(x1, y1, x2, y2, rson1);
    return min(Q1(x1, y1, m, y2, lson1), Q1(m + 1, y1, x2, y2, rson1));
}
int main()
```

```
{
    int t, q, x1, x2, y1, y2;
    scanf("%d", &t);
    while (t--)
    {
        scanf("%d", &N), B1();
        scanf("%d", &q);
        while (q--)
        {
            scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
            printf("%d\n", Q1(x1, y1, x2, y2));
        }
    }
    return 0;
}
```

KMP

```
//这个模板 字符串是从0开始的 Next数组是从1开始的
int next[N];
char S[N], T[N];
int slen, tlen;
void getNext()
{
    int j, k;
    j = 0; k = -1; next[0] = -1;
    while (j < tlen)
        if (k == -1 || T[j] == T[k])
            next[++j] = ++k;
        else
            k = next[k];
}
//返回模式串T在主串S中首次出现的位置
//返回的位置是从0开始的。
int KMP_Index()
{
    int i = 0, j = 0;
    getNext();
    while (i < slen && j < tlen)
    {
        if (j == -1 || S[i] == T[j])
        {
            i++; j++;
        }
        else
            j = next[j];
    }
    if (j == tlen)
        return i - tlen;
    else
        return -1;
}
//返回模式串在主串S中出现的次数
int KMP_Count()
{
    int ans = 0;
```

```
    int i, j = 0;
    if (slen == 1 && tlen == 1)
    {
        if (S[0] == T[0])
            return 1;
        else
            return 0;
    }
    getNext();
    for (i = 0; i < slen; i++)
    {
        while (j > 0 && S[i] != T[j])
            j = next[j];
        if (S[i] == T[j])
            j++;
        if (j == tlen)
        {
            ans++;
            j = next[j];
        }
    }
    return ans;
}
int main()
{

    int TT;
    int i, cc;
    cin >> TT;
    while (TT--)
    {
        cin >> S >> T;
        slen = strlen(S);
        tlen = strlen(T);
        cout << "模式串T在主串S中首次出现的位置是: " << KMP_Index() << endl;
        cout << "模式串T在主串S中出现的次数为: " << KMP_Count() << endl;
    }
    return 0;
}
/*
test case
4
aaaaaa a
abcd d
aabaa b
*/
```

Manacher

```
char s[MX];
int l[MX];
void palindrome(char cs[], int len[], int n)   //len[i] means the max palindrome
length centered i/2
{
    for (int i = 0; i < n * 2; ++i)
    {
```

```
        len[i] = 0;
    }
    for (int i = 0, j = 0, k; i < n * 2; i += k, j = max(j - k, 0))
    {
        while (i - j >= 0 && i + j + 1 < n * 2 && cs[(i - j) / 2] == cs[(i + j +
1) / 2])
            j++;
        len[i] = j;
        for (k = 1; i - k >= 0 && j - k >= 0 && len[i - k] != j - k; k++)
        {
            len[i + k] = min(len[i - k], j - k);
        }
    }
}
int main()
{
    while (~scanf("%s", s))
    {
        int ans = 0, sl = strlen(s);
        palindrome(s, l, sl);
        for (int i = 0; i < sl * 2; ++i)
            ans = max(ans, l[i]);
        printf("%d\n", ans);
    }
    return 0;
}
```

Prime

```
void get_prime()
{
    int cnt = 0;
    for (int i = 2; i < N; i++)
    {
        if (!tag[i])    p[cnt++] = i;
        for (int j = 0; j < cnt && p[j] * i < N; j++)
        {
            tag[i*p[j]] = 1;
            if (i % p[j] == 0)
                break;
        }
    }
}
int phi[3000010]; //Euler Function
int euler()
{
    int i,j;
    for(i=0;i<3000010;i++)
    phi[i]=0;
    for(i=2;i<3000010;i++)
    {
        if(!phi[i])
        for(j=i;j<3000010;j+=i)
        {
           if(!phi[j])phi[j]=j;
           phi[j]=phi[j]/i*(i-1);
```

```
        }
    }
}
```

Hash

```
const ull B = 100000007ULL; /// 哈希基数，1e8 + 7
const int mx_s_num = 105; /// 字符串个数

char s[mx_s_num][mx]; /// 注意，一定要用gets(s[i] + 1)，从下标1开始读
ull ha[mx_s_num][mx], bp[mx] = {1ULL}; /// ha[i]从1开始，一直到ha[i][n]
int len[mx_s_num]; /// len[i] = strlen(s[i] + 1); 一定要是s[i] + 1，否则n会是0

void init_hash(int s_num) /// 请在main()中完成len的求取。
{
    int i, j;
    For(i, s_num) Forr(j, 1, len[i] + 1) ha[i][j] = ha[i][j - 1] * B + s[i][j];
    int n = Max(len, s_num); /// 调用#define的Max()
    Forr(i, 1, n + 1) bp[i] = bp[i - 1] * B;
}
ull get_hash(char *s) /// 直接返回整个字符串的hash
{
    ull ha = 0ULL;
    for (int i = 0; s[i]; ++i) ha = ha * B + s[i];
    return ha;
}
ull get_hash(int *a, int n) /// 返回整个int数组的hash值
{
    int i;
    ull ha = 0ULL;
    For(i, n) ha = ha * B + (ull)a[i];
    return ha;
}
/// 注意pos一定不能是0!!!!
inline ull get_hash(ull *Ha, int pos, int l) /// 返回Ha[pos...pos+l-1]的值，pos与l
必须是正数
{
    return Ha[pos + l - 1] - Ha[pos - 1] * bp[l];
}
inline ull merge_hash(ull ha1, ull ha2, int len2) /// 返回s1+s2拼接后的hash值
{
    return ha1 * bp[len2] + ha2;
}
bool contain(int ida, int idb) /// b是否为a的子串 ，ida和idb为字符串下标，若只有两个字
符串，使用时传入参数(0, 1)、(1, 0)就行
{
    if (len[ida] < len[idb]) return false;
    ull hab = ha[idb][len[idb]];
    for (int i = 1; i + len[idb] <= len[ida]; ++i)
        if (get_hash(ha[ida], i, len[idb]) == hab) return true;
    return false;
}
int overlap(int ida, int idb) /// 求a后缀与b前缀的最长公共子串，ida和idb为字符串下标，
若只有两个字符串，使用时传入参数(0, 1)、(1, 0)就行
{
    int ans = 0, i;
```

```
    Forr(i, 1, min(len[ida], len[idb]) + 1)
    if (get_hash(ha[ida], len[ida] - i + 1, i) == get_hash(ha[idb], 1, i)) ans =
i;
    // 可在if中加上 && strncmp(s[ida] + len[ida] - i + 1, s[idb] + 1, i) == 0(不过
这就失去意义了，还不如双hash)
    return ans;
}

//CF 443B
//Primary usage
#include <cstdio>
#include <cstring>
#include <algorithm>
#define MAXL 222
typedef unsigned long long ull;
using namespace std;
const ull B = 100000007ULL;
char s[MAXL];
int len;
ull ha[MAXL], bp[MAXL] = {1ULL};
void hash()
{
    ha[0] = s[0];
    for (int i = 1; i < len; ++i)
        ha[i] = ha[i - 1] * B + s[i], bp[i] = bp[i - 1] * B;
}
ull get(int pos, int l)
{
    return ha[pos + l - 1] - ha[pos - 1] * bp[l];
}
bool check(int p, int l)
{
    return get(p, min(l, len - p - l)) == get(p + l, min(l, len - p - l));
}
int main()
{
    int k;
    scanf("%s%d", s, &k);
    len = strlen(s);
    if (k >= len)
    {
        printf("%d\n", 2 * ((k + len) >> 1));
        return 0;
    }
    hash();
    for (int i = (len + k) >> 1; i >= k; --i)
    {
        for (int j = 0; j <= len + k - 2 * i; ++j)
        {
            if (check(j, i))
            {
                printf("%d\n", 2 * i);
                return 0;
            }
        }
    }
```

```
}
```

RMQ

//给出一个非降序列，求某区间内数字最大的出现次数。
//st算法比树状数组更快一点，但是树状数组需要的空间小。

RMQ	初始化	查询	空间
st算法	O(nlogn)	O(1)	O(nlogn)
树状数组	O(nlogn)	O(logn)	O(n)

```
//st算法
int p[MX], l[MX], r[MX], val[MX], cnt[MX], num[MX], d[MX][20];
void init(int n)
{
    for (int i = 0; i < n; ++i)
        d[i][0] = cnt[i];
    for (int j = 1; (1 << j) <= n; ++j)
        for (int i = 0; i + (1 << j) - 1 < n; ++i)
            d[i][j] = max(d[i][j - 1], d[i + (1 << (j - 1))][j - 1]);
}
int rmq(int x, int y)
{
    if (x > y)
        return 0;
    int k = 0;
    while ((1 << (k + 1)) <= y - x + 1)
        ++k;
    return max(d[x][k], d[y - (1 << k) + 1][k]);
}
int main()
{
    int n, q;
    while (~scanf("%d", &n) && n)
    {
        scanf("%d", &q);
        for (int i = 1; i <= n; ++i)
            scanf("%d", &p[i]);
        p[n + 1] = -MX;
        int k = 0;
        cnt[k] = 1, val[k] = p[1], l[1] = 1;
        for (int i = 2; i <= n + 1; ++i)
        {
            if (p[i] == p[i - 1])
                ++cnt[k], l[i] = l[i - 1], num[i] = k;
            else
            {
                for (int j = l[i - 1]; j < i; ++j)
                    r[j] = i - 1;
                ++k, num[i] = k, l[i] = i, cnt[k] = 1, val[k] = p[i];
            }
        }
        // for (int i = 1; i <= n; ++i)
        //     printf("#%d %d %d\n", l[i], r[i], num[i]);
```

```
        // for (int i = 0; i < k; ++i)
        //     printf("##%d %d\n", cnt[i], val[i]);
        int x, y, ans;
        init(k);
        while (q--)
        {
            scanf("%d%d", &x, &y);
            if (num[x] == num[y])
                ans = y - x + 1;
            else ans = max(max(r[x] - x + 1, y - l[y] + 1), rmq(num[x] + 1,
num[y] - 1));
            printf("%d\n", ans);
        }
    }
    return 0;
}

//树状数组
#include <bits/stdc++.h>
#define MX 100010
using namespace std;
int p[MX], l[MX], r[MX], val[MX], cnt[MX], num[MX], idx[MX];
void init(int n)
{
    for (int i = 1; i < n; ++i)
    {
        idx[i] = cnt[i];
        for (int j = 1; j < (i & -i); j <<= 1)
            idx[i] = max(idx[i], idx[i - j]);
    }
}
int query(int x, int y)
{
    if (x > y)return 0;
    int ans = cnt[y];
    while (1)
    {
        ans = max(ans, cnt[y]);
        if (x == y) break;
        for (y -= 1; y - x >= (y & -y); y -= (y & -y))
            ans = max(ans, idx[y]);
    }
    return ans;
}
int main()
{
    int n, q;
    while (~scanf("%d", &n) && n)
    {
        scanf("%d", &q);
        for (int i = 1; i <= n; ++i)
            scanf("%d", &p[i]);
        p[n + 1] = -MX;
        int k = 1;
        cnt[k] = 1, val[k] = p[1], l[1] = 1, num[1] = k;
        for (int i = 2; i <= n + 1; ++i)
```

```
        {
            if (p[i] == p[i - 1])
                ++cnt[k], l[i] = l[i - 1], num[i] = k;
            else
            {
                for (int j = l[i - 1]; j < i; ++j)
                    r[j] = i - 1;
                ++k, num[i] = k, l[i] = i, cnt[k] = 1, val[k] = p[i];
            }
        }
        int x, y, ans;
        init(k);
        while (q--)
        {
            scanf("%d%d", &x, &y);
            if (num[x] == num[y])
                ans = y - x + 1;
            else ans = max(max(r[x] - x + 1, y - l[y] + 1), query(num[x] + 1,
num[y] - 1));
            printf("%d\n", ans);
        }
    }
    return 0;
}
```

LIS & LCS

```
#include <bits/stdc++.h>
using namespace std;
int lis[MAXN];
int lis()
{
    int n, i, j, x, len = 0;
    for(i = 1; i <= n; ++i)
    {
        scanf("%d", &x); // for existing array, use x=num[i]
        j = lower_bound(lis + 1, lis + len + 1, x) - lis;
        // LDS : j = lower_bound(lds + 1, lds + len + 1, x, greater<int>()) -
lds;
        lis[j] = x;
        len = max(len, j);
    }
    return len;
}
//LIS[i] = max{1,LIS[k]+1}  (∀k<i, arr[i] > arr[k])

/*
Longest Not-decrease Sequence:
bool cmp(int a, int b)
{
    return a <= b;
}
j = lower_bound(lis + 1, lis + len + 1, x, cmp) - lis;
*/
more
```

```
#include<bits/stdc++.h>
using namespace std;
const int MAXSTRLEN = 1000;

char a[MAXSTRLEN], b[MAXSTRLEN];
int dp[MAXSTRLEN][MAXSTRLEN], path[MAXSTRLEN][MAXSTRLEN];

int Lcs(char x[], char y[])
{
    int i, j, len1 = strlen(x + 1), len2 = strlen(y + 1);
    memset(dp, 0, sizeof(dp));
    for (i = 1; i <= len1; ++i)
        for (j = 1; j <= len2; ++j)
        {
            if (x[i] == y[j])
                dp[i][j] = dp[i - 1][j - 1] + 1, path[i][j] = 1;
            else if (dp[i - 1][j] >= dp[i][j - 1])
                dp[i][j] = dp[i - 1][j], path[i][j] = 2;
            else
                dp[i][j] = dp[i][j - 1], path[i][j] = 3;
        }
    return dp[len1][len2];
}

void PrintLcs(int i, int j)
{
    if (i == 0 || j == 0) return;
    if (path[i][j] == 1)
    {
        PrintLcs(i - 1, j - 1);
        putchar(a[i]);
    }
    else if (path[i][j] == 2) PrintLcs(i - 1, j);
    else PrintLcs(i, j - 1);
}

int main()
{
    while (gets(a + 1))
    {
        gets(b + 1);
        printf("%d\n", Lcs(a, b));
        PrintLcs(strlen(a + 1), strlen(b + 1));
        putchar(10);
    }
    return 0;
}
```

Binary

```
int l = MIN, r = MAX, p = (l + r) >> 1;
while (l <= r)
{
    if (arr[p] == k)
    {
        cout << p << endl;
```

```
        return;
    }
    if (arr[p] < k)
        l = p + 1;
    else if (arr[p] > k)
        r = p - 1;
    p = (l + r) >> 1;
}
cout << "No solution" << endl;
```

Trie

```
#define MAXW 100010
#define MAXL 12
const int MAXN = MAXW * MAXL;
struct node
{
    int next[26];
    int cnt;
} t[MAXN];
int ts = 0;
int insert(char s[])
{
    int len = strlen(s), p = 0;
    for (int i = 0; i < len; ++i)
    {
        if (!t[p].next[s[i] - 'a'])
            t[p].next[s[i] - 'a'] = ++ts;
        ++t[p].cnt;
        p = t[p].next[s[i] - 'a'];
    }
    return ++t[p].cnt;
}
int query(char s[])
{
    int len = strlen(s), p = 0;
    for (int i = 0; i < len; ++i)
    {
        if (!t[p].next[s[i] - 'a'])
            return 0;
        p = t[p].next[s[i] - 'a'];
    }
    return t[p].cnt;
}
int main()
{
    char s[MAXL];
    int n;
    scanf("%d", &n);
    while (n--)
    {
        scanf("%s", s);
        insert(s);
    }
    scanf("%d", &n);
    while (n--)
```

```
    {
        scanf("%s", s);
        printf("%d\n", query(s));
    }
    return 0;
}

//lrj (with attachment)
const int maxnode = 4000 * 100 + 5; /// trie树节点上限 = maxw * maxwl
const int sigma_size = 26;
const int maxl = 300000 + 5; /// 文本串最大长度
const int maxw = 4000 + 5;   /// 单词最大个数
const int maxwl = 100 + 5;   /// 每个单词最大长度
const int MOD = 20071027;

struct Trie
{
    int ch[maxnode][sigma_size]; /// ch[node_id][nextchar]表示第id号节点下的节点对应
的id号
    int val[maxnode];
    int sz; /// 结点总数
    void clear()
    {
        sz = 1;    /// 初始时只有一个根结点
        memset(ch[0], 0, sizeof(ch[0]));
    }

    /// 插入字符串s，附加信息为v。注意v必须非0，因为0代表“本结点不是单词结点”
    void insert(const char *s, int v)
    {
        int u = 0, n = strlen(s), c;
        for (int i = 0; i < n; ++i)
        {
            c = s[i] - 'a';
            if (ch[u][c] == 0)  /// 结点不存在
            {
                memset(ch[sz], 0, sizeof(ch[sz]));
                val[sz] = 0;  /// 中间结点的附加信息为0
                ch[u][c] = sz++; /// 新建结点
            }
            u = ch[u][c]; /// 往下走
        }
        val[u] = v; /// 字符串的最后一个字符的附加信息为v
    }

    /// 找字符串s的前缀(前缀长度<=len)
    void find_prefixes(const char *s, int len, vector<int> &ans)
    {
        int u = 0, c;
        for (int i = 0; s[i] && i < len; ++i)
        {
            c = s[i] - 'a';
            if (ch[u][c] == 0) break;
            u = ch[u][c];
            if (val[u]) ans.push_back(val[u]); /// 找到一个前缀
```

```
        }
    }
} trie;
```

Splay

```
const int MAX_N = 50000 + 10;
const int INF = ~0U >> 1;
struct Node
{
    Node *ch[2], *p;
    int size, val, mx;
    int add;
    bool rev;
    Node()
    {
        size = 0;
        val = mx = -INF;
        add = 0;
    }
    bool d()
    {
        return this == p->ch[1];
    }
    void setc(Node *c, int d)
    {
        ch[d] = c;
        c->p = this;
    }
    void addIt(int ad)
    {
        add += ad;
        mx += ad;
        val += ad;
    }
    void revIt()
    {
        rev ^= 1;
    }
    void relax();
    void upd()
    {
        size = ch[0]->size + ch[1]->size + 1;
        mx = max(val, max(ch[0]->mx, ch[1]->mx));
    }
} Tnull, *null = &Tnull;
Node mem[MAX_N], *C = mem;

void Node::relax()
{
    if (add != 0)
    {
        for (int i = 0; i < 2; ++i)
        {
            if (ch[i] != null)
                ch[i]->addIt(add);
```

```
        }
        add = 0;
    }
    if (rev)
    {
        swap(ch[0], ch[1]);
        for (int i = 0; i < 2; ++i)
        {
            if (ch[i] != null)
                ch[i]->revIt();
        }
        rev = 0;
    }
}

Node *make(int v)
{
    C->ch[0] = C->ch[1] = null;
    C->size = 1;
    C->val = v;
    C->mx = v;
    C->add = 0;
    C->rev = 0;
    return C++;
}

Node *build(int l, int r)
{
    if (l >= r)
        return null;
    int m = (l + r) >> 1;
    Node *t = make(0);
    t->setc(build(l, m), 0);
    t->setc(build(m + 1, r), 1);
    t->upd();
    return t;
}

Node *root;

Node *rot(Node *t)
{
    Node *p = t->p;
    p->relax();
    t->relax();
    int d = t->d();
    p->p->setc(t, p->d());
    p->setc(t->ch[!d], d);
    t->setc(p, !d);
    p->upd();
    if (p == root)
        root = t;
}

void splay(Node *t, Node *f = null)
{
```

```
    while (t->p != f)
    {
        if (t->p->p == f)
            rot(t);
        else
            t->d() == t->p->d() ? (rot(t->p), rot(t)) : (rot(t), rot(t));
    }
    t->upd();
}

Node *select(int k)
{
    for (Node *t = root;;)
    {
        t->relax();
        int c = t->ch[0]->size;
        if (k == c)
            return t;
        if (k > c)
            k -= c + 1, t = t->ch[1];
        else
            t = t->ch[0];
    }
}

Node *&get(int l, int r)  //[l,r)
{
    Node *L = select(l - 1);
    Node *R = select(r);
    splay(L);
    splay(R, L);
    return R->ch[0];
}

int n, m;

int main()
{
    cin >> n >> m;
    root = build(0, n + 2);
    root->p = null;
    for (int i = 0; i < m; ++i)
    {
        int k, l, r, v;
        scanf("%d%d%d", &k, &l, &r);
        Node *&t = get(l, r + 1);
        if (k == 1)
        {
            scanf("%d", &v);
            t->addIt(v);
            splay(t);
        }
        else if (k == 2)
        {
            t->revIt();
            splay(t);
```

```
        }
        else
        {
            printf("%d\n", t->mx);
        }
    }
}
```

Map

```
typedef pair<string, int> PAIR;
struct cmp
{
    bool operator()(const PAIR &lhs, const PAIR &rhs)
    {
        return lhs.second > rhs.second;
    }
};
int main()
{
    int n;
    while (cin >> n && n)
    {
        string t;
        map<string, int> m;
        for (int i = 0; i < n; ++i)
            cin >> t, m[t]++;
        vector<PAIR> v(m.begin(), m.end());
        sort(v.begin(), v.end(), cmp());
        cout << v[0].first << endl;
    }
    return 0;
}
```

Suffix Array

```
#include <algorithm>
#include <numeric>
#include <cassert>
const int MAX_LEN = 100000;
using namespace std;
struct SuffixArray {
      int n;
      int m[2][MAX_LEN];
      int sa[MAX_LEN];

      void indexSort(int sa[], int ord[], int id[], int nId) { //ord is the
ordering get from prev stage
            static int cnt[MAX_LEN];
            memset(cnt, 0, sizeof(0) * nId);

            for (int i = 0; i < n; ++i) {
                  cnt[id[i]]++;
            }
            partial_sum(cnt, cnt + nId, cnt);
            for (int i = n - 1; i >= 0; --i) {
                  sa[--cnt[id[ord[i]]]] = ord[i];
```

```
            }
      }

      int*id, *oId;
      void init(int s[], int _n) { //s[n] == 0
            n = _n;
            assert(s[n - 1] == *min_element(s, s + n));
            static int w[MAX_LEN];
            memcpy(w, s, sizeof(int) * n);
            sort(w, w + n);
            int nId = unique(w, w + n) - w;
            id = m[0], oId = m[1];
            for (int i = 0; i < n; ++i) {
                  id[i] = lower_bound(w, w + nId, s[i]) - w;
            }
            static int ord[MAX_LEN];
            for (int i = 0; i < n; ++i) {
                  ord[i] = i;
            }
            indexSort(sa, ord, id, nId);
            for (int k = 1; k <= n && nId < n; k <<= 1) {
                  //get the prev order
                  // k -> k*2
                  int cur = 0;
                  for (int i = n - k; i < n; ++i) {
                        ord[cur++] = i;
                  }
                  for (int i = 0; i < n; ++i) {
                        if (sa[i] >= k)
                              ord[cur++] = sa[i] - k;
                  }
                  indexSort(sa, ord, id, nId);
                  //get new id
                  cur = 0;
                  swap(oId, id);
                  for (int i = 0; i < n; ++i) {
                        int c = sa[i], p = i ? sa[i - 1] : 0;
                        id[c] = (i == 0 || oId[c] != oId[p] || oId[c + k] !=
oId[p + k]) ? cur++ : cur - 1;
                  }
                  nId = cur;
            }
      }
} sa;
```

AC Auto Maton

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

const int CHARSET = 26;
```

```
const int MAX_N_NODES = int(3e5) + 10;

int pointer;
struct Node {
	Node*ch[CHARSET], *fail, *par;
	Node() {
		memset(ch, 0, sizeof ch);
		fail = 0;
	}
	Node*go(int w);
}*root;

Node nodePool[MAX_N_NODES], *cur;

Node*newNode() {
	Node*t = cur++;
	memset(t->ch, 0, sizeof t->ch);
	t->fail = 0;
	return t;
}

Node* Node::go(int w) {
	if (ch[w] == 0) {
		ch[w] = newNode();
		ch[w]->par = this;
	}
	return ch[w];
}

void init() {
	cur = nodePool;
	root = newNode();
	root->par = 0;
}

void build() {
	static Node*que[MAX_N_NODES];
	int qh = 0, qt = 0;
	que[qt++] = root;
	while (qh < qt) {
		Node*t = que[qh++];
		for (int c = 0; c < CHARSET; ++c) {
			Node*v = t->ch[c];
			if (!v)
				continue;
			Node*f = t->fail;
			while (f && f->ch[c] == 0)
				f = f->fail;
			if (f == 0)
				v->fail = root;
			else
				v->fail = f->ch[c];
			que[qt++] = v;
		}
	}
}
```