ICPC Templates For Africamonkey

Africamonkey

2018 年 7 月 27 日

目录

1 莫队算法

1.1 普通莫队

分块块数为 $\sqrt{n}$ 是最优的。

记每次进行 add() 操作的复杂度为 $O(A)$ ，del() 操作的复杂度为 $O(D)$ ，查询答案 answer() 的复杂度为 $O(S)$ 。

则总复杂度为 $O(n\sqrt{n}(A+D)+qS)$ 。

S 可以大一点，但必须保证 A, D 尽可能小。

```
struct Q { int l, r, sqrtl, id; } q[N];
int n, m, v[N], ans[N], nowans;
bool cmp(const Q &a, const Q &b) {
    if (a.sqrtl != b.sqrtl) return a.sqrtl < b.sqrtl;
    return a.r < b.r;
}
void change(int x) {
    if (!v[x]) add(x);
    else del(x);
    v[x] ^= 1;
}
int main() {
    ......
    for (int i=1;i<=m;i++) q[i].sqrtl = q[i].l / sqrt(n), q[i].id = i;
    sort(q+1, q+m+1, cmp);
    int L=1, R=0;
    memset(v, 0, sizeof(v));
    for (int i=1;i<=m;i++) {
        while (L<q[i].l) change(L++);
        while (L>q[i].l) change(--L);
        while (R<q[i].r) change(++R);
        while (R>q[i].r) change(R--);
        ans[q[i].id] = answer();
    }
    ......
}
```

1.2 树上莫队

分块块数为 $\sqrt{n}$ 是最优的。

记每次进行 add() 操作的复杂度为 $O(A)$ ，del() 操作的复杂度为 $O(D)$ ，查询答案 answer() 的复杂度为 $O(S)$ 。

则总复杂度为 $O(n\sqrt{n}(A+D)+qS)$ 。

S 可以大一点，但必须保证 A, D 尽可能小。

```
struct Query { int l, r, id, l_group; } query[N];
struct EDGE { int adj, next; } edge[N*2];
int n, m, top, gh[N], c[N], reorder[N], deep[N], father[N], size[N], son[N], Top[N];
void addedge(int x, int y) {
```

```
    edge[++top].adj = y;
    edge[top].next = gh[x];
    gh[x] = top;
}
void dfs(int x, int root=0) {
    reorder[x] = ++top; father[x] = root; deep[x] = deep[root] + 1;
    son[x] = 0; size[x] = 1; int dd = 0;
    for (int p=gh[x]; p; p=edge[p].next)
        if (edge[p].adj != root) {
            dfs(edge[p].adj, x);
            if (size[edge[p].adj] > dd) {
                son[x] = edge[p].adj;
                dd = size[edge[p].adj];
            }
            size[x] += size[edge[p].adj];
        }
}
void split(int x, int tp) {
    Top[x] = tp;
    if (son[x]) split(son[x], tp);
    for (int p=gh[x]; p; p=edge[p].next)
        if (edge[p].adj != father[x] && edge[p].adj != son[x])
            split(edge[p].adj, edge[p].adj);
}
int lca(int x, int y) {
    int tx = Top[x], ty = Top[y];
    while (tx != ty) {
        if (deep[tx] < deep[ty]) {
            swap(tx, ty);
            swap(x, y);
        }
        x = father[tx];
        tx = Top[x];
    }
    if (deep[x] < deep[y]) swap(x, y);
    return y;
}
bool cmp(const Query &a, const Query &b) {
    if (a.l_group != b.l_group) return a.l_group < b.l_group;
    return reorder[a.r] < reorder[b.r];
}
int v[N], ans[N];

void upd(int x) {
    if (!v[x]) add(x);
    else del(x);
    v[x] ^= 1;
}

void go(int &u, int taru, int v) {
```

```
    int lca0 = lca(u, taru);
    int lca1 = lca(u, v);   upd(lca1);
    int lca2 = lca(taru, v); upd(lca2);
    for (int x=u; x!=lca0; x=father[x]) upd(x);
    for (int x=taru; x!=lca0; x=father[x]) upd(x);
    u = taru;
}
int main() {
    memset(gh, 0, sizeof(gh));
    scanf("%d%d", &n, &m); top = 0;
    for (int i=1;i<n;i++) {
        int x,y; scanf("%d%d", &x, &y);
        addedge(x, y); addedge(y, x);
    }
    top = 0; dfs(1); split(1, 1);
    for (int i=1;i<=m;i++) {
        if (reorder[query[i].l] > reorder[query[i].r])
            swap(query[i].l, query[i].r);
        query[i].id = i;
        query[i].l_group = reorder[query[i].l] / sqrt(n);
    }
    sort(query+1, query+m+1, cmp);
    int L=1,R=1; upd(1);
    for (int i=1;i<=m;i++) {
        go(L,query[i].l,R);
        go(R,query[i].r,L);
        ans[query[i].id] = answer();
    }
    ......
}
```

2 字符串

2.1 哈希

```
const int P=31,D=1000173169;
int n, pow[N], f[N]; char a[N];
int hash(int l, int r) { return (LL)(f[r]-(LL)f[l-1]*pow[r-l+1]%D+D)%D; }
int main() {
    scanf("%d%s", &n, a+1);
    pow[0] = 1;
    for (int i=1;i<=n;i++) pow[i] = (LL)pow[i-1]*P%D;
    for (int i=1;i<=n;i++) f[i] = (LL)((LL)f[i-1]*P+a[i])%D;
}
```

2.2 KMP

接口：void kmp(int n, char *a, int m, char *b);

输入：模式串长度 n ，模式串 a ，匹配串长度 m ，匹配串 b

输出：依次输出每个匹配成功的起始位置

下标从 0 开始。

```
void kmp(int n, char* a, int m, char *b) {
    int i, j;
    for (nxt[0] = j = -1, i = 1; i < n; nxt[i++] = j) {
        while (~j && a[j + 1] != a[i]) j = nxt[j];
        if (a[j + 1] == a[i]) ++j;
    }
    for (j = -1, i = 0; i < m; ++i) {
        while (~j && a[j + 1] != b[i]) j = nxt[j];
        if (a[j + 1] == b[i]) ++j;
        if (j == n - 1) {
            printf("%d\n", i - n + 1);
            j = nxt[j];
        }
    }
}
```

2.3 可动态修改的 KMP

支持：加入一个字符，删除一个字符。

时间复杂度：$O(n\alpha)$ ，α 为字符集大小。

代码中的字符为 $'0'-'9'$ ，可自行修改为 $'a'-'z'$

```
char t[N];
int top, nxt[N], nxt_l[N][10];
inline void del_letter() { --top; }
inline void add_letter(char x) {
    t[top++] = x;
    int j = top-1;
    memset(nxt_l[top], 0, sizeof(nxt_l[top]));
    nxt[top] = nxt_l[top-1][x-'0'];
    memcpy(nxt_l[top], nxt_l[nxt[top]], sizeof(nxt_l[nxt[top]]));
    nxt_l[top][t[nxt[top]]-'0'] = nxt[top]+1;
}
```

2.4 扩展 KMP

接口：void ExtendedKMP(char *a, char *b, int *next, int *ret);

输出：

next: a 关于自己每个后缀的最长公共前缀

ret: a 关于 b 的每个后缀的最长公共前缀

EXKMP 的 next[i] 表示：从 i 到 n-1 的字符串 st 前缀和原串前缀的最长重叠长度。

```
void get_next(char *a, int *next) {
    int i, j, k;
    int n = strlen(a);
```

```
    for (j = 0; j+1<n && a[j]==a[j+1];j++);
    next[1] = j;
    k = 1;
    for (i=2;i<n;i++) {
        int len = k+next[k], l = next[i-k];
        if (l < len-i) {
            next[i] = l;
        } else {
            for (j = max(0, len-i);i+j<n && a[j]==a[i+j];j++);
            next[i] = j;
            k = i;
        }
    }
}
void ExtendedKMP(char *a, char *b, int *next, int *ret) {
    get_next(a, next);
    int n = strlen(a), m = strlen(b);
    int i, j, k;
    for (j=0;j<n && j<m && a[j]==b[j];j++);
    ret[0] = j;
    k = 0;
    for (i=1;i<m;i++) {
        int len = k+ret[k], l = next[i-k];
        if (l < len-i) {
            ret[i] = l;
        } else {
            for (j = max(0, len-i);j<n && i+j<m && a[j]==b[i+j];j++);
            ret[i] = j;
            k = i;
        }
    }
}
```

2.5 Manacher

p[i] 表示以 i 为对称轴的最长回文串长度

```
char st[N*2], s[N];
int len, p[N*2];

while (scanf("%s", s) != EOF) {
    len = strlen(s);
    st[0] = '$', st[1] = '#';
    for (int i=1;i<=len;i++)
        st[i*2] = s[i-1], st[i*2+1] = '#';
    len = len * 2 + 2;
    int mx = 0, id = 0, ans = 0;
    for (int i=1;i<=len;i++) {
        p[i] = (mx > i) ? min(p[id*2-i]+1, mx-i) : 1;
        for (; st[i+p[i]] == st[i-p[i]]; ++p[i]) ;
```

```
        if (p[i]+i > mx) mx = p[i]+i, id = i;
        p[i] --;
        if (p[i] > ans) ans = p[i];
    }
    printf("%d\n", ans);
}
```

2.6 最小表示法

```
string smallestRepresation(string s) {
    int i, j, k, l;
    int n = s.length();
    s += s;
    for (i=0,j=1;j<n;) {
        for (k=0;k<n && s[i+k]==s[j+k];k++);
        if (k>=n) break;
        if (s[i+k]<s[j+k]) j+=k+1;
        else {
            l=i+k;
            i=j;
            j=max(l,j)+1;
        }
    }
    return s.substr(i, n);
}
```

2.7 AC 自动机

```
struct Node {
    int next[**Size of Alphabet**];
    int terminal, fail;
} node[**Number of Nodes**];
int top;
void add(char *st) {
    int len = strlen(st), x = 1;
    for (int i=0;i<len;i++) {
        int ind = trans(st[i]);
        if (!node[x].next[ind])
            node[x].next[ind] = ++top;
        x = node[x].next[ind];
    }
    node[x].terminal = 1;
}
int q[**Number of Nodes**], head, tail;
void build() {
    head = 0, tail = 1; q[1] = 1;
    while (head != tail) {
        int x = q[++head];
```

```
        /*(when necessary) node[x].terminal |= node[node[x].fail].terminal; */
        for (int i=0;i<n;i++)
            if (node[x].next[i]) {
                if (x == 1) node[node[x].next[i]].fail = 1;
                else {
                    int y = node[x].fail;
                    while (y) {
                        if (node[y].next[i]) {
                            node[node[x].next[i]].fail = node[y].next[i];
                            break;
                        }
                        y = node[y].fail;
                    }
                    if (!node[node[x].next[i]].fail) node[node[x].next[i]].fail = 1;
                }
                q[++tail] = node[x].next[i];
            }
    }
}
```

2.8 后缀数组

2.8.1 倍增算法

参数 m 表示字符集的大小，即 $0 \le r_i < m$

```
#define rank rank2
int n, r[N], wa[N], wb[N], ws[N], sa[N], rank[N], height[N];
int cmp(int *r, int a, int b, int l, int n) {
    if (r[a]==r[b]) {
        if (a+l<n && b+l<n && r[a+l]==r[b+l])
            return 1;
    }
    return 0;
}
void suffix_array(int m) {
    int i, j, p, *x=wa, *y=wb, *t;
    for (i=0;i<m;i++) ws[i]=0;
    for (i=0;i<n;i++) ws[x[i]=r[i]]++;
    for (i=1;i<m;i++) ws[i]+=ws[i-1];
    for (i=n-1;i>=0;i--) sa[--ws[x[i]]]=i;
    for (j=1,p=1;p<n;m=p,j<<=1) {
        for (p=0,i=n-j;i<n;i++) y[p++]=i;
        for (i=0;i<n;i++) if (sa[i]>=j) y[p++]=sa[i]-j;
        for (i=0;i<m;i++) ws[i]=0;
        for (i=0;i<n;i++) ws[x[y[i]]]++;
        for (i=1;i<m;i++) ws[i]+=ws[i-1];
        for (i=n-1;i>=0;i--) sa[--ws[x[y[i]]]]=y[i];
        for (t=x,x=y,y=t,x[sa[0]]=0,i=1,p=1;i<n;i++)
            x[sa[i]]=cmp(y,sa[i-1],sa[i],j,n)?p-1:p++;
```

```cpp
    }
    for (i=0;i<n;i++) rank[sa[i]]=i;
    rank[n] = -1;
}
void calc_height() {
    int j=0;
    for (int i=0;i<n;i++)
        if (rank[i])
        {
            while (r[i+j]==r[sa[rank[i]-1]+j]) j++;
            height[rank[i]]=j;
            if (j) j--;
        }
}
```

2.8.2 DC3 算法

感谢浙江大学陈靖邦提供本模板。

```cpp
namespace SA {
int sa[N], rk[N], ht[N], s[N<<1], t[N<<1], p[N], cnt[N], cur[N];
#define pushS(x) sa[cur[s[x]]--] = x
#define pushL(x) sa[cur[s[x]]++] = x
#define inducedSort(v) fill_n(sa, n, -1); fill_n(cnt, m, 0);                  \
    for (int i = 0; i < n; i++) cnt[s[i]]++;                                  \
    for (int i = 1; i < m; i++) cnt[i] += cnt[i-1];                           \
    for (int i = 0; i < m; i++) cur[i] = cnt[i]-1;                            \
    for (int i = n1-1; ~i; i--) pushS(v[i]);                                  \
    for (int i = 1; i < m; i++) cur[i] = cnt[i-1];                            \
    for (int i = 0; i < n; i++) if (sa[i] > 0 &&  t[sa[i]-1]) pushL(sa[i]-1); \
    for (int i = 0; i < m; i++) cur[i] = cnt[i]-1;                            \
    for (int i = n-1;  ~i; i--) if (sa[i] > 0 && !t[sa[i]-1]) pushS(sa[i]-1)
void sais(int n, int m, int *s, int *t, int *p) {
    int n1 = t[n-1] = 0, ch = rk[0] = -1, *s1 = s+n;
    for (int i = n-2; ~i; i--) t[i] = s[i] == s[i+1] ? t[i+1] : s[i] > s[i+1];
    for (int i = 1; i < n; i++) rk[i] = t[i-1] && !t[i] ? (p[n1] = i, n1++) : -1;
    inducedSort(p);
    for (int i = 0, x, y; i < n; i++) if (~(x = rk[sa[i]])) {
        if (ch < 1 || p[x+1] - p[x] != p[y+1] - p[y]) ch++;
        else for (int j = p[x], k = p[y]; j <= p[x+1]; j++, k++)
            if ((s[j]<<1|t[j]) != (s[k]<<1|t[k])) {ch++; break;}
        s1[y = x] = ch;
    }
    if (ch+1 < n1) sais(n1, ch+1, s1, t+n, p+n1);
    else for (int i = 0; i < n1; i++) sa[s1[i]] = i;
    for (int i = 0; i < n1; i++) s1[i] = p[sa[i]];
    inducedSort(s1);
}
template<typename T>
int mapCharToInt(int n, const T *str) {
```

```
    int m = *max_element(str, str+n);
    fill_n(rk, m+1, 0);
    for (int i = 0; i < n; i++) rk[str[i]] = 1;
    for (int i = 0; i < m; i++) rk[i+1] += rk[i];
    for (int i = 0; i < n; i++) s[i] = rk[str[i]] - 1;
    return rk[m];
}
// Ensure that str[n] is the unique lexicographically smallest character in str.
template<typename T>
void suffixArray(int n, const T *str) {
    int m = mapCharToInt(++n, str);
    sais(n, m, s, t, p);
    for (int i = 0; i < n; i++) rk[sa[i]] = i;
    for (int i = 0, h = ht[0] = 0; i < n-1; i++) {
        int j = sa[rk[i]-1];
        while (i+h < n && j+h < n && s[i+h] == s[j+h]) h++;
        if (ht[rk[i]] = h) h--;
    }
}
};
```

2.8.3 小技巧：拼接字符串

接口：

int gao1(int l, int r, int c, int p); 区间 $[l,r)$ 中保证第 0 位到第 $c-1$ 位都是相同的（设为字符串 s ），现在我们在 s 后面接一个字符 p ，得到一个新的字符串 s' 。返回值为最小的 k 满足后缀 $sa[k]$ 前 $c+1$ 位为 s'

int gao2(int l, int r, int c, int p); 区间 $[l,r)$ 中保证第 0 位到第 $c-1$ 位都是相同的（设为字符串 s ），现在我们在 s 后面接一个后缀 $sa[p]$ ，得到一个新的字符串 s' 。返回值为最小的 k 满足后缀 $sa[k]$ 前 $c+len(sa[p])$ 位为 s'

```
int gao1(int l,int r,int c,int p) {
        --l;
        while (l+1<r) {
                int md=(l+r)>>1;
                if (sa[md]+c<n&&s[sa[md]+c]>=p) r=md; else l=md;
        }
        return r;
}
int gao2(int l,int r,int c,int p) {
        --l;
        while (l+1<r) {
                int md=(l+r)>>1;
                if (sa[md]+c<=n&&rk[sa[md]+c]>=p) r=md; else l=md;
        }
        return r;
}
```

示例调用：

```
suf1[m] = -1, suf2[m] = n;
for (int i = m - 1; i >= 0; --i) {
    int l = gao1(0, n, 0, t[i]), r = gao1(0, n, 0, t[i]);
    suf1[i] = gao2(l, r, 1, suf1[i + 1]);
    suf2[i] = gao2(l, r, 1, suf2[i + 1]);
}
```

2.9 后缀自动机

下面的代码是求两个串的 LCS（最长公共子串）。

```
#include <bits/stdc++.h>

#define N 500001
#define M (N << 1)

using namespace std;

char st[N];
int pre[M], son[26][M], step[M], refer[M], size[M], tmp[M], topo[M], last, total;

int apply(int x, int now) {
    step[++total] = x;
    refer[total] = now;
    return total;
}

void extend(char x, int now) {
    int p = last, np = apply(step[last]+1, now);
    size[np] = 1;
    for (; p && !son[x][p]; p=pre[p]) son[x][p] = np;
    if (!p) pre[np] = 1;
    else {
        int q = son[x][p];
        if (step[p]+1 == step[q]) pre[np] = q;
        else {
            int nq = apply(step[p]+1, now);
            for (int i=0;i<26;i++) son[i][nq] = son[i][q];
            pre[nq] = pre[q];
            pre[q] = pre[np] = nq;
            for (; p && son[x][p]==q; p=pre[p]) son[x][p] = nq;
        }
    }
    last = np;
}
void init() {
    last = total = 0;
    last = apply(0, 0);
    scanf("%s",st);
```

```
    int n = strlen(st);
    for (int i = 0; i <= n * 2; ++i) {
        pre[i] = step[i] = refer[i] = size[i] = tmp[i] = topo[i] = 0;
        for (int j = 0; j < 26; ++j)
            son[j][i] = 0;
    }
    for (int i = 0; i < n; ++i)
        extend(st[i] - 'a', i);
    for (int i = 1; i <= total; ++i)
        tmp[step[i]] ++;
    for (int i = 1; i <= n; ++i)
        tmp[i] += tmp[i - 1];
    for (int i = 1; i <= total; ++i)
        topo[tmp[step[i]]--] = i;
    for (int i = total; i; --i)
        size[pre[topo[i]]] += size[topo[i]];
}
int main() {
    init();
    int p = 1, now = 0, ans = 0;
    scanf("%s", st);
    for (int i=0; st[i]; i++) {
        int index = st[i]-'a';
        for (; p && !son[index][p]; p = pre[p], now = step[p]) ;
        if (!p) p = 1;
        if (son[index][p]) {
            p = son[index][p];
            now++;
            if (now > ans) ans = now;
        }
    }
    printf("%d\n",ans);
    return 0;
}
```

一些定义和性质 Right(str) 表示 str 在母串 S 中所有出现的结束位置集合

一个状态 s 表示的所有子串 Right 集合相同，为 Right(s)

Parent(s) 满足 Right(s) 是 Right(Parent(s)) 的真子集，并且 Right(Parent(s)) 的大小最小

Parent 函数可以表示一个树形结构。不妨叫它 Parent 树

一个 Right 集合和一个长度定义了一个子串

对于状态 s ，使得 Right(s) 合法的子串长度是一个区间 [min(s), max(s)]

max(Parent(s)) = min(s) - 1

令 refer(s) 表示产生 s 状态的字符所在位置。则 Right(s) 的合法子串的起始位置为 [refer(s) - max(s) + 1, refer(s) - min(s) + 1] ，即 [refer(s) - max(s) + 1, refer(s) - max(Parent(s))]

代码中变量名含义 pre[s] 为上述定义中的 Parent(s)

step[s] 为从初始状态走到 s 状态最多需要多少步

refer[s] 为上述定义中的 refer(s)
size[s] 为 Right(s) 集合的大小
topo[s] 为 Parent 树的拓扑序，根（初始状态）在前

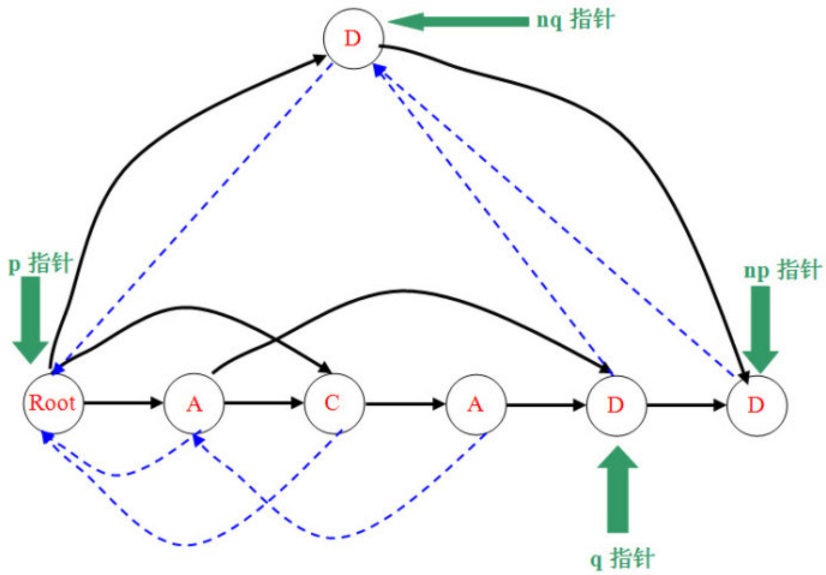


图 1: ACADD 构成的后缀自动机

我们发现 fail 构出一棵前缀树
和后缀树相同，为了使每个前缀都是叶子结点，我们不妨在串 s 前加入一个没出现的字符'#'

2.9.1 广义后缀自动机

先建 Trie ，再按照 *BFS* 序建后缀自动机。从节点 x 开始向子树更新时，其所有儿子都从同一个 last ，即 $last[x]$ 更新。

2.10 回文树

【URAL2040】Palindromes and Super Abilities 2
逐个添加字符串 S 里的字符 $S_1, S_2, ..., S_n$ 。每次添加字符后，他想知道添加字符后将出现多少个新的本质不同的回文子串。字符集为 $\{a, b\}$

```
#include <bits/stdc++.h>
#define N 5000020

char st[N], answer[N];
int n;

struct PAM {
    int n, tot, last;
    int len[N], fail[N], next[N][2];
    void init() {
        n=0; tot=1;
```

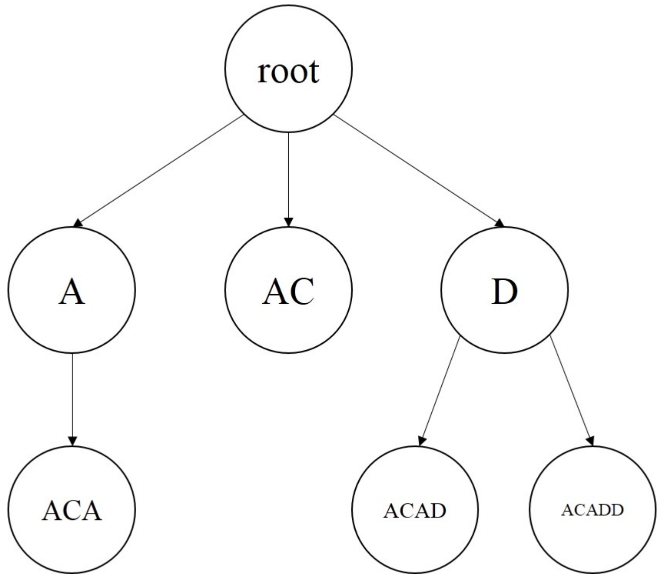


图 2: 串 ACADD 按 fail 构出的前缀树，与图 1 对应

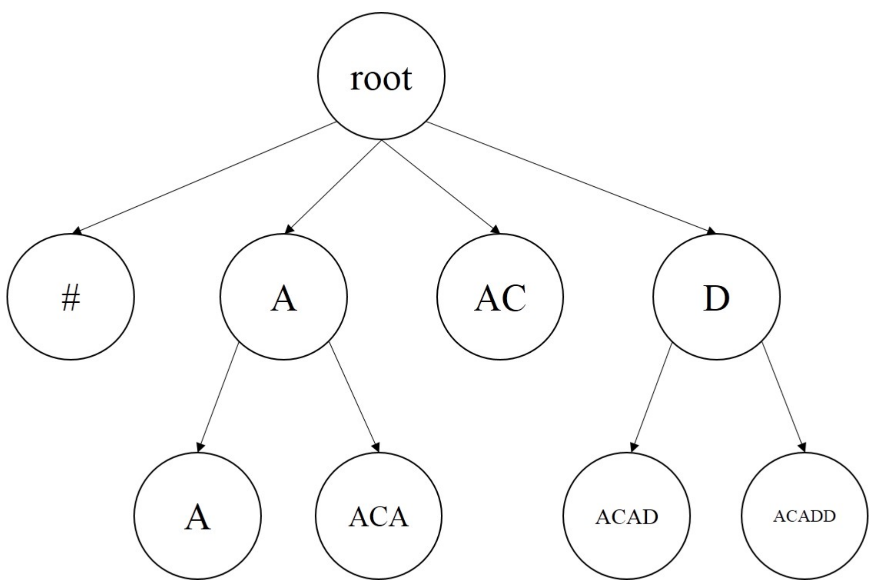


图 3: 串 #ACADD 按 fail 构出的前缀树

```
        len[1]=-1; fail[1]=0;
        len[0]=+0; fail[0]=1;
        last=1;
    }
    int get_fail(int x) {
        for (; st[n-len[x]-1]!=st[n]; x=fail[x]);
        return x;
    }
    void insert(char c) {
        ++n; int cur=get_fail(last); // 判断上一个串的前一个位置和新添加的位置是否相同，相同则说明构成回文。否则找 fail 指针。
        if (!next[cur][c]) {
            ++tot;
            len[tot]=len[cur]+2;
            fail[tot]=next[get_fail(fail[cur])][c];
            next[cur][c]=tot;
            answer[n]='1';
        } else {
            answer[n]='0';
        }
        last=next[cur][c];
    }
} pam;

int main() {
    scanf("%s", st+1); n=strlen(st+1);
    pam.init();
    for (int i=1;i<=n;i++) pam.insert(st[i]-'a');
    puts(answer+1);
    return 0;
}
```

3 数据结构

3.1 ST 表

```
int Log[N],f[17][N];
int ask(int x,int y){
    int k=Log[y-x+1];
    return max(f[k][x],f[k][y-(1<<k)+1]);
}
int main(){
    for (int i=2;i<=n;i++)Log[i]=Log[i>>1]+1;
    for (int j=1;j<K;j++)
        for (int i=1;i+(1<<j-1)<=n;i++)
            f[j][i]=max(f[j-1][i],f[j-1][i+(1<<j-1)]);
}
```

3.2 K-D Tree

```
int n, cmp_d, root, id[N];

struct node {
    int d[2], l, r, Max[2], Min[2], val, sum, f;
} t[N];

inline bool cmp(const node &a, const node &b) {
    if (a.d[cmp_d] != b.d[cmp_d]) return a.d[cmp_d] < b.d[cmp_d];
    return a.d[cmp_d ^ 1] < b.d[cmp_d ^ 1];
}

inline void umax(int &a, int b) {
    if (b > a) a = b;
}

inline void umin(int &a, int b) {
    if (b < a) a = b;
}

inline void up(int x, int y) {
    umax(t[x].Max[0], t[y].Max[0]);
    umin(t[x].Min[0], t[y].Min[0]);
    umax(t[x].Max[1], t[y].Max[1]);
    umin(t[x].Min[1], t[y].Min[1]);
}

int build(int l, int r, int D, int f) {
    int mid = (l + r) / 2;
    cmp_d = D;
    nth_element(t + l + 1, t + mid + 1, t + r + 1, cmp);
    id[t[mid].f] = mid;
    t[mid].f = f;
    t[mid].Max[0] = t[mid].Min[0] = t[mid].d[0];
    t[mid].Max[1] = t[mid].Min[1] = t[mid].d[1];
    t[mid].val = t[mid].sum = 0;
    if (l != mid) t[mid].l = build(l, mid - 1, !D, mid);
    else t[mid].l = 0;
    if (r != mid) t[mid].r = build(mid + 1, r, !D, mid);
    else t[mid].r = 0;
    if (t[mid].l) up(mid, t[mid].l);
    if (t[mid].r) up(mid, t[mid].r);
    return mid;
}

// 将编号为 x 的点的权值增加 p
// 请注意，此处的 x 是经过排序的。你需要将点的坐标先作映射。
void change(int x, int p) {
    x = id[x];
```

```
    for (t[x].val += p; x; x = t[x].f)
        t[x].sum += p;
}

inline long long sqr(long long x) {
    return x * x;
}

// 欧几里得距离的平方，下界
inline long long euclid_lower_bound(const node &a, int X, int Y) {
    return sqr(max(max(X - a.Max[0], a.Min[0] - X), 0)) +
        sqr(max(max(Y - a.Max[1], a.Min[1] - Y), 0));
}

// 欧几里得距离的平方，上界
inline long long euclid_upper_bound(const node &a, int X, int Y) {
    return max(sqr(X - a.Min[0]), sqr(X - a.Max[0])) +
        max(sqr(Y - a.Min[1]), sqr(Y - a.Max[1]));
}

// 曼哈顿距离，下界
inline long long manhattan_lower_bound(const node &a, int X, int Y) {
    return max(a.Min[0] - X, 0) + max(X - a.Max[0], 0) +
        max(a.Min[1] - Y, 0) + max(Y - a.Max[1], 0);
}

// 曼哈顿距离，上界
inline long long manhattan_upper_bound(const node &a, int X, int Y) {
    return max(abs(X - a.Max[0]), abs(a.Min[0] - X)) +
        max(abs(Y - a.Max[1]), abs(a.Min[1] - Y));
}

// 添加一个点（注意此处的添加可能导致这棵树不平衡，慎用！）
void add(int k) {
    t[k].Max[0] = t[k].Min[0] = t[k].d[0];
    t[k].Max[1] = t[k].Min[1] = t[k].d[1];
    t[k].val = t[k].sum = 0;
    t[k].l = t[k].r = t[k].f = 0;
    if (!root) {
        root = k;
        return;
    }
    int p = root;
    int D = 0;
    while (1) {
        up(p, k);
        if (t[k].d[D] <= t[p].d[D]) {
            if (t[p].l) p = t[p].l;
            else {
                t[p].l = k;
```

```
                t[k].f = p;
                return;
            }
        } else {
            if (t[p].r) p = t[p].r;
            else {
                t[p].r = k;
                t[k].f = p;
                return;
            }
        }
        D ^= 1;
    }
}

inline long long getdis(const node &a, int X, int Y) {
    return sqr(a.d[0] - X) + sqr(a.d[1] - Y);
}

// 此处询问距离点 (X, Y) 最远的一个点的距离， ans 需传入无穷小
void ask(int p, int X, int Y, long long &ans) {
    if (!p) return;
    ans = max(ans, getdis(t[p], X, Y));
    long long dl = t[p].l ? euclid_upper_bound(t[t[p].l], X, Y) : 0;
    long long dr = t[p].r ? euclid_upper_bound(t[t[p].r], X, Y) : 0;
    if (dl > dr) {
        if (dl > ans) ask(t[p].l, X, Y, ans);
        if (dr > ans) ask(t[p].r, X, Y, ans);
    } else {
        if (dr > ans) ask(t[p].r, X, Y, ans);
        if (dl > ans) ask(t[p].l, X, Y, ans);
    }
}

// 查询矩形范围内所有点的权值和
int ask(int p, int x1, int y1, int x2, int y2) {
    if (t[p].Min[0] > x2 || t[p].Max[0] < x1 || t[p].Min[1] > y2 || t[p].Max[1] < y1
        ) return 0;
    if (t[p].Min[0] >= x1 && t[p].Max[0] <= x2 && t[p].Min[1] >= y1 && t[p].Max[1]
        <= y2) return t[p].sum;
    int s = 0;
    if (t[p].d[0] >= x1 && t[p].d[0] <= x2 && t[p].d[1] >= y1 && t[p].d[1] <= y2) s
        += t[p].val;
    if (t[p].l) s += ask(t[p].l, x1, y1, x2, y2);
    if (t[p].r) s += ask(t[p].r, x1, y1, x2, y2);
    return s;
}

int main() {
    while (~scanf("%d", &n)) {
```

```
        for (int i = 1; i <= n; ++i) {
            int x, y, type;
            scanf("%d%d%d", &x, &y, &type);
            t[i].d[0] = x;
            t[i].d[1] = y;
        }
        root = build(1, n, 0, 0);
    }
}
```

3.3 左偏树

左偏树是一个可并堆。

下面的程序写的是一个小根堆，如果需要改成大根堆请在注释了 here 那行修改。

接口：

void push(const T &x); 插入一个元素。

void merge(leftist &x); 合并两个堆。注意，合并后原来那个堆将不可访问。

T top() const; 返回堆顶元素。

void pop(); 删除堆顶元素。

int size() const; 返回堆的大小。

```
template <class T>
class leftist {
public:
    struct node {
        T key;
        int dist;
        node *l, *r;
    };
    leftist() : root(NULL), s(0) {}
    void push(const T &x) {
        leftist y;
        y.s = 1;
        y.root = new node;
        y.root -> key = x;
        y.root -> dist = 0;
        y.root -> l = y.root -> r = NULL;
        merge(y);
    }
    node* merge(node *x, node *y) {
        if (x == NULL) return y;
        if (y == NULL) return x;
        if (y -> key < x -> key) swap(x, y); //here
        x -> r = merge(x -> r, y);
        int ld = x -> l ? x -> l -> dist : -1;
        int rd = x -> r ? x -> r -> dist : -1;
        if (ld < rd) swap(x -> l, x -> r);
        if (x -> r == NULL) x -> dist = 0;
        else x -> dist = x -> r -> dist + 1;
```

```
        return x;
    }
    void merge(leftist &x) {
        root = merge(root, x.root);
        s += x.s;
    }
    T top() const {
        if (root == NULL) return T();
        return root -> key;
    }
    void pop() {
        if (root == NULL) return;
        node *p = root;
        root = merge(root -> l, root -> r);
        --s;
        delete p;
    }
    int size() const {
        return s;
    }
private:
    node* root;
    int s;
};
```

3.4 线段树小技巧

给定一个序列 a ，寻找一个最大的 i 使得 $i \le y$ 且满足一些条件（如 $a[i] \ge w$ ，那么需要在线段树维护 a 的区间最大值）

```
int queryl(int p, int left, int right, int y, int w) {
    if (right <= y) {
        if (! __condition__ ) return -1;
        else if (left == right) return left;
    }
    int mid = (left + right) / 2;
    if (y <= mid) return queryl(p<<1|0, left, mid, y, w);
    int ret = queryl(p<<1|1, mid+1, right, y, w);
    if (ret != -1) return ret;
    return queryl(p<<1|0, left, mid, y, w);
}
```

给定一个序列 a ，寻找一个最小的 i 使得 $i \ge x$ 且满足一些条件（如 $a[i] \ge w$ ，那么需要在线段树维护 a 的区间最大值）

```
int queryr(int p, int left, int right, int x, int w) {
    if (left >= x) {
        if (! __condition__ ) return -1;
        else if (left == right) return left;
    }
```

```
    int mid = (left + right) / 2;
    if (x > mid) return queryr(p<<1|1, mid+1, right, x, w);
    int ret = queryr(p<<1|0, left, mid, x, w);
    if (ret != -1) return ret;
    return queryr(p<<1|1, mid+1, right, x, w);
}
```

3.5 Splay

接口:

ADD x y d： 将 $[x, y]$ 的所有数加上 d

REVERSE x y： 将 $[x, y]$ 翻转

INSERT x p： 将 p 插入到第 x 个数的后面

DEL x： 将第 x 个数删除

```
struct SPLAY {
    struct NODE {
        int w, min;
        int son[2], size, father, rev, lazy;
    } node[N];
    int top, rt;
    void pushdown(int x) {
        if (!x) return;
        if (node[x].rev) {
            node[node[x].son[0]].rev ^= 1;
            node[node[x].son[1]].rev ^= 1;
            swap(node[x].son[0], node[x].son[1]);
            node[x].rev = 0;
        }
        if (node[x].lazy) {
            node[node[x].son[0]].lazy += node[x].lazy;
            node[node[x].son[1]].lazy += node[x].lazy;
            node[x].w += node[x].lazy;
            node[x].min += node[x].lazy;
            node[x].lazy = 0;
        }
    }
    void pushup(int x) {
        if (!x) return;
        pushdown(node[x].son[0]);
        pushdown(node[x].son[1]);
        node[x].size = node[node[x].son[0]].size + node[node[x].son[1]].size + 1;
        node[x].min = node[x].w;
        if (node[x].son[0]) node[x].min = min(node[x].min, node[node[x].son[0]].min)
            ;
        if (node[x].son[1]) node[x].min = min(node[x].min, node[node[x].son[1]].min)
            ;
    }
    void sc(int x, int y, int w) {
```

```
        node[x].son[w] = y;
        node[y].father = x;
        pushup(x);
    }
    void _ins(int w) {
        top++;
        node[top].w = node[top].min = w;
        node[top].son[0] = node[top].son[1] = 0;
        node[top].size = 1; node[top].father = 0; node[top].rev = 0;
    }
    void init() {
        top = 0;
        _ins(0); _ins(0); rt=1;
        sc(1, 2, 1);
    }
    void rotate(int x) {
        if (!x) return;
        int y = node[x].father;
        int w = node[y].son[1]==x;
        sc(y, node[x].son[w^1], w);
        sc(node[y].father, x, node[node[y].father].son[1]==y);
        sc(x, y, w^1);
    }
    int q[N];
    void flushdown(int x) {
        int t=0; for (; x; x=node[x].father) q[++t]=x;
        for (; t; t--) pushdown(q[t]);
    }
    void Splay(int x, int root=0) {
        flushdown(x);
        while (node[x].father != root) {
            int y=node[x].father;
            int w=node[y].son[1]==x;
            if (node[y].father != root && node[node[y].father].son[w]==y) rotate(y);
            rotate(x);
        }
    }
    int find(int k) {
        Splay(rt);
        while (1) {
            pushdown(rt);
            if (node[node[rt].son[0]].size+1==k) {
                Splay(rt);
                return rt;
            } else
            if (node[node[rt].son[0]].size+1<k) {
                k-=node[node[rt].son[0]].size+1;
                rt=node[rt].son[1];
            } else {
                rt=node[rt].son[0];
```

```
            }
        }
    }
    int split(int x, int y) {
        int fx = find(x);
        int fy = find(y+2);
        Splay(fx);
        Splay(fy, fx);
        return node[fy].son[0];
    }
    void add(int x, int y, int d) { //add d to each number in a[x]...a[y]
        int t = split(x, y);
        node[t].lazy += d;
        Splay(t); rt=t;
    }
    void reverse(int x, int y) { // reverse the x-th to y-th elements
        int t = split(x, y);
        node[t].rev ^= 1;
        Splay(t); rt=t;
    }
    void insert(int x, int p) { // insert p after the x-th element
        int fx = find(x+1);
        int fy = find(x+2);
        Splay(fx);
        Splay(fy, fx);
        _ins(p);
        sc(fy, top, 0);
        Splay(top); rt=top;
    }
    void del(int x) { // delete the x-th element in Splay
        int fx = find(x), fy = find(x+2);
        Splay(fx); Splay(fy, fx);
        node[fy].son[0] = 0;
        Splay(fy); rt=fy;
    }
} tree;
```

3.6 可持久化 Treap

接口：

void insert(int x, char c); 在当前第 x 个字符后插入 c

void del(int x, int y); 删除第 x 个字符到第 y 个字符

void copy(int l, int r, int x); 复制第 l 个字符到第 r 个字符，然后粘贴到第 x 个字符后

void reverse(int x, int y); 翻转第 x 个到第 y 个字符

char query(int k); 表示询问当前第 x 个字符是什么

```
#define mod 1000000007
struct Treap {
    struct Node {
```

```
        char key;
        bool reverse;
        int lc, rc, size; // if size is long long, remember here
    } node[N];
    int n, root, rd;
    int Rand() { rd = (rd * 20372052LL + 25022087LL) % mod; return rd; }

    /*
    LL Rand() {
        LL t1 = rand() % 32768;
        LL t2 = rand() % 32768;
        LL t3 = rand() % 32768;
        LL t4 = rand() % 32768;
        return (((t1 * 32768) + t2) * 32768 + t3) * 32768 + t4;
    }
    */

    void init() {
        n = root = 0;
    }
    inline int copy(int x) {
        node[++n] = node[x]; return n;
    }
    inline void pushdown(int x) {
        if (!node[x].reverse) return;
        if (node[x].lc) node[x].lc = copy(node[x].lc);
        if (node[x].rc) node[x].rc = copy(node[x].rc);
        swap(node[x].lc, node[x].rc);
        node[node[x].lc].reverse ^= 1;
        node[node[x].rc].reverse ^= 1;
        node[x].reverse = 0;
    }
    inline void pushup(int x) {
        node[x].size = node[node[x].lc].size + node[node[x].rc].size + 1;
    }
    int merge(int u, int v) {
        if (!u || !v) return u+v;
        pushdown(u); pushdown(v);
        int t = Rand() % (node[u].size + node[v].size), r;  // if size is long long,
             remember here
        if (t < node[u].size) {
            r = copy(u);
            node[r].rc = merge(node[u].rc, v);
        } else {
            r = copy(v);
            node[r].lc = merge(u, node[v].lc);
        }
        pushup(r);
        return r;
    }
```

```
    int split(int u, int x, int y) { // if size is long long, remember here
        if (x > y) return 0;
        pushdown(u);
        if (x == 1 && y == node[u].size) return copy(u);
        if (y <= node[node[u].lc].size) return split(node[u].lc, x, y);
        int t = node[node[u].lc].size + 1; // if size is long long, remember here
        if (x > t) return split(node[u].rc, x-t, y-t);
        int num = copy(u);
        node[num].lc = split(node[u].lc, x, t-1);
        node[num].rc = split(node[u].rc, 1, y-t);
        pushup(num);
        return num;
    }
    void insert(int x, char c) {
        int t1 = split(root, 1, x), t2 = split(root, x+1, node[root].size);
        node[++n].key = c;
        node[n].lc = node[n].rc = 0;
        node[n].reverse = 0;
        pushup(n);
        root = merge(merge(t1, n), t2);
    }
    void del(int x, int y) {
        int t1 = split(root, 1, x-1), t2 = split(root, y+1, node[root].size);
        root = merge(t1, t2);
    }
    void copy(int l, int r, int x) {
        int t1 = split(root, 1, x), t2 = split(root, l, r), t3 = split(root, x+1,
            node[root].size);
        root = merge(merge(t1, t2), t3);
    }
    void reverse(int x, int y) {
        int t1 = split(root, 1, x-1), t2 = split(root, x, y), t3 = split(root, y+1,
            node[root].size);
        node[t2].reverse ^= 1;
        root = merge(merge(t1, t2), t3);
    }
    char query(int k) {
        int x = root;
        while (1) {
            pushdown(x);
            if (k <= node[node[x].lc].size) x = node[x].lc;
            else
            if (k == node[node[x].lc].size + 1) return node[x].key;
            else
            k -= node[node[x].lc].size + 1, x = node[x].rc;
        }
    }
} treap;
```

3.7 可持久化并查集

接口：

void init() 初始化

void merge(int x, int y, int time) 在 time 时刻将 x 和 y 连一条边，注意加边顺序必须按 time 从小到大加边

void GetFather(int x, int time) 询问 time 时刻及以前的连边状态中，x 所属的集合

```
namespace pers_union {
    const int inf = 0x3f3f3f3f;
    int father[N], Father[N], Time[N];
    vector<int> e[N];
    void init() {
        for (int i=1;i<=n;i++) {
            father[i] = i;
            Father[i] = i;
            Time[i] = inf;
            e[i].clear();
            e[i].push_back(i);
        }
    }
    int getfather(int x) {
        return (father[x] == x) ? x : father[x] = getfather(father[x]);
    }
    int GetFather(int x, int time) {
        return (Time[x] <= time) ? GetFather(Father[x], time) : x;
    }
    void merge(int x, int y, int time) {
        int fx = getfather(x), fy = getfather(y);
        if (fx == fy) return;
        if (e[fx].size() > e[fy].size()) swap(fx, fy);
        father[fx] = fy;
        Father[fx] = fy;
        Time[fx] = time;
        for (int i=0;i<e[fx].size();i++) {
            e[fy].push_back(e[fx][i]);
        }
    }
};
```

4 树

4.1 树链剖分

接口：

void addedge(int x, int y);	将 x 到 y 连边，注意这是单向边
void dfs(int x, int root = 0);	从 x 开始遍历整棵树
void split(int x, int tp);	划分轻重链
int lca(int x, int y);	求 x 和 y 的 lca
int query(int x, int y);	求 x 到 y 经过的点数
int skip(int x, int k);	求从 x 向根方向跳 k 步到达的节点（若超出根，则返回 0）
void get_data(int x, int y);	将 x 到 y 路径上的重链找出来，存在 seg[0] 中

Debug 技巧：换一个根来 dfs 以测试程序是否能通过 $father[i] > i$ 的数据

```
struct EDGE {
    int adj, next;
} edge[N * 2];

int n, gh[N], top, s_top;
int father[N], deep[N], son[N], size[N], Top[N], dfn[N], rdfn[N];

void addedge(int x, int y) {
    edge[++top].adj = y;
    edge[top].next = gh[x];
    gh[x] = top;
}

void dfs(int x, int root = 0) {
    father[x] = root;
    deep[x] = deep[root] + 1;
    son[x] = 0;
    size[x] = 1;
    int dd = 0;
    for (int p = gh[x]; p; p = edge[p].next)
        if (edge[p].adj != root) {
            dfs(edge[p].adj, x);
            if (size[edge[p].adj] > dd) {
                dd = size[edge[p].adj];
                son[x] = edge[p].adj;
            }
            size[x] += size[edge[p].adj];
        }
}

void split(int x, int tp) {
    Top[x] = tp; dfn[x] = ++s_top; rdfn[s_top] = x;
    if (son[x]) split(son[x], tp);
    for (int p = gh[x]; p; p = edge[p].next)
        if (edge[p].adj != father[x] && edge[p].adj != son[x])
            split(edge[p].adj, edge[p].adj);
}

int lca(int x, int y) {
    int tx = Top[x], ty = Top[y];
```

```
    while (tx != ty) {
        if (deep[tx] < deep[ty]) {
            swap(tx, ty);
            swap(x, y);
        }
        x = father[tx];
        tx = Top[x];
    }
    if (deep[x] < deep[y])
        swap(x, y);
    return y;
}

int query(int x, int y) {
    int tx = Top[x], ty = Top[y];
    int ans = 0;
    while (tx != ty) {
        if (deep[tx] < deep[ty]) {
            swap(tx, ty);
            swap(x, y);
        }
        ans += dfn[x] - dfn[tx] + 1;
        x = father[tx];
        tx = Top[x];
    }
    if (deep[x] < deep[y])
        swap(x, y);
    ans += dfn[x] - dfn[y] + 1;
    return ans;
}

int skip(int x, int k) {
    int tx = Top[x];
    while (tx) {
        if (k < dfn[x] - dfn[tx] + 1) {
            return rdfn[ dfn[x] - k ];
        } else {
            k -= dfn[x] - dfn[tx] + 1;
            x = father[tx];
            tx = Top[x];
        }
    }
    return 0;
}

struct segment {
    int l, r;
    data d;
    segment(int _l, int _r) { // from _l to _r
        l = _l, r = _r;
```

```
        if (l <= r) d = query(l, r, 0);
        else d = query(r, l, 1); //reverse
    }
};

vector&lt;segment&gt; seg[2];

void get_data(int x, int y) {
    seg[0].clear(); seg[1].clear();
    int tx = Top[x], ty = Top[y];
    int s = 0;
    while (tx != ty) {
        if (deep[tx] < deep[ty]) {
            swap(tx, ty);
            swap(x, y);
            s ^= 1;
        }
        if (s == 0)
            seg[s].push_back(segment(w[x], w[tx]));
        else
            seg[s].push_back(segment(w[tx], w[x]));
        x = father[tx];
        tx = Top[x];
    }
    if (x != y) {
        if (deep[x] < deep[y]) {
            swap(x, y);
            s ^= 1;
        }
        if (s == 0)
            seg[s].push_back(segment(w[x], w[y] + 1));
        else
            seg[s].push_back(segment(w[y] + 1, w[x]));
    }
    reverse(seg[1].begin(), seg[1].end());
    for (int i = 0; i < seg[1].size(); ++i)
        seg[0].push_back(seg[1][i]);
    // saved to seg[0]
}

void init() {
    top = s_top = 0;
    for (int i = 1; i <= n; ++i) gh[i] = 0;
}
```

4.2 点分治

初始化时须设置 $top = 1$ 。

```
void addedge(int x, int y) {
```

```
    edge[++top].adj = y;
    edge[top].valid = 1;
    edge[top].next = gh[x];
    gh[x] = top;
}
void get_size(int x, int root=0) {
    size[x] = 1; son[x] = 0;
    int dd = 0;
    for (int p=gh[x]; p; p=edge[p].next)
        if (edge[p].adj != root && edge[p].valid) {
            get_size(edge[p].adj, x);
            size[x] += size[edge[p].adj];
            if (size[edge[p].adj] > dd) {
                dd = size[edge[p].adj];
                son[x] = edge[p].adj;
            }
        }
}
int getroot(int x) {
    get_size(x);
    int sz = size[x];
    while (size[son[x]] > sz/2)
        x = son[x];
    return x;
}
void dc(int x) {
    x = getroot(x);
    static int list[N], ltop;
    ltop = 0;
    for (int p=gh[x]; p; p=edge[p].next)
        if (edge[p].valid)
            list[++ltop] = edge[p].adj;
    clear();
    for (int i=1;i<=ltop;i++) {
        update();
        modify();
    }
    clear();
    for (int i=ltop;i>=1;i--) {
        update();
        modify();
    }
    //be careful about the root
    for (int p=gh[x]; p; p=edge[p].next)
        if (edge[p].valid) {
            edge[p].valid = 0;
            edge[p^1].valid = 0;
            dc(edge[p].adj);
        }
}
```

4.3 Link Cut Tree

请注意，一开始必须调用 lct.init(0) ，否则求出的最小值一定会是 0 。

```
struct DTree {
    int f[N], son[N][2], sz[N], rev[N], val[N], minid[N], minval[N];
    int tot;
    stack<int> s;
    void init(int i) {
        tot = max(tot, i);
        son[i][0] = son[i][1] = 0;
        f[i] = sz[i] = rev[i] = 0;
        val[i] = minval[i] = inf;
        minid[i] = i;
    }
    bool isroot(int x) {
        return !f[x] || (son[f[x]][0] != x && son[f[x]][1] != x);
    }
    void rev1(int x) {
        if (!x) return;
        swap(son[x][0], son[x][1]);
        rev[x] ^= 1;
    }
    void down(int x) {
        if (!x) return;
        if (rev[x]) rev1(son[x][0]), rev1(son[x][1]), rev[x] = 0;
    }
    void up(int x) {
        if (!x) return;
        down(son[x][0]); down(son[x][1]);
        sz[x] = sz[son[x][0]] + sz[son[x][1]] + 1;
        minval[x] = val[x]; minid[x] = x;
        if (minval[son[x][0]] < minval[x]) minval[x] = minval[son[x][0]], minid[x] = minid[son[x][0]];
        if (minval[son[x][1]] < minval[x]) minval[x] = minval[son[x][1]], minid[x] = minid[son[x][1]];
    }
    void rotate(int x) {
        int y = f[x], w = son[y][1] == x;
        son[y][w] = son[x][w ^ 1];
        if (son[x][w ^ 1]) f[son[x][w ^ 1]] = y;
        if (f[y]) {
            int z = f[y];
            if (son[z][0] == y) son[z][0] = x;
            else if (son[z][1] == y) son[z][1] = x;
        }
        f[x] = f[y]; f[y] = x; son[x][w ^ 1] = y;
        up(y);
```

```
    }
    void splay(int x) {
        while (!s.empty()) s.pop();
        s.push(x);
        for (int i = x; !isroot(i); i = f[i]) s.push(f[i]);
        while (!s.empty()) down(s.top()), s.pop();
        while (!isroot(x)) {
            int y = f[x];
            if (!isroot(y)) {
                if ((son[f[y]][0] == y) ^ (son[y][0] == x))
                    rotate(x);
                else
                    rotate(y);
            }
            rotate(x);
        }
        up(x);
    }
    void access(int x) {
        for (int y = 0; x; y = x, x = f[x]) {
            splay(x);
            son[x][1] = y;
            up(x);
        }
    }
    int root(int x) {
        access(x);
        splay(x);
        while (son[x][0]) x = son[x][0];
        return x;
    }
    void makeroot(int x) {
        access(x);
        splay(x);
        rev1(x);
    }
    void link(int x, int y) {
        makeroot(x);
        f[x] = y;
        access(x);
    }
    void cutf(int x) { // 它和父亲的边
        access(x);
        splay(x);
        f[son[x][0]] = 0;
        son[x][0] = 0;
        up(x);
    }
    void cut(int x, int y) { // 切断 x 与 y 之间的边(须保证 x 与 y 相邻)
        makeroot(x);
```

```
        cutf(y);
    }
    int ask(int x, int y) { // 询问 x 到 y 之间取得最小值的点
        makeroot(x);
        access(y);
        splay(y);
        return minid[y];
    }
    int querymin_cut(int x, int y) { // 询问 x 到 y 之间取得最小值的点，并把它删去(须保证该点在 x 和 y 之间，且度数恰好为 2)
        int m = ask(x, y);
        makeroot(x);
        cutf(m);
        makeroot(y);
        cutf(m);
        return val[m];
    }
    void link(int x, int y, int w) { // 在 x 和 y 之间添加一条权值为 w 的边(将边视为点插入)
        init(++tot);
        val[tot] = minval[tot] = w;
        link(x, tot);
        link(y, tot);
    }
} lct;
```

4.4 求子树的直径

树形 DP 。

答案保存在 u, d 数组中。

$u[x].exc$ 表示切断 x 与 $father[x]$ 的边，$father[x]$ 表示的那颗子树的直径。

$d[x].exc$ 表示切断 x 与 $father[x]$ 的边，x 表示的那颗子树的直径。

```
#include <bits/stdc++.h>

#define N 200020

using namespace std;

vector<int> g[N];
int n, q, top;
int deep[N], father[N], son[N], size[N], Top[N], dfn[N], rdfn[N];

void dfs(int x, int root = 0) {
    deep[x] = deep[root] + 1;
    father[x] = root;
    son[x] = 0; size[x] = 1;
    if (root) g[x].erase(lower_bound(g[x].begin(), g[x].end(), root));
    // 去根
```

```
    int dd = 0;
    for (int i = 0; i < g[x].size(); ++i) {
        dfs(g[x][i], x);
        if (size[g[x][i]] > dd) {
            dd = size[g[x][i]];
            son[x] = g[x][i];
        }
        size[x] += size[g[x][i]];
    }
}

void split(int x, int tp) {
    dfn[x] = ++top; rdfn[top] = x; Top[x] = tp;
    if (son[x]) split(son[x], tp);
    for (int i = 0; i < g[x].size(); ++i)
        if (g[x][i] != son[x])
            split(g[x][i], g[x][i]);
}

struct data {
    int inc, inc_id;
    int exc, exc_l, exc_r;
    //inc 表示从该点出发可以走到的最远距离
    //inc_id 表示从该点出发可以走到的最远点的编号
    //exc 表示子树中两点最远距离
    //exc_l, exc_r 表示子树中两点取得最远距离的两点的编号
    data() {
        inc = inc_id = 0;
        exc = exc_l = exc_r = 0;
    }
} u[N], d[N];

int safe(int x, int y) {
    // 防止 inc_id = 0 的情况
    if (x) return x;
    return y;
}

void dfs1(int x) {
    d[x].inc = 1; d[x].inc_id = x;
    data mx1 = data(), mx2 = data();
    // mx1, mx2 表示儿子 inc 最大、第2大值，用于更新该点 exc
    for (int i = 0; i < g[x].size(); ++i) {
        dfs1(g[x][i]);
        if (d[g[x][i]].inc + 1 > d[x].inc) {
            d[x].inc = d[g[x][i]].inc + 1;
            d[x].inc_id = d[g[x][i]].inc_id;
        }
        if (d[g[x][i]].inc > mx1.inc) {
            mx2 = mx1;
```

```
            mx1 = d[g[x][i]];
        } else
        if (d[g[x][i]].inc > mx2.inc) {
            mx2 = d[g[x][i]];
        }
    }
    d[x].exc = mx1.inc + mx2.inc + 1;
    d[x].exc_l = safe(mx1.inc_id, x);
    d[x].exc_r = safe(mx2.inc_id, x);
    for (int i = 0; i < g[x].size(); ++i)
        if (d[g[x][i]].exc > d[x].exc) {
            d[x].exc = d[g[x][i]].exc;
            d[x].exc_l = d[g[x][i]].exc_l;
            d[x].exc_r = d[g[x][i]].exc_r;
        }
}

void dfs2(int x, data y) {
    u[x] = y;
    if (!y.exc) y.exc = 1, y.exc_l = y.exc_r = x;
    data mx1 = y, mx2 = data(), mx3 = data(), mxe1 = y, mxe2 = data();
    // mx1, mx2, mx3 表示根过来的子树中 inc 的最大、第2大、第3大值
    // mxe1, mxe2 表示根过来的子树中 exc 的最大、第2大值
    int mx1_id = -1, mx2_id = -1, mx3_id = -1, mxe1_id = -1, mxe2_id = -1;
    for (int i = 0; i < g[x].size(); ++i) {
        if (d[g[x][i]].inc > mx1.inc) {
            mx3 = mx2; mx3_id = mx2_id;
            mx2 = mx1; mx2_id = mx1_id;
            mx1 = d[g[x][i]]; mx1_id = i;
        } else
        if (d[g[x][i]].inc > mx2.inc) {
            mx3 = mx2; mx3_id = mx2_id;
            mx2 = d[g[x][i]]; mx2_id = i;
        } else
        if (d[g[x][i]].inc > mx3.inc) {
            mx3 = d[g[x][i]]; mx3_id = i;
        }
        if (d[g[x][i]].exc > mxe1.exc) {
            mxe2 = mxe1; mxe2_id = mxe1_id;
            mxe1 = d[g[x][i]]; mxe1_id = i;
        } else
        if (d[g[x][i]].exc > mxe2.exc) {
            mxe2 = d[g[x][i]]; mxe2_id = i;
        }
    }
    for (int i = 0; i < g[x].size(); ++i) {
        data z = data();
        if (i == mx1_id) {
            z.exc = mx2.inc + mx3.inc + 1;
            z.exc_l = safe(mx2.inc_id, x);
```

```
            z.exc_r = safe(mx3.inc_id, x);
        } else
        if (i == mx2_id) {
            z.exc = mx1.inc + mx3.inc + 1;
            z.exc_l = safe(mx1.inc_id, x);
            z.exc_r = safe(mx3.inc_id, x);
        } else {
            z.exc = mx1.inc + mx2.inc + 1;
            z.exc_l = safe(mx1.inc_id, x);
            z.exc_r = safe(mx2.inc_id, x);
        }
        if (i == mxe1_id) {
            if (mxe2.exc > z.exc) z = mxe2;
        } else {
            if (mxe1.exc > z.exc) z = mxe1;
        }
        if (i == mx1_id) {
            z.inc = mx2.inc + 1;
            z.inc_id = safe(mx2.inc_id, x);
        } else {
            z.inc = mx1.inc + 1;
            z.inc_id = safe(mx1.inc_id, x);
        }
        dfs2(g[x][i], z);
    }
}
```

4.5 虚树

设 $a[0 \cdots k-1]$ 为需要构建虚树的点。

构建出虚树的节点保存在 a 数组中，k 为节点个数。加边调用函数 addedge(int x, int y, int w) 。

```
bool cmp(int x, int y) {
    return dfn[x] < dfn[y];
}

stack<int> stk;

void solve() {
    sort(a, a + k, cmp);
    int m = k;
    for (int j = 1; j < m; ++j)
        a[k++] = lca(a[j - 1], a[j]);
    sort(a, a + k, cmp);
    k = unique(a, a + k) - a;
    stk.push(a[0]);
    for (int j = 1; j < k; ++j) {
        int u = lca(stk.top(), a[j]);
        while (dep[stk.top()] > dep[u]) --top;
        assert(stk.top() == u);
```

```
        stk.push(a[j]);
        addedge(u, a[j], dis[a[j]] - dis[u]);
    }
}
```

5 图

5.1 欧拉回路

欧拉回路:

无向图：每个顶点的度数都是偶数，则存在欧拉回路。

有向图：每个顶点的入度 = 出度，则存在欧拉回路。

欧拉路径:

无向图：当且仅当该图所有顶点的度数为偶数，或者除了两个度数为奇数外其余的全是偶数。

有向图：当且仅当该图所有顶点出度 = 入度或者一个顶点出度 = 入度 + 1，另一个顶点入度 = 出度 + 1，其他顶点出度 = 入度。

下面 $O(n+m)$ 求欧拉回路的代码中，n 为点数，m 为边数，若有解则依次输出经过的边的编号, 若是无向图, 则正数表示 x 到 y ，负数表示 y 到 x 。

```
namespace UndirectedGraph{
    int n,m,i,x,y,d[N],g[N],v[M<<1],w[M<<1],vis[M<<1],nxt[M<<1],ed;
    int ans[M],cnt;
    void add(int x,int y,int z){
        d[x]++;
        v[++ed]=y;w[ed]=z;nxt[ed]=g[x];g[x]=ed;
    }
    void dfs(int x){
        for(int&i=g[x];i;){
            if(vis[i]){i=nxt[i];continue;}
            vis[i]=vis[i^1]=1;
            int j=w[i];
            dfs(v[i]);
            ans[++cnt]=j;
        }
    }
    void solve(){
        scanf("%d%d",&n,&m);
        for(i=ed=1;i<=m;i++)scanf("%d%d",&x,&y),add(x,y,i),add(y,x,-i);
        for(i=1;i<=n;i++)if(d[i]&1){puts("NO");return;}
        for(i=1;i<=n;i++)if(g[i]){dfs(i);break;}
        for(i=1;i<=n;i++)if(g[i]){puts("NO");return;}
        puts("YES");
        for(i=m;i;i--)printf("%d␣",ans[i]);
    }
}
namespace DirectedGraph{
    int n,m,i,x,y,d[N],g[N],v[M],vis[M],nxt[M],ed;
    int ans[M],cnt;
```

```
    void add(int x,int y){
        d[x]++;d[y]--;
        v[++ed]=y;nxt[ed]=g[x];g[x]=ed;
    }
    void dfs(int x){
        for(int&i=g[x];i;){
            if(vis[i]){i=nxt[i];continue;}
            vis[i]=1;
            int j=i;
            dfs(v[i]);
            ans[++cnt]=j;
        }
    }
    void solve(){
        scanf("%d%d",&n,&m);
        for(i=1;i<=m;i++)scanf("%d%d",&x,&y),add(x,y);
        for(i=1;i<=n;i++)if(d[i]){puts("NO");return;}
        for(i=1;i<=n;i++)if(g[i]){dfs(i);break;}
        for(i=1;i<=n;i++)if(g[i]){puts("NO");return;}
        puts("YES");
        for(i=m;i;i--)printf("%d ",ans[i]);
    }
}
```

5.2 最短路径

5.2.1 Dijkstra

```
#define LL long long

struct EDGE {
    int adj, w, next;
} edge[M*2];

typedef pair<LL, int> pli;
priority_queue <pli, vector<pli>, greater<pli> > q;

int n, top, gh[N];
LL dist[N];

void addedge(int x, int y, int w) {
    edge[++top].adj = y;
    edge[top].w = w;
    edge[top].next = gh[x];
    gh[x] = top;
}

LL dijkstra(int s, int t) {
    memset(dist, 63, sizeof(dist));
```

```
    memset(v, 0, sizeof(v));
    dist[s] = 0;
    q.push(make_pair(dist[s], s));
    while (!q.empty()) {
        LL dis = q.top().first;
        int x = q.top().second;
        q.pop();
        if (dis != dist[x]) continue;
        for (int p=gh[x]; p; p=edge[p].next) {
            if (dis + edge[p].w < dist[edge[p].adj]) {
                dist[edge[p].adj] = dis + edge[p].w;
                q.push(make_pair(dist[edge[p].adj], edge[p].adj));
            }
        }
    }
    return dist[t];
}
```

5.2.2 SPFA

```
struct EDGE {
    int adj, w, next;
} edge[M*2];

int n,m,top,gh[N],v[N],cnt[N],q[N],dist[N],head,tail;

void addedge(int x, int y, int w) {
    edge[++top].adj = y;
    edge[top].w = w;
    edge[top].next = gh[x];
    gh[x] = top;
}

int spfa(int S, int T) {
    memset(v, 0, sizeof(v));
    memset(cnt, 0, sizeof(cnt));
    memset(dist, 63, sizeof(dist));
    head = 0, tail = 1;
    dist[S] = 0; q[1] = S;
    while (head != tail) {
        (head += 1) %= N;
        int x = q[head]; v[x] = 0;
        ++cnt[x]; if (cnt[x] > n) return -1;
        for (int p=gh[x]; p; p=edge[p].next)
            if (dist[x] + edge[p].w < dist[edge[p].adj]) {
                dist[edge[p].adj] = dist[x] + edge[p].w;
                if (!v[edge[p].adj]) {
                    v[edge[p].adj] = 1;
                    (tail += 1) %= N;
```

```
                    q[tail] = edge[p].adj;
                }
            }
    }
    return dist[T];
}
```

5.3 K 短路

接口：

kthsp::init(n)： 初始化并设置节点个数为 n

kthsp::add(x, y, w)： 添加一条 x 到 y 的有向边

kthsp::work(S, T, k)： 求 S 到 T 的第 k 短路

```
#define N 200020
#define M 400020
#define LOGM 20
#define LL long long
#define inf (1LL<<61)

namespace pheap {
    struct Node {
        int next, son[2];
        LL val;
    } node[M*LOGM];
    int LOG[M];
    int root[M], size[M*LOGM], top;
    int add() {
        ++top; assert(top < M*LOGM);
        node[top].next = node[top].son[0] = node[top].son[1] = 0;
        node[top].val = inf;
        return top;
    }
    int copy(int x) {
        int t = add();
        node[t] = node[x];
        return t;
    }
    void init() {
        memset(root, 0, sizeof(root));
        top = -1; add();
        LOG[1] = 0;
        for (int i=2;i<M;i++) LOG[i] = LOG[i>>1] + 1;
    }
    void upd(int x, int &next, LL &val) {
        if (val < node[x].val) {
            swap(val, node[x].val);
            swap(next, node[x].next);
        }
```

```cpp
    }
    void insert(int x, int next, LL val) {
        int sz = size[root[x]] + 1;
        root[x] = copy(root[x]);
        size[root[x]] = sz; x = root[x];
        upd(x, next, val);
        for (int i=LOG[sz]-1;i>=0;i--) {
            int ind = (sz>>i)&1;
            node[x].son[ind] = copy(node[x].son[ind]);
            x = node[x].son[ind];
            upd(x, next, val);
        }
    }
};

namespace kthsp {
    using namespace pheap;
    struct EDGE {
        int adj, w, next;
    } edge[2][M];
    struct W {
        int x, y, w;
    } e[M];
    bool has_init = 0;
    int n, m, top[2], gh[2][N], v[N];
    LL dist[N];
    void init(int n1) {
        has_init = 1;
        n = n1; m = 0;
        memset(top, 0, sizeof(top));
        memset(gh, 0, sizeof(gh));
        for (int i=1;i<=n;i++) dist[i] = inf;
    }
    void addedge(int id, int x, int y, int w) {
        edge[id][++top[id]].adj = y;
        edge[id][top[id]].w = w;
        edge[id][top[id]].next = gh[id][x];
        gh[id][x] = top[id];
    }
    void add(int x, int y, int w) {
        assert(has_init);
        e[++m].x=x; e[m].y=y; e[m].w=w;
    }
    int best[N], bestw[N];
    typedef pair<LL, int> pli;
    priority_queue <pli, vector<pli>, greater<pli> > q;

    // you can replace dijkstra with SPFA or TOPSORT(DAG)
    void dijkstra(int S) {
        while (!q.empty()) q.pop();
```

```
        dist[S] = 0; q.push(make_pair(dist[S], S));
        while (!q.empty()) {
            LL dis = q.top().first;
            int x = q.top().second;
            q.pop();
            if (dist[x] != dis) continue;
            for (int p=gh[1][x]; p; p=edge[1][p].next) {
                int y = edge[1][p].adj;
                if (dist[x] + edge[1][p].w < dist[y]) {
                    dist[y] = dist[x] + edge[1][p].w;
                    best[y] = x;
                    bestw[y] = p;
                    q.push(make_pair(dist[y], y));
                }
            }
        }
    }
    void dfs(int x) {
        if (v[x]) return;
        v[x] = 1;
        if (best[x]) root[x] = root[best[x]];
        for (int p=gh[0][x]; p; p=edge[0][p].next)
            if (dist[edge[0][p].adj] != inf && bestw[x] != p) {
                insert(x, edge[0][p].adj, edge[0][p].w + dist[edge[0][p].adj] - dist
                    [x]);
            }
        for (int p=gh[1][x]; p; p=edge[1][p].next)
            if (best[edge[1][p].adj] == x)
                dfs(edge[1][p].adj);
    }
    LL work(int S, int T, int k) {
        assert(has_init);
        n++; add(T, n, 0);
        if (S == T) k ++;
        T = n;
        for (int i=1;i<=m;i++) {
            addedge(0, e[i].x, e[i].y, e[i].w);
            addedge(1, e[i].y, e[i].x, e[i].w);
        }
        dijkstra(T);
        root[T] = 0; pheap::init();
        memset(v, 0, sizeof(v));
        dfs(T);
        while (!q.empty()) q.pop();
        if (k == 1) return dist[S];
        if (root[S]) q.push(make_pair(dist[S] + node[root[S]].val, root[S]));
        while (k--) {
            if (q.empty()) return inf;
            pli now = q.top(); q.pop();
            if (k == 1) return now.first;
```

```
            int x = node[now.second].next, u = node[now.second].son[0], v = node[now
                .second].son[1];
            if (root[x]) q.push(make_pair(now.first + node[root[x]].val, root[x]));
            if (u) q.push(make_pair(now.first - node[now.second].val + node[u].val,
                u));
            if (v) q.push(make_pair(now.first - node[now.second].val + node[v].val,
                v));
        }
        return 0;
    }
};
```

5.4 Tarjan

割点的判断：一个顶点 u 是割点，当且仅当满足 (1) 或 (2)：
(1) u 为树根，且 u 有多于一个子树（即：存在一个儿子 v 使得 $dfn[u]+1 \neq dfn[v]$ ）
(2) u 不为树根，且满足存在 (u,v) 为树枝边（u 为 v 的父亲），使得 $dfn[u] \leq low[v]$
桥的判断：一条无向边 (u,v) 是桥，当且仅当 (u,v) 为树枝边，满足 $dfn[u] < low[v]$

```
struct EDGE { int adj, next; } edge[M];
int n, m, top, gh[N];
int dfn[N], low[N], cnt, ind, stop, instack[N], stack[N], belong[N];
void addedge(int x, int y) {
    edge[++top].adj = y;
    edge[top].next = gh[x];
    gh[x] = top;
}
void tarjan(int x) {
    dfn[x] = low[x] = ++ind;
    instack[x] = 1; stack[++stop] = x;
    for (int p=gh[x]; p; p=edge[p].next)
        if (!dfn[edge[p].adj]) {
            tarjan(edge[p].adj);
            low[x] = min(low[x], low[edge[p].adj]);
        } else if (instack[edge[p].adj]) {
            low[x] = min(low[x], dfn[edge[p].adj]);
        }
    if (dfn[x] == low[x]) {
        ++cnt; int tmp=0;
        while (tmp!=x) {
            tmp = stack[stop--];
            belong[tmp] = cnt;
            instack[tmp] = 0;
        }
    }
}
```

5.5 2-SAT

```cpp
#define N number_of_vertex
#define M number_of_edges

struct MergePoint {
    struct EDGE {
        int adj, next;
    } edge[M];
    int ex[M], ey[M];
    bool instack[N];
    int gh[N], top, dfn[N], low[N], cnt, ind, stop, stack[N], belong[N];
    void init() {
        cnt = ind = stop = top = 0;
        memset(dfn, 0, sizeof(dfn));
        memset(instack, 0, sizeof(instack));
        memset(gh, 0, sizeof(gh));
    }
    void addedge(int x, int y) { //reverse
        std::swap(x, y);
        edge[++top].adj = y;
        edge[top].next = gh[x];
        gh[x] = top;
        ex[top] = x;
        ey[top] = y;
    }
    void tarjan(int x) {
        dfn[x] = low[x] = ++ind;
        instack[x] = 1; stack[++stop] = x;
        for (int p=gh[x]; p; p=edge[p].next)
            if (!dfn[edge[p].adj]) {
                tarjan(edge[p].adj);
                low[x] = std::min(low[x], low[edge[p].adj]);
            } else if (instack[edge[p].adj]) {
                low[x] = std::min(low[x], dfn[edge[p].adj]);
            }
        if (dfn[x] == low[x]) {
            ++cnt; int tmp = 0;
            while (tmp!=x) {
                tmp = stack[stop--];
                belong[tmp] = cnt;
                instack[tmp] = 0;
            }
        }
    }
    void work() {
        for (int i = (__first__); i <= (__last__); ++i)
            if (!dfn[i])
                tarjan(i);
    }
} merge;
```

```cpp

struct Topsort {
    struct EDGE {
        int adj, next;
    } edge[M];
    int n, top, gh[N], ops[N], deg[N], ans[N];
    std::queue<int> q;
    void init() {
        n = merge.cnt; top = 0;
        memset(gh, 0, sizeof(gh));
        memset(deg, 0, sizeof(deg));
    }
    void addedge(int x, int y) {
        if (x == y) return;
        edge[++top].adj = y;
        edge[top].next = gh[x];
        gh[x] = top;
        ++deg[y];
    }
    void work() {
        for (int i = 1; i <= n; ++i)
            if (!deg[i])
                q.push(i);
        while (!q.empty()) {
            int x = q.front();
            q.pop();
            for (int p = gh[x]; p; p = edge[p].next)
                if (!--deg[edge[p].adj])
                    q.push(edge[p].adj);
            if (ans[x]) continue;
            ans[x] = -1; //not selected
            ans[ops[x]] = 1; //selected
        }
    }
} ts;
```

调用示例:

```cpp
    merge.init();
    merge.addedge();
    merge.work();
    for (int i = 1; i <= n; ++i) {
        if (merge.belong[U(i, 0)] == merge.belong[U(i, 1)]) {
            puts("NO");
            return 0;
        }
        ts.ops[merge.belong[U(i, 0)]] = merge.belong[U(i, 1)];
        ts.ops[merge.belong[U(i, 1)]] = merge.belong[U(i, 0)];
    }
    ts.init();
    ts.work();
```

```
    puts("YES");
    for (int i = 1; i <= n; ++i) {
        int x = U(i, 0), y = U(i, 1);
        x = merge.belong[x], y = merge.belong[y];
        x = ts.ans[x], y = ts.ans[y];
        if (x == 1) puts("0␣is␣selected");
        if (y == 1) puts("1␣is␣selected");
    }
```

5.6 统治者树（Dominator Tree）

Dominator Tree 可以解决判断一类有向图必经点的问题。

idom[x] 表示离 x 最近的必经点（重编号后）。将 idom[x] 作为 x 的父亲，构成一棵 Dominator Tree

接口：

void dominator::init(int n); 初始化，有向图节点数为 n

void dominator::addedge(int u, int v); 添加一条有向边 (u, v)

void dominator::work(int root); 以 root 为根，建立一棵 Dominator Tree

结果的返回：

在执行 dominator::work(int root); 后，树边保存在 vector <int> tree[N] 中

```
namespace dominator {
    vector <int> g[N], rg[N], bucket[N], tree[N];
    int n, ind, idom[N], sdom[N], dfn[N], dsu[N], father[N], label[N], rev[N];
    void dfs(int x) {
        ++ind;
        dfn[x] = ind; rev[ind] = x;
        label[ind] = dsu[ind] = sdom[ind] = ind;
        for (auto p : g[x]) {
            if (!dfn[p]) dfs(p), father[dfn[p]] = dfn[x];
            rg[dfn[p]].push_back(dfn[x]);
        }
    }
    void init(int n1) {
        n = n1; ind = 0;
        for (int i = 1; i <= n; ++i) {
            g[i].clear();
            rg[i].clear();
            bucket[i].clear();
            tree[i].clear();
            dfn[i] = 0;
        }
    }
    void addedge(int u, int v) {
        g[u].push_back(v);
    }
    int find(int x, int step=0) {
        if (dsu[x] == x) return step ? -1 : x;
```

```cpp
        int y = find(dsu[x], 1);
        if (y < 0) return x;
        if (sdom[label[dsu[x]]] < sdom[label[x]])
            label[x] = label[dsu[x]];
        dsu[x] = y;
        return step ? dsu[x] : label[x];
    }
    void work(int root) {
        dfs(root); n = ind;
        for (int i = n; i; --i) {
            for (auto p : rg[i])
                sdom[i] = min(sdom[i], sdom[find(p)]);
            if (i > 1) bucket[sdom[i]].push_back(i);
            for (auto p : bucket[i]) {
                int u = find(p);
                if (sdom[p] == sdom[u]) idom[p] = sdom[p];
                else idom[p] = u;
            }
            if (i > 1) dsu[i] = father[i];
        }
        for (int i = 2; i <= n; ++i) {
            if (idom[i] != sdom[i])
                idom[i] = idom[idom[i]];
            tree[rev[i]].push_back(rev[idom[i]]);
            tree[rev[idom[i]]].push_back(rev[i]);
        }
    }
};
```

5.7 网络流

5.7.1 最大流

注意：*top 要初始化为 1*

```cpp
struct EDGE { int adj, w, next; } edge[M];
int n, top, gh[N], nrl[N];
void addedge(int x, int y, int w) {
    edge[++top].adj = y;
    edge[top].w = w;
    edge[top].next = gh[x];
    gh[x] = top;
    edge[++top].adj = x;
    edge[top].w = 0;
    edge[top].next = gh[y];
    gh[y] = top;
}
int dist[N], q[N];
int bfs() {
    memset(dist, 0, sizeof(dist));
```

```
    q[1] = S; int head = 0, tail = 1; dist[S] = 1;
    while (head != tail) {
        int x = q[++head];
        for (int p=gh[x]; p; p=edge[p].next)
            if (edge[p].w && !dist[edge[p].adj]) {
                dist[edge[p].adj] = dist[x] + 1;
                q[++tail] = edge[p].adj;
            }
    }
    return dist[T];
}
int dinic(int x, int delta) {
    if (x==T) return delta;
    for (int& p=nrl[x]; p && delta; p=edge[p].next)
        if (edge[p].w && dist[x]+1 == dist[edge[p].adj]) {
            int dd = dinic(edge[p].adj, min(delta, edge[p].w));
            if (!dd) continue;
            edge[p].w -= dd;
            edge[p^1].w += dd;
            return dd;
        }
    return 0;
}
int work() {
    int ans = 0;
    while (bfs()) {
        memcpy(nrl, gh, sizeof(gh));
        int t; while (t = dinic(S, inf)) ans += t;
    }
    return ans;
}
```

5.7.2 上下界有源汇网络流

T 向 S 连容量为正无穷的边，将有源汇转化为无源汇。

每条边容量减去下界，设 $in[i]$ 表示流入 i 的下界之和减去流出 i 的下界之和。

新建超级源汇 SS,TT ，对于 $in[i] > 0$ 的点，SS 向 i 连容量为 $in[i]$ 的边。对于 $in[i] < 0$ 的点，i 向 TT 连容量为 $-in[i]$ 的边。

求出以 SS,TT 为源汇的最大流，如果等于 $\Sigma in[i](in[i] > 0)$ ，则存在可行流。再求出 S,T 为源汇的最大流即为最大流。

费用流：建完图后等价于求以 SS,TT 为源汇的费用流。

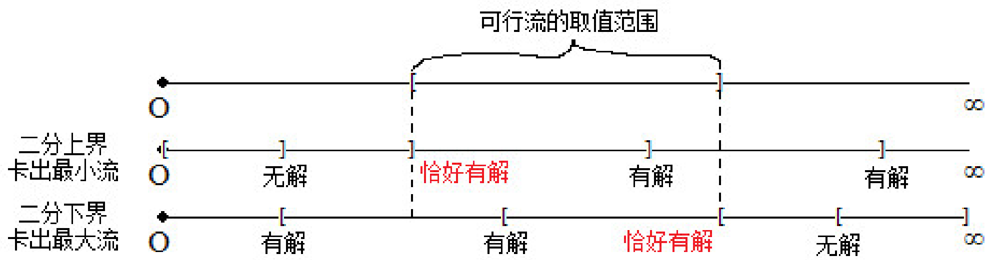
5.7.3 上下界无源汇网络流

1. 怎样求无源汇有上下界网络的可行流?

由于有源汇的网络我们先要转化成无源汇，所以本来就无源汇的网络不用再作特殊处理。

2. 怎样求无源汇有上下界网络的最大流、最小流?

一种简易的方法是采用二分的思想，不断通过可行流的存在与否对 (t,s) 边的上下界 U,L 进行调整。求最大流时令 $U=\infty$ 并二分 L ；求最小流时令 $L=0$ 并二分 U 。道理很简单，因为可行流的取值范围是一段连续的区间，我们只要通过二分找到有解和无解的分界线即可。



5.7.4 费用流

注意：*top 要初始化为 1*

```
#define inf 0x3f3f3f3f
struct NetWorkFlow {
    struct EDGE {
        int adj, w, cost, next;
    } edge[M*2];
    int gh[N], q[N], dist[N], v[N], pre[N], prev[N], top;
    int S, T;
    void addedge(int x, int y, int w, int cost) {
        edge[++top].adj = y;
        edge[top].w = w;
        edge[top].cost = cost;
        edge[top].next = gh[x];
        gh[x] = top;
        edge[++top].adj = x;
        edge[top].w = 0;
        edge[top].cost = -cost;
        edge[top].next = gh[y];
        gh[y] = top;
    }
    void clear() {
        top = 1;
        memset(gh, 0, sizeof(gh));
    }
    int spfa() {
        memset(dist, 63, sizeof(dist));
        memset(v, 0, sizeof(v));
        int head = 0, tail = 1;
        q[1] = S; v[S] = 1; dist[S] = 0;
        while (head != tail) {
            (head += 1) %= N;
            int x = q[head];
```

```
            v[x] = 0;
            for (int p=gh[x]; p; p=edge[p].next)
                if (edge[p].w && dist[x] + edge[p].cost < dist[edge[p].adj]) {
                    dist[edge[p].adj] = dist[x] + edge[p].cost;
                    pre[edge[p].adj] = x;
                    prev[edge[p].adj] = p;
                    if (!v[edge[p].adj]) {
                        v[edge[p].adj] = 1;
                        (tail += 1) %= N;
                        q[tail] = edge[p].adj;
                    }
                }
        }
        return dist[T] != inf;
    }
    int work() {
        int ans = 0;
        while (spfa()) {
            int mx = inf;
            for (int x=T;x!=S;x=pre[x])
                mx = min(edge[prev[x]].w, mx);
            ans += dist[T] * mx;
            for (int x=T;x!=S;x=pre[x]) {
                edge[prev[x]].w -= mx;
                edge[prev[x]^1].w += mx;
            }
        }
        return ans;
    }
} nwf;
```

5.7.5 zkw 费用流

注意：*top* 要初始化为 *1*，不得用于有负权的图

```
#define inf 0x3f3f3f3f //modify if you use long long or double
template <class _tp>
struct NetWorkFlow {
    struct EDGE {
        int adj, next;
        _tp w, cost;
    } edge[M*2];
    int gh[N], top;
    int S, T;
    void addedge(int x, int y, _tp w, _tp cost) {
        edge[++top].adj = y;
        edge[top].w = w;
        edge[top].cost = cost;
        edge[top].next = gh[x];
        gh[x] = top;
```

```
        edge[++top].adj = x;
        edge[top].w = 0;
        edge[top].cost = -cost;
        edge[top].next = gh[y];
        gh[y] = top;
    }
    void clear() {
        top = 1;
        memset(gh, 0, sizeof(gh));
    }
    int v[N];
    _tp cost, d[N], slk[N];
    _tp aug(int x, _tp f) {
        _tp left = f;
        if (x == T) {
            cost += f * d[S];
            return f;
        }
        v[x] = true;
        for (int p=gh[x]; p; p=edge[p].next)
            if (edge[p].w && !v[edge[p].adj]) {
                _tp t = d[edge[p].adj] + edge[p].cost - d[x];
                if (t == 0) {
                    _tp delt = aug(edge[p].adj, min(left, edge[p].w));
                    if (delt > 0) {
                        edge[p].w -= delt;
                        edge[p^1].w += delt;
                        left -= delt;
                    }
                    if (left == 0) return f;
                } else {
                if (t < slk[edge[p].adj])
                    slk[edge[p].adj] = t;
                }
            }
        return f-left;
    }
    bool modlabel() {
        _tp delt = inf;
        for (int i=1;i<=T;i++)
            if (!v[i]) {
                if (slk[i] < delt) delt = slk[i];
                slk[i] = inf;
            }
        if (delt == inf) return true;
        for (int i=1;i<=T;i++)
            if (v[i]) d[i] += delt;
        return false;
    }
    _tp work() {
```

```
        cost = 0;
        memset(d, 0, sizeof(d));
        memset(slk, 63, sizeof(slk));
        do {
            do {
                memset(v, 0, sizeof(v));
            } while (aug(S, inf));
        } while (!modlabel());
        return cost;
    }
};
NetWorkFlow<int> nwf;
```

6 数学

6.1 扩展欧几里得解同余方程

ans[] 保存的是循环节内所有的解

```
int exgcd(int a,int b,int&x,int&y){
    if(!b)return x=1,y=0,a;
    int d=exgcd(b,a%b,x,y),t=x;
    return x=y,y=t-a/b*y,d;
}
void cal(ll a,ll b,ll n){//ax=b(mod n)
    ll x,y,d=exgcd(a,n,x,y);
    if(b%d)return;
    x=(x%n+n)%n;
    ans[cnt=1]=x*(b/d)%(n/d);
    for(ll i=1;i<d;i++)ans[++cnt]=(ans[1]+i*n/d)%n;
}
```

6.2 同余方程组

```
int n,flag,k,m,a,r,d,x,y;
int main(){
    scanf("%d",&n);
    flag=k=1,m=0;
    while(n--){
        scanf("%d%d",&a,&r);//ans%a=r
        if(flag){
            d=exgcd(k,a,x,y);
            if((r-m)%d){flag=0;continue;}
            x=(x*((r-m)/d)+a/d)%(a/d),y=k/d*a,m=((x*k+m)%y)%y;
            if(m<0)m+=y;
            k=y;
        }
    }
```

```
    printf("%d",flag?m:-1);//若flag=1,说明有解,解为ki+m,i为任意整数
}
```

6.3 类欧几里得算法

类欧几里得模板有三种形式:

$$f(a,b,c,n)=\sum_{i=0}^{n}\lfloor\frac{ai+b}{c}\rfloor$$

$$g(a,b,c,n)=\sum_{i=0}^{n}i\lfloor\frac{ai+b}{c}\rfloor$$

$$h(a,b,c,n)=\sum_{i=0}^{n}\lfloor\frac{ai+b}{c}\rfloor^2$$

```
#define LL long long

const int P = 1000000007;
int Inv(int x) {
    return x == 1 ? 1 : 1ll * (P - P / x) * Inv(P % x) % P;
}
const int i2 = Inv(2);
const int i6 = Inv(6);

struct ifo {
    int f, g, h;
    ifo(int f, int g, int h) : f(f), g(g), h(h) {}
};

int S1(int n) {
    return 1ll * n * (n + 1) % P * i2 % P;
}

int S2(int n) {
    return 1ll * n * (n + 1) % P * (2 * n + 1) % P * i6 % P;
}

ifo Get(int n, int A, int B, int C) {
    if (!A) {
        int t = B / C;
        int f = 1ll * (n + 1) * t % P;
        int g = 1ll * S1(n) * t % P;
        int h = 1ll * (n + 1) * t % P * t % P;
        return ifo(f, g, h);
    } else if (A >= C || B >= C) {
        ifo Nx = Get(n, A % C, B % C, C);
        int p = A / C, q = B / C;
        int f = (1ll * p * S1(n) + 1ll * q * (n + 1) + Nx.f) % P;
```

```
        int g = (1ll * p * S2(n) + 1ll * q * S1(n) + Nx.g) % P;
        int h = (1ll * p * p % P * S2(n) + 2ll * p * q % P * S1(n) + 1ll * (n + 1) *
            q % P * q + 2ll * p * Nx.g % P + 2ll * q * Nx.f % P + Nx.h) % P;
        return ifo(f, g, h);
    } else {
        int m = (1ll * A * n + B) / C;
        ifo Nx = Get(m - 1, C, C - B - 1, A);
        int f = (1ll * n * m - Nx.f) % P;
        int g = (1ll * m * S1(n) - 1ll * i2 * Nx.h - 1ll * i2 * Nx.f) % P;
        int h = (2ll * n * S1(m - 1) % P + 1ll * n * m - 2ll * Nx.g - Nx.f) % P;
        return ifo(f, g, h);
    }
}
```

6.4 卡特兰数

$h_1 = 1, h_n = \frac{h_{n-1}(4n-2)}{n+1} = \frac{C(2n,n)}{n+1} = C(2n,n) - C(2n,n-1)$

在一个格点阵列中，从 $(0,0)$ 点走到 (n,m) 点且不经过对角线 $x = y$ 的方案数 $(x > y)$：

$C(n+m-1,m) - C(n+m-1,m-1)$

在一个格点阵列中，从 $(0,0)$ 点走到 (n,m) 点且不穿过对角线 $x = y$ 的方案数 $(x \geq y)$：

$C(n+m,m) - C(n+m,m-1)$

6.5 斯特林数

6.5.1 第一类斯特林数

第一类 Stirling 数 $S(p,k)$ 的一个组合学解释是：将 p 个物体排成 k 个非空循环排列的方法数。

$S(p,k)$ 的递推公式：$S(p,k) = (p-1)S(p-1,k) + S(p-1,k-1), 1 \leq k \leq p-1$

边界条件：$S(p,0) = 0, p \geq 1$ $S(p,p) = 1, p \geq 0$

6.5.2 第二类斯特林数

第二类 Stirling 数 $S(p,k)$ 的一个组合学解释是：将 p 个物体划分成 k 个非空的不可辨别（可以理解为盒子没有编号）集合的方法数。

$S(p,k)$ 的递推公式：$S(p,k) = kS(p-1,k) + S(p-1,k-1), 1 \leq k \leq p-1$

边界条件：$S(p,0) = 0, p \geq 1$ $S(p,p) = 1, p \geq 0$

也有卷积形式：

$$S(n,m) = \frac{1}{m!}\sum_{k=0}^{m}(-1)^k C(m,k)(m-k)^n = \sum_{k=0}^{m}\frac{(-1)^k(m-k)^n}{k!(m-k)!} = \sum_{k=0}^{m}\frac{(-1)^k}{k!} \times \frac{(m-k)^n}{(m-k)!}$$

6.6 错排公式

$$D_1 = 0, D_2 = 1, D_n = (n-1)(D_{n-2} + D_{n-1})$$

6.7 Lucas 定理

接口：

初始化：void lucas::init();

计算 $C(n,m)\%mod$ 的值：LL lucas::Lucas(LL n, LL m);

```
#define mod 110119
#define LL long long
namespace lucas {
    LL fac[mod+1], facv[mod+1];
    LL power(LL base, LL times) {
        LL ans = 1;
        while (times) {
            if (times&1) (ans *= base) %= mod;
            (base *= base) %= mod;
            times >>= 1;
        }
        return ans;
    }
    void init() {
        fac[0] = 1; for (int i=1;i<mod;i++) fac[i] = (fac[i-1] * i) % mod;
        facv[mod-1] = power(fac[mod-1], mod-2);
        for (int i=mod-2;i>=0;--i) facv[i] = (facv[i+1] * (i+1)) % mod;
    }
    LL C(unsigned LL n, unsigned LL m) {
        if (n < m) return 0;
        return (fac[n] * facv[m] % mod * facv[n-m] % mod) % mod;
    }
    LL Lucas(unsigned LL n, unsigned LL m)
    {
        if (m == 0) return 1;
        return (C(n%mod, m%mod) * Lucas(n/mod, m/mod)) %mod;
    }
};
```

6.8 线性规划

6.8.1 单纯形法

单纯形法用于解决线性规划问题：

$$\max_{x_1,x_2,\cdots,x_n} x_0 = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

$$s.t. \begin{cases} A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{in}x_n \le b_i, & i = 1,2,\cdots,m \\ x_j \ge 0, & j = 1,2,\cdots,n \end{cases}$$

小心：单纯形法通常能解决 $n \le 500, m \le 500$ 的数据规模的问题。若规模过大，可能导致精度爆炸。

小心：单纯形法只能解决一般线性规划问题，不能解决整数规划问题（NP Hard）。若要用单纯形法解决整数规划问题，必须先证明一般线性规划的解不比整数规划好。

若 $b_i \geq 0, i = 1, 2, \cdots, n$ ，则不需要执行 init ，因为至少有一组解 $x_1 = x_2 = \cdots = x_n = 0$ 。

输入格式 .

$$\begin{array}{llllll} n & m & & & & \\ c_1 & c_2 & c_3 & \cdots & c_n & \\ A_{11} & A_{12} & A_{13} & \cdots & A_{1n} & b_1 \\ A_{21} & A_{22} & A_{23} & \cdots & A_{2n} & b_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ A_{m1} & A_{m2} & A_{m3} & \cdots & A_{mn} & b_m \end{array}$$

输出格式 .

若无解，输出 Infeasible 。

若 x_0 无界，输出 Unbounded 。

第一行输出答案 x_0 。

接下来一行输出 n 个实数表示 $x_1, x_2, \cdots, x_n$ 。

```
#include <bits/stdc++.h>

#define N 25
#define M 25

using namespace std;

const double eps = 1e-8, INF = 1e15;

int n, m;
double a[M][N], ans[N + M];
int id[N + M];

void pivot(int l, int e) {
    swap(id[n + l], id[e]);
    double t = a[l][e];
    a[l][e] = 1;
    for (int j = 0; j <= n; ++j) a[l][j] /= t;
    for (int i = 0; i <= m; ++i)
        if (i != l && abs(a[i][e]) > eps) {
            t = a[i][e];
            a[i][e] = 0;
            for (int j = 0; j <= n; ++j) a[i][j] -= a[l][j] * t;
        }
}

bool init() {
    while (1) {
        int e = 0, l = 0;
```

```
        for (int i = 1; i <= m; ++i)
            if (a[i][0] < -eps && (!l || (rand() & 1)))
                l = i;
        if (!l) break;
        for (int j = 1; j <= n; ++j)
            if (a[l][j] < -eps && (!e || (rand() & 1)))
                e = j;
        if (!e) return false; // Infeasible
        pivot(l, e);
    }
    return true;
}

bool simplex() {
    while (1) {
        int l = 0, e = 0;
        double mn = INF;
        for (int j = 1; j <= n; ++j)
            if (a[0][j] > eps) {
                e = j;
                break;
            }
        if (!e) break;
        for (int i = 1; i <= m; ++i)
            if (a[i][e] > eps && a[i][0] / a[i][e] < mn) {
                mn = a[i][0] / a[i][e];
                l = i;
            }
        if (!l) return false; // Unbounded
        pivot(l, e);
    }
    return true;
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i) scanf("%lf", &a[0][i]);
    for (int i = 1; i <= m; ++i) {
        for (int j = 1; j <= n; ++j) scanf("%lf", &a[i][j]);
        scanf("%lf", &a[i][0]);
    }
    for (int i = 0; i <= n + m; ++i) id[i] = 0;
    for (int i = 1; i <= n; ++i) id[i] = i;
    if (!init()) {
        puts("Infeasible");
        return 0;
    }
    if (!simplex()) {
        puts("Unbounded");
        return 0;
```

```
    }
    printf("%.10lf\n", -a[0][0]);
    for (int i = 0; i <= n + m; ++i) ans[i] = 0;
    for (int i = 1; i <= m; ++i) ans[id[n + i]] = a[i][0];
    for (int i = 1; i <= n; ++i) printf("%.10lf␣", ans[i]);
    puts("");
}
```

6.8.2 对偶理论

原始问题:

$$\max_{x_1,x_2,\cdots,x_n} x_0 = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

$$s.t. \begin{cases} A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{in}x_n \le b_i, & i = 1,2,\cdots,m \\ x_j \ge 0, & j = 1,2,\cdots,n \end{cases}$$

对偶问题:

$$\min_{w_1,w_2,\cdots,w_m} w_0 = b_1x_1 + b_2x_2 + \cdots + b_mx_m$$

$$s.t. \begin{cases} A_{i1}^T w_1 + A_{i2}^T w_2 + \cdots A_{im}^T w_m \ge c_i, & i = 1,2,\cdots,n \\ w_j \ge 0, & j = 1,2,\cdots,m \end{cases}$$

6.9 高斯消元

6.9.1 行列式

```
int ans = 1;
for (int i=0;i<n;i++) {
    for (int j=i;j<n;j++)
        if (g[j][i]) {
            for (int k=i;k<n;k++)
                swap(g[i][k], g[j][k]);
            if (j != i) ans *= -1;
            break;
        }
    if (g[i][i] == 0) {
        ans = 0;
        break;
    }
    for (int j=i+1;j<n;j++) {
        while (g[j][i]) {
            int t = g[i][i] / g[j][i];
            for (int k=i;k<n;k++)
                g[i][k] = (g[i][k] + mod - ((LL)t * g[j][k] % mod)) % mod;
            for (int k=i;k<n;k++)
                swap(g[i][k], g[j][k]);
```

```
            ans *= -1;
        }
    }
}
for (int i=0;i<n;i++)
    ans = ((LL)ans * g[i][i]) % mod;
ans = (ans % mod + mod) % mod;
printf("%d\n", ans);
```

6.9.2 Matrix-Tree 定理

对于一张图，建立矩阵 C ，$C[i][i] =$ i 的度数，若 i, j 之间有边，那么 $C[i][j] = -1$ ，否则为 0 。这张图的生成树个数等于矩阵 C 的 $n - 1$ 阶行列式的值。

6.10 调和级数

$\sum_{i=1}^{n} \frac{1}{i}$ 在 n 较大时约等于 $ln(n) + r$ ，r 为欧拉常数，约等于 0.5772156649015328 。

6.11 曼哈顿距离的变换

$|x_1 - x_2| + |y_1 - y_2| = max(|(x_1 + y_1) - (x_2 + y_2)|, |(x_1 - y_1) - (x_2 - y_2)|)$

6.12 数论函数变换

常见积性函数:
欧拉函数 $\phi(n)$ 为不超过 n 的与 n 互质的正整数个数
莫比乌斯函数 $\mu(n) = \begin{cases} 1, & 若n = 1 \\ (-1)^k, & 若n无平方数因数，且n = p_1 p_2 \cdots p_k \\ 0, & 若n有大于 1 的平方数因数 \end{cases}$
常见积性函数的性质:

$$n = \sum_{d|n} \phi(d)$$

$$\sum_{d|n} \mu(d) = \begin{cases} 1, & n = 1 \\ 0, & n > 1 \end{cases}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m} i \times j[gcd(i,j) = d] = \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} \sum_{j=1}^{\lfloor \frac{m}{d} \rfloor} id \times jd[gcd(i,j) = 1]$$

$$\phi(n) = \sum_{d|n} \mu(d) \frac{n}{d}$$

6.13 莫比乌斯反演

$F(n)$ 和 $f(n)$ 是定义在非负整数集合上的两个函数，则:

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

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

6.14 线性筛素数

```
mu[1]=phi[1]=1;top=0;
for (int i=2;i<N;i++) {
    if (!v[i]) prime[++top]=i, mu[i] = -1, phi[i] = i-1;
    for (int j=1;i*prime[j]<N && j<=top;j++) {
        v[i*prime[j]] = 1;
        if (i%prime[j]) {
            mu[i*prime[j]] = -mu[i];
            phi[i*prime[j]] = phi[i] * (prime[j]-1);
        } else {
            mu[i*prime[j]] = 0;
            phi[i*prime[j]] = phi[i] * prime[j];
            break;
        }
    }
}
```

6.15 杜教筛

getphi(t, x) 表示求 $\sum\limits_{i=1}^{x}i^t\phi(i)$ 。

推导过程:

记 $S(n)=\sum\limits_{i=1}^{n}f(i)$ ，取任意函数 g 有恒等式

$$S(n)=\sum_{i=1}^{n}(f\cdot g)(i)-\sum_{i=2}^{n}g(i)S(\lfloor\frac{n}{i}\rfloor)$$

其中，$(f\cdot g)$ 表示 f 和 g 的狄利克雷卷积：即：$(f\cdot g)(n)=\sum\limits_{d|n}f(d)g(\frac{n}{d})$

关于恒等式的证明:

将 $\sum\limits_{i=2}^{n}g(i)S(\lfloor\frac{n}{i}\rfloor)$ 移到左边去，即只需证

$$\sum_{i=1}^{n}(f\cdot g)(i)=\sum_{i=1}^{n}g(i)S(\lfloor\frac{n}{i}\rfloor)$$

将狄利克雷卷积展开，得:

$$\sum_{i=1}^{n}\sum_{d|i} g(d)f(\frac{i}{d}) = \sum_{i=1}^{n} g(i)S(\lfloor\frac{n}{i}\rfloor)$$

即：

$$\sum_{d=1}^{n} g(d)\sum_{i=1}^{\lfloor\frac{n}{d}\rfloor} f(i) = \sum_{i=1}^{n} g(i)S(\lfloor\frac{n}{i}\rfloor)$$

显然相等，恒等式证完。

取 $f(i) = i^p\phi(i), g(i) = i^p$ ，则有：

$$S(n) = \sum_{i=1}^{n} i^p\phi(i) = \sum_{i=1}^{n} i^{p+1} - \sum_{i=2}^{n} i^p S(\lfloor\frac{n}{i}\rfloor)$$

其中有用到等式 $\sum\limits_{d|n}\phi(d) = n$

另外，莫比乌斯函数 $\mu(n)$ 也可以使用杜教筛求前缀和，记 $S'(n) = \sum\limits_{i=1}^{n}\mu(i)$ ，则 $S'(n) = 1 - \sum\limits_{i=2}^{n} S'(\lfloor\frac{n}{i}\rfloor)$

```cpp
#include <bits/stdc++.h>

#define N 5000020
#define LL long long
#define mod 1000000007
#define div2 ((mod+1)/2)
#define div6 ((mod+1)/6)

using namespace std;

int n, prime[N], v[N];
LL phi[3][N];

map<int, int> mp[3];

int sum(int t, int x) { //calculate 1^t + 2^t + ... + x^t
    if (t == 0) return x;
    if (t == 1) return 1ll * x * (x + 1) % mod * div2 % mod;
    if (t == 2) return 1ll * x * (x + 1) % mod * (2ll * x % mod + 1) % mod * div6 %
        mod;
    if (t == 3) return 1ll * x * x % mod * (x + 1) % mod * (x + 1) % mod * div2 %
        mod * div2 % mod;
}

int getphi(int t, int x) {
    if (x < N) return phi[t][x];
    if (mp[t].find(x) != mp[t].end()) return mp[t][x];
    LL ans = 0; int r = 0;
    for (int l = 2; l <= x; l = r + 1) {
        r = x / (x / l);
```

```
        ans += 1ll * getphi(t, x / l) * (((LL)sum(t, r) - sum(t, l - 1) + mod) % mod) % mod;
        ans %= mod;
    }
    ans = (LL)sum(t + 1, x) - ans + mod;
    ans %= mod;
    mp[t][x] = ans;
    return (int)ans;
}

int main() {
    memset(v, 0, sizeof(v));
    int top = 0;
    phi[0][1] = 1, phi[1][1] = 1, phi[2][1] = 1;
    for (int i = 2; i < N; ++i) {
        if (!v[i]) prime[++top] = i, phi[0][i] = i - 1, phi[1][i] = 1ll * i * phi[0][i] % mod, phi[2][i] = 1ll * i * phi[1][i] % mod;
        for (int j = 1; j <= top && prime[j] * i < N; ++j) {
            v[i * prime[j]] = 1;
            if (i % prime[j] == 0) {
                phi[0][i * prime[j]] = phi[0][i] * prime[j];
                phi[1][i * prime[j]] = 1ll * phi[1][i] * prime[j] % mod * prime[j] % mod;
                phi[2][i * prime[j]] = 1ll * phi[2][i] * prime[j] % mod * prime[j] % mod * prime[j] % mod;
                break;
            } else {
                phi[0][i * prime[j]] = phi[0][i] * (prime[j] - 1);
                phi[1][i * prime[j]] = 1ll * phi[1][i] * (prime[j] - 1) % mod * prime[j] % mod;
                phi[2][i * prime[j]] = 1ll * phi[2][i] * (prime[j] - 1) % mod * prime[j] % mod * prime[j] % mod;
            }
        }
    }
    for (int i = 2; i < N; ++i) {
        phi[0][i] = (phi[0][i] + phi[0][i - 1]) % mod;
        phi[1][i] = (phi[1][i] + phi[1][i - 1]) % mod;
        phi[2][i] = (phi[2][i] + phi[2][i - 1]) % mod;
    }
}
```

6.16 FFT

6.16.1 普通 FFT

```
namespace FFT {
    const int maxn = 65537;
    const double pi = acos(-1.0);
```

```

    struct comp {
        double real , imag;
        comp() {}
        comp(double real , double imag): real(real) , imag(imag) {}
        friend inline comp operator+(const comp &a , const comp &b) {
            return comp(a.real + b.real , a.imag + b.imag);
        }
        friend inline comp operator-(const comp &a , const comp &b) {
            return comp(a.real - b.real , a.imag - b.imag);
        }
        friend inline comp operator*(const comp &a , const comp &b) {
            return comp(a.real * b.real - a.imag * b.imag , a.real * b.imag + a.imag
                 * b.real);
        }
    };

    comp A[maxn] , B[maxn];
    int rev[maxn], m, len;

    inline void init(int n) {
        for (m = 1, len = 0; m < n + n; m <<= 1 , len ++);
        for (int i = 0; i < m; ++i) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (len -
            1));
        for (int i = 0; i < m; ++i) A[i] = B[i] = comp(0, 0);
    }

    inline void dft(comp *a , int v) {
        for (int i = 0; i < m; ++i) if (i < rev[i]) swap(a[i] , a[rev[i]]);
        for (int s = 2; s <= m; s <<= 1) {
            comp g(cos(2 * pi / s) , v * sin(2 * pi / s));
            for (int k = 0; k < m; k += s) {
                comp w(1 , 0);
                for (int j = 0; j < s / 2; ++j) {
                    comp &u = a[k + j + s / 2], &v = a[k + j];
                    comp t = w * u;
                    u = v - t;
                    v = v + t;
                    w = w * g;
                }
            }
        }
        if (v == -1)
            for (int i = 0; i < m; ++i) a[i].real /= m , a[i].imag /= m;
    }
}
```

6.16.2 模任意素数 FFT

注意：调用 $mulmod$ 前先调用 $init$ 。调用 $mulmod$ 前请确保 a, b 数组足够大（比 $2n$ 大的 2 的整数次幂）且经过初始化。

```
namespace FFT {
    const long double pi = acos(-1.0);

    struct comp {
        long double real, imag;
        comp() {}
        comp(long double real, long double imag) : real(real), imag(imag) {}
        friend inline comp operator + (const comp &a, const comp &b) {
            return comp(a.real + b.real, a.imag + b.imag);
        }
        friend inline comp operator - (const comp &a, const comp &b) {
            return comp(a.real - b.real, a.imag - b.imag);
        }
        friend inline comp operator * (const comp &a, const comp &b) {
            return comp(a.real * b.real - a.imag * b.imag, a.real * b.imag + a.imag
                * b.real);
        }
        inline comp conj() {
            return comp(real, -imag);
        }
    };

    comp A[maxn], B[maxn];
    int rev[maxn], m, len;

    inline void init(int n) {
        for (m = 1, len = 0; m < n + n; m <<= 1, ++len);
        for (int i = 0; i < m; ++i) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (len -
            1));
        for (int i = 0; i < m; ++i) A[i] = B[i] = comp(0, 0);
    }

    inline void dft(comp *a, int v) {
        for (int i = 0; i < m; ++i) if (i < rev[i]) swap(a[i], a[rev[i]]);
        for (int s = 2; s <= m; s <<= 1) {
            comp g(cos(2 * pi / s), v * sin(2 * pi / s));
            for (int k = 0; k < m; k += s) {
                comp w(1, 0);
                for (int j = 0; j < s / 2; ++j) {
                    comp &u = a[k + j + s / 2], &v = a[k + j];
                    comp t = w * u;
                    u = v - t;
                    v = v + t;
                    w = w * g;
                }
            }
```

```
        }
        if (v == -1)
            for (int i = 0; i < m; ++i) a[i].real /= m, a[i].imag /= m;
    }

    inline void mulmod(int *a, int *b, int *c) { // c = a * b % mod, c不能为a或b
        int M = sqrt(mod);
        for (int i = 0; i < m; ++i) {
            A[i] = comp(a[i] / M, a[i] % M);
            B[i] = comp(b[i] / M, b[i] % M);
        }
        dft(A, 1); dft(B, 1);
        static comp t[maxn];
        for (int i = 0; i < m; ++i) {
            int j = i ? m - i : 0;
            t[i] = ((A[i] + A[j].conj()) * (B[j].conj() - B[i]) + (A[j].conj() - A[i
                ]) * (B[i] + B[j].conj())) * comp(0, 0.25);
        }
        dft(t, -1);
        for (int i = 0; i < m; ++i)
            c[i] = (LL)(t[i].real + 0.5) % mod * M % mod;
        for (int i = 0; i < m; ++i) {
            int j = i ? m - i : 0;
            t[i] = (A[j].conj() - A[i]) * (B[j].conj() - B[i]) * comp(-0.25, 0) +
                comp(0, 0.25) * (A[i] + A[j].conj()) * (B[i] + B[j].conj());
        }
        dft(t, -1);
        for (int i = 0; i < m; ++i)
            c[i] = (1ll * c[i] + (LL)(t[i].real + 0.5) + (LL)(t[i].imag + 0.5) % mod
                 * M * M % mod) % mod;
    }
};
```

6.17 FWT

给定长度为 2^n 的序列 $A[0 \cdots 2^n - 1], B[0 \cdots 2^n - 1]$ ，求这两序列的

or 卷积：$C_k = \sum_{i\ or\ j=k} A_i B_j$

and 卷积：$C_k = \sum_{i\ and\ j=k} A_i B_j$

xor 卷积：$C_k = \sum_{i\ xor\ j=k} A_i B_j$

```
void FWT(int *a, int n) {
    for (int d = 1; d < n; d <<= 1)
        for (int m = d << 1, i = 0; i < n; i += m)
            for (int j = 0; j < d; ++j) {
                int x = a[i + j], y = a[i + j + d];
                //or: a[i + j + d] = x + y;
                //and: a[i + j] = x + y;
                //xor: a[i + j] = x + y, a[i + j + d] = x - y;
```

```
                // 如答案要求取模，此处记得取模
            }
}

void UFWT(int *a, int n) {
    for (int d = 1; d < n; d <<= 1)
        for (int m = d << 1, i = 0; i < n; i += m)
            for (int j = 0; j < d; ++j) {
                int x = a[i + j], y = a[i + j + d];
                //or: a[i + j + d] = y - x;
                //and: a[i + j] = x - y;
                //xor: a[i + j] = (x + y) / 2, a[i + j + d] = (x - y) / 2;
                // 如答案要求取模，此处记得取模
            }
}
```

6.18 求原根

接口：LL p_root(LL p);

输入：一个素数 p

输出：p 的原根

```
#include <bits/stdc++.h>
#define LL long long

using namespace std;

vector <LL> a;

LL pow_mod(LL base, LL times, LL mod) {
    LL ret = 1;
    while (times) {
        if (times&1) ret = ret * base % mod;
        base = base * base % mod;
        times>>=1;
    }
    return ret;
}

bool g_test(LL g, LL p) {
    for (LL i = 0; i < a.size(); ++i)
        if (pow_mod(g, (p-1)/a[i], p) == 1) return 0;
    return 1;
}

LL p_root(LL p) {
    LL tmp = p - 1;
    for (LL i = 2; i <= tmp / i; ++i)
        if (tmp % i == 0) {
```

```
            a.push_back(i);
            while (tmp % i == 0)
                tmp /= i;
        }
    if (tmp != 1) a.push_back(tmp);
    LL g = 1;
    while (1) {
        if (g_test(g, p)) return g;
        ++g;
    }
}

int main() {
    LL p;
    cin >> p;
    cout << p_root(p) << endl;
}
```

6.19 NTT

NTT 公式:

$$y_n = \sum_{i=0}^{d-1} x_i (g^{\frac{P-1}{d}})^{in} \bmod P$$

```
#define mod 998244353
#define gg 3

int power(int base, int times) {
    int ans = 1;
    while (times) {
        if (times & 1) ans = 1ll * ans * base % mod;
        base = 1ll * base * base % mod;
        times >>= 1;
    }
    return ans;
}

void NTT(int *x, int n, int reverse) {
    static int rev[N];
    int m = 1, len = 0;
    for (; m < n + n; m <<= 1, ++len);
    for (int i = 0; i < m; ++i) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (len - 1))
        ;
    for (int i = 0; i < m; ++i)
        if (i < rev[i])
            swap(x[i], x[rev[i]]);
    for (int h = 2; h <= m; h <<= 1) {
        int wn = power(gg, (mod - 1) / h);
        if (reverse == -1) wn = power(wn, mod - 2);
```

```
        for (int i = 0; i < m; i += h) {
            int w = 1;
            for (int j = i; j < i + h / 2; ++j) {
                int u = x[j];
                int v = 1ll * w * x[j + h / 2] % mod;
                x[j] = (u + v) % mod;
                x[j + h / 2] = (u - v + mod) % mod;
                w = 1ll * w * wn % mod;
            }
        }
    }
    if (reverse == -1) {
        int t = power(m, mod - 2);
        for (int i = 0; i < m; ++i)
            x[i] = 1ll * x[i] * t % mod;
    }
}

int A[N], B[N];
int main() {
    memset(A, 0, sizeof(A));
    memset(B, 0, sizeof(B));
    NTT(A, len, 1); NTT(B, len, 1);
    for (int i=0;i<len;i++) A[i] = 1ll * A[i] * B[i] % mod;
    NTT(A, len, -1);
}
```

6.19.1 NTT 常用原根表

这张表格仅包含 $2^{18}k+1$ 的质数。

模数	最大长度	原根	模数	最大长度	原根	模数	最大长度	原根
786433	262144	10	5767169	524288	3	7340033	1048576	3
8650753	262144	10	10223617	262144	5	11272193	262144	3
13631489	1048576	15	14155777	524288	7	14942209	262144	11
16515073	262144	5	21495809	524288	3	22806529	262144	13
23068673	2097152	3	26214401	1048576	3	27000833	262144	3
28311553	1048576	5	29884417	524288	5	33292289	262144	3
35389441	262144	7	36175873	524288	7	37224449	524288	3
38535169	262144	11	40370177	524288	3	41680897	262144	5
42729473	262144	3	52166657	262144	3	63700993	262144	5
64749569	262144	6	68681729	524288	3	69206017	2097152	5
70254593	1048576	3	72613889	262144	3	74711041	262144	7
77070337	524288	13	81788929	2097152	7	83361793	524288	5
83623937	262144	3	85196801	262144	3	87293953	262144	7
90439681	262144	7	93585409	262144	13	93847553	524288	3

100139009	524288	3	101711873	1048576	3	103284737	524288	3
104857601	4194304	3	107216897	262144	3	111149057	2097152	3
113246209	4194304	7	114032641	262144	11	115081217	262144	3
117964801	524288	14	118751233	262144	5	120324097	262144	7
120586241	1048576	6	125042689	262144	13	126615553	262144	10
127664129	262144	3	130809857	262144	3	132120577	2097152	5
136314881	2097152	3	138412033	4194304	5	140771329	262144	7
141557761	1048576	26	142344193	262144	7	145489921	262144	7
147849217	1048576	5	151257089	262144	3	155189249	4194304	6
156499969	262144	7	158072833	262144	5	158334977	1048576	3
159645697	262144	15	163577857	4194304	23	167510017	262144	5
167772161	33554432	3	169869313	2097152	5	173801473	262144	5
175636481	524288	3	178782209	524288	3	184811521	262144	13
185597953	1048576	5	186646529	2097152	3	187432961	262144	6
189530113	262144	5	191365121	524288	3	199229441	2097152	3
200540161	262144	7	204472321	1048576	19	206307329	262144	3
207880193	262144	3	211025921	262144	6	211812353	2097152	3
214171649	262144	3	215482369	524288	13	215744513	262144	3
217317377	262144	3	218628097	524288	5	219676673	524288	3
221249537	1048576	3	222035969	262144	3	224133121	262144	23
228065281	524288	7	230424577	262144	5	230686721	4194304	6
231473153	262144	3	234356737	524288	7	236716033	262144	5
239337473	262144	3	239861761	262144	11	240648193	524288	5
244842497	524288	3	246415361	1048576	3	249561089	2097152	3
253493249	262144	3	254279681	524288	3	256376833	524288	7
257949697	2097152	5	260571137	524288	3	261881857	262144	7
263454721	262144	11	269221889	262144	3	270532609	2097152	22
270794753	262144	3	274726913	2097152	3	276037633	262144	15
277086209	262144	6	284950529	262144	3	285474817	262144	7
288882689	524288	3	290455553	1048576	3	302252033	262144	3
302776321	262144	17	305135617	1048576	5	306708481	524288	19
311427073	1048576	7	319291393	524288	5	323223553	262144	5
325844993	262144	3	328728577	524288	10	329515009	262144	13
329777153	524288	5	330301441	1048576	22	332660737	262144	10
336068609	524288	3	336855041	262144	3	340000769	262144	3
347078657	1048576	3	349962241	262144	7	351797249	524288	3
359661569	1048576	3	360972289	262144	7	361758721	1048576	29
371458049	262144	3	374603777	262144	3	376963073	524288	3
377487361	8388608	7	383778817	2097152	5	384040961	262144	3
386400257	524288	3	387186689	262144	3	387973121	2097152	6
390332417	262144	3	391643137	524288	5	395837441	524288	6

399507457	1048576	5	404226049	524288	7	409993217	1048576	3
413925377	262144	3	415236097	4194304	5	416808961	524288	37
424148993	524288	3	429391873	524288	10	433586177	524288	3
434896897	262144	15	438829057	524288	5	442761217	262144	5
444334081	262144	37	447741953	1048576	3	452198401	262144	11
455344129	262144	13	458752001	524288	6	459276289	2097152	11
460849153	524288	5	462684161	262144	3	463470593	2097152	3
464781313	262144	5	466354177	262144	10	468713473	1048576	5
469762049	67108864	3	471072769	262144	7	473694209	262144	6
475267073	262144	3	478937089	262144	13	483131393	262144	3
483655681	262144	14	487063553	524288	3	489422849	262144	3
493879297	1048576	10	495452161	524288	11	498597889	524288	7
500432897	262144	5	511967233	262144	5	517472257	524288	5
518520833	524288	3	524812289	524288	3	526123009	262144	7
529268737	262144	5	531628033	1048576	5	533463041	262144	3
536608769	262144	3	537133057	262144	5	539754497	262144	3
540540929	524288	3	541327361	262144	3	549978113	524288	3
551288833	262144	5	552861697	262144	5	555220993	524288	7
561774593	262144	3	564658177	524288	5	568066049	262144	3
569638913	262144	3	570163201	262144	7	570949633	524288	5
576454657	262144	10	576716801	2097152	6	581959681	1048576	11
582746113	262144	5	583794689	262144	3	584581121	524288	3
590872577	524288	3	595591169	8388608	3	597688321	2097152	11
605028353	1048576	3	605552641	524288	17	606339073	262144	5
607911937	262144	7	608698369	524288	7	611844097	524288	5
612892673	524288	3	615776257	262144	5	619184129	524288	3
621281281	524288	7	626262017	262144	3	629932033	262144	14
632553473	262144	3	635437057	2097152	11	637009921	524288	17
638058497	524288	3	639369217	262144	5	639631361	2097152	6
644087809	262144	11	645922817	8388608	3	648019969	2097152	17
649592833	524288	5	651952129	262144	7	655360001	1048576	3
657719297	262144	3	660078593	524288	3	663224321	524288	3
665583617	262144	3	666894337	4194304	5	675545089	262144	11
675807233	524288	3	681312257	262144	3	683409409	262144	13
683671553	4194304	3	684982273	262144	5	687603713	262144	3
690749441	262144	3	692846593	262144	5	699138049	262144	19
699924481	524288	17	703070209	524288	11	704905217	262144	3
710410241	524288	3	710934529	2097152	17	712769537	262144	3
714342401	262144	3	715128833	2097152	3	717488129	262144	3
718274561	1048576	3	720633857	262144	3	725876737	262144	7
730595329	262144	17	734527489	524288	7	737673217	524288	11

740294657	2097152	3	741605377	262144	11	745537537	1048576	5
748158977	524288	3	753664001	262144	3	754974721	16777216	11
758906881	262144	11	759693313	524288	5	760741889	524288	3
763887617	524288	3	769130497	524288	15	770703361	1048576	11
771489793	262144	10	772538369	262144	6	775421953	524288	5
781975553	262144	3	782499841	262144	11	786432001	2097152	7
790364161	262144	14	792199169	524288	3	793509889	262144	11
795082753	262144	5	798228481	262144	13	799014913	2097152	13
800063489	1048576	3	800849921	262144	6	801374209	262144	14
802160641	1048576	11	808714241	262144	3	810024961	524288	13
811859969	262144	3	813170689	524288	13	813432833	262144	3
818937857	1048576	5	820248577	262144	5	820510721	524288	3
821297153	262144	3	824180737	2097152	5	824442881	262144	3
825753601	524288	23	828112897	262144	10	829685761	262144	19
833617921	1048576	13	835452929	262144	3	839385089	524288	3
842530817	524288	3	844627969	524288	17	844890113	262144	3
848560129	262144	22	850395137	1048576	3	851705857	262144	5
860618753	262144	3	862978049	1048576	3	863764481	262144	3
864550913	524288	3	867434497	262144	5	872153089	262144	7
873725953	262144	10	875298817	262144	5	879230977	524288	15
880803841	8388608	26	881590273	262144	5	883949569	1048576	7
885522433	524288	5	888668161	524288	14	889454593	262144	15
894959617	524288	10	896008193	524288	3	897318913	262144	5
897581057	8388608	3	899678209	2097152	7	900464641	262144	7
903086081	262144	3	907018241	1048576	3	907542529	524288	7
907804673	262144	3	908328961	262144	26	909377537	262144	3
913309697	1048576	3	914096129	262144	3	918552577	4194304	5
919339009	262144	59	919601153	1048576	3	924844033	2097152	5
925892609	1048576	3	932970497	262144	3	935329793	4194304	3
938475521	1048576	3	940572673	1048576	7	943718401	4194304	7
946339841	524288	3	948699137	262144	3	950009857	2097152	7
951582721	524288	14	957349889	1048576	6	958136321	262144	3
958922753	524288	3	962592769	2097152	7	962854913	262144	3
969146369	262144	3	971243521	262144	28	972029953	1048576	10
975175681	2097152	17	976224257	1048576	3	977534977	262144	5
979107841	262144	11	980156417	262144	3	983826433	262144	11
985661441	4194304	3	993263617	262144	5	995622913	524288	5
998244353	8388608	3	1004535809	2097152	3	1005060097	524288	5
1006108673	524288	3	1007681537	1048576	3	1010565121	262144	7
1012924417	2097152	5	1015283713	262144	5	1018429441	262144	11
1019478017	262144	3	1023148033	262144	7	1036779521	262144	3

1037303809	262144	21	1045430273	1048576	3	1049100289	524288	7
1051721729	1048576	6	1052508161	262144	3	1053818881	1048576	7
1056178177	262144	5	1056440321	524288	3	1062469633	262144	5
1068236801	262144	3	1073479681	262144	11			

6.19.2 多项式求逆元

对于一个多项式 $A(x)$ ，如果存在 $B(x)$ 满足 $\deg(B) \le \deg(A)$ 并且 $A(x)B(x) \equiv 1 (\bmod x^n)$ ，那么称 $B(x)$ 为 $A(x)$ 在 $\bmod x^n$ 意义下的逆元，记为 $A^{-1}(x)$ 。

```
// x := 1 / y
void inverse(int n0, int *x, const int *y) {
    static int fy[N];
    x[0] = 1;
    for (int i = 1; i < n0; i <<= 1) {
        for (int j = 0; j < 4 * i; ++j) {
            fy[j] = (j < 2 * i) ? y[j] : 0;
            if (j >= i) x[j] = 0;
        }
        NTT(fy, 2 * i, 1);
        NTT(x, 2 * i, 1);
        for (int j = 0; j < 4 * i; ++j) {
            x[j] = (2 * x[j] - 1ll * x[j] * x[j] % mod * fy[j]) % mod;
            if (x[j] < 0) x[j] += mod;
        }
        NTT(x, 2 * i, -1);
    }
}
```

6.19.3 多项式取对数

```
// x := log(y)
void logarithm(int n0, int *x, int *y) {
    static int tmp[N];
    static int invs[N];
    inverse(n0, x, y);
    for (int i = 0; i < n0 * 2; ++i) {
        tmp[i] = i < n0 - 1 ? 1ll * y[i + 1] * (i + 1) % mod : 0;
        if (i >= n0) x[i] = 0;
    }
    NTT(tmp, n0, 1);
    NTT(x, n0, 1);
    for (int i = 0; i < n0 * 2; ++i)
        x[i] = 1ll * x[i] * tmp[i] % mod;
    NTT(x, n0, -1);
    invs[1] = 1;
    for (int i = 2; i < n0; ++i)
        invs[i] = (mod - 1ll * mod / i * invs[mod % i] % mod) % mod;
```

```
    for (int i = n0 - 1; i; --i)
        x[i] = 1ll * x[i - 1] * invs[i] % mod;
    x[0] = 0;
}
```

6.19.4 多项式取指数

```
// a := exp(b)
void exponent(int n0, int *a, int *b) {
    static int fb[N], x[N], y[N];
    a[0] = 1;
    for (int i = 1; i < n0; i <<= 1) {
        for (int j = 0; j < i * 2; ++j)
            y[j] = (j < i) ? a[j] : 0;
        logarithm(i * 2, x, y);
        for (int j = 0; j < 4 * i; ++j) {
            fb[j] = !j;
            if (j < 2 * i) {
                fb[j] = (fb[j] + b[j]) % mod;
                fb[j] = (fb[j] + mod - x[j]) % mod;
            }
            if (j >= i) a[j] = 0;
        }
        NTT(a, 2 * i, 1);
        NTT(fb, 2 * i, 1);
        for (int j = 0; j < 4 * i; ++j)
            a[j] = 1ll * a[j] * fb[j] % mod;
        NTT(a, 2 * i, -1);
    }
}
```

6.20 Berlekamp Messay 算法求线性递推式

适合所有 $S_n = \sum_{i=1}^{L} a_i S_{n-i}$ 的递推式。只需在 $vector < int > t$ 中输入前 $2L$ 项，即可计算出第 m 项的值 modulo MOD 。

时间复杂度 $O(L^2 \log(m))$ 。

异常处理：若提示 48 行 assertion error (assert(l * 2 + 1 < s.size()) ，则表示输入项数不足 $2L + 2$ 项，需要更多的项来确定线性递推式。

```
1 #include <bits/stdc++.h>
2 
3 using namespace std;
4 typedef long long ll;
5 
6 int MOD;
7 
8 int bin(int a, int n) {
```

```
    int res = 1;
    while (n) {
        if (n & 1) res = 1LL * res * a % MOD;
        a = 1LL * a * a % MOD;
        n >>= 1;
    }
    return res;
}

int inv(int x) {
    return bin(x, MOD - 2);
}

vector<int> berlekamp(vector<int> s) {
    int l = 0;
    vector<int> la(1, 1);
    vector<int> b(1, 1);
    for (int r = 1; r <= (int)s.size(); r++) {
        int delta = 0;
        for (int j = 0; j <= l; j++) {
            delta = (delta + 1LL * s[r - 1 - j] * la[j]) % MOD;
        }
        b.insert(b.begin(), 0);
        if (delta != 0) {
            vector<int> t(max(la.size(), b.size()));
            for (int i = 0; i < (int)t.size(); i++) {
                if (i < (int)la.size()) t[i] = (t[i] + la[i]) % MOD;
                if (i < (int)b.size()) t[i] = (t[i] - 1LL * delta * b[i] % MOD + MOD
                    ) % MOD;
            }
            if (2 * l <= r - 1) {
                b = la;
                int od = inv(delta);
                for (int &x : b) x = 1LL * x * od % MOD;
                l = r - l;
            }
            la = t;
        }
    }
    assert(la.size() == l + 1);
    assert(l * 2 + 1 < s.size());
    reverse(la.begin(), la.end());
    return la;
}

vector<int> mul(vector<int> a, vector<int> b) {
    vector<int> c(a.size() + b.size() - 1);
    for (int i = 0; i < (int)a.size(); i++) {
        for (int j = 0; j < (int)b.size(); j++) {
            c[i + j] = (c[i + j] + 1LL * a[i] * b[j]) % MOD;
```

```
        }
    }
    vector<int> res(c.size());
    for (int i = 0; i < (int)res.size(); i++) res[i] = c[i] % MOD;
    return res;
}

vector<int> mod(vector<int> a, vector<int> b) {
    if (a.size() < b.size()) a.resize(b.size() - 1);

    int o = inv(b.back());
    for (int i = (int)a.size() - 1; i >= b.size() - 1; i--) {
        if (a[i] == 0) continue;
        int coef = 1LL * o * (MOD - a[i]) % MOD;
        for (int j = 0; j < (int)b.size(); j++) {
            a[i - (int)b.size() + 1 + j] = (a[i - (int)b.size() + 1 + j] + 1LL *
                coef * b[j]) % MOD;
        }
    }
    while (a.size() >= b.size()) {
        assert(a.back() == 0);
        a.pop_back();
    }
    return a;
}

vector<int> bin(int n, vector<int> p) {
    vector<int> res(1, 1);
    vector<int> a(2); a[1] = 1;
    while (n) {
        if (n & 1) res = mod(mul(res, a), p);
        a = mod(mul(a, a), p);
        n >>= 1;
    }
    return res;
}

void solve() {
    int m = 22;
    vector<int> t;
    t.push_back(1);
    t.push_back(9);
    t.push_back(41);
    t.push_back(109);
    t.push_back(205);
    t.push_back(325);
    t.push_back(473);
    t.push_back(649);
    t.push_back(853);
    t.push_back(1085);
```

```
    t.push_back(1345);
    t.push_back(1633);
    t.push_back(1949);
    t.push_back(2293);

    MOD = 998244353;
    vector<int> v = berlekamp(t);
    vector<int> o = bin(m - 1, v);
    int res = 0;
    for (int i = 0; i < (int)o.size(); i++) res = (res + 1LL * o[i] * t[i]) % MOD;
    printf("%d\n", res);
}

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

6.21 幂和

$$\sum_{i=1}^{n} i^1 = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$$

$$\sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4} = \frac{1}{4}n^4 + \frac{1}{2}n^3 + \frac{1}{4}n^2$$

$$\sum_{i=1}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{1}{5}n^5 + \frac{1}{2}n^4 + \frac{1}{3}n^3 - \frac{1}{30}n$$

$$\sum_{i=1}^{n} i^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12} = \frac{1}{6}n^6 + \frac{1}{2}n^5 + \frac{5}{12}n^4 - \frac{1}{12}n^2$$

$$\sum_{i=1}^{n} i^6 = \frac{n(n+1)(2n+1)(3n^4+6n^3-3n+1)}{42} = \frac{1}{7}n^7 + \frac{1}{2}n^6 + \frac{1}{2}n^5 - \frac{1}{6}n^3 + \frac{1}{42}n$$

6.22 蔡勒公式

$$w = (\lfloor\frac{c}{4}\rfloor - 2c + y + \lfloor\frac{y}{4}\rfloor + \lfloor\frac{13(m+1)}{5}\rfloor + d - 1) \bmod 7$$

w： 0 星期日，1 星期一，2 星期二，3 星期三，4 星期四，5 星期五，6 星期六

c： 年份前两位数

y： 年份后两位数

m： 月（$3 \leq m \leq 14$ ，即在蔡勒公式中，1、2 月要看作上一年的 13、14 月来计算）

d： 日

6.23 皮克定理

给定顶点坐标均是整点（或正方形格点）的简单多边形（凸多边形），皮克定理说明了其面积 S 和内部格点数目 n 、边上格点数目 s 的关系：$S = n + \frac{s}{2} + 1$ 。

6.24 组合数 lcm

$(n+1)lcm(C(n,0),C(n,1),...,C(n,k)) = lcm(n+1,n,n-1,...,n-k+1)$

6.25 区间 lcm 的维护

对于一个数，将其分解质因数，若有因子 p^k ，那么拆分出 k 个数 $p,p^2,...,p^k$ ，权值都为 p ，那么查询区间 $[l,r]$ 内所有数的 lcm 的答案 = 所有在该区间中出现过的数的权值之积，可持久化线段树维护即可。

7 几何

7.1 二维计算几何

7.1.1 计算几何误差修正

```
const double pi = acos(-1.0);
const double eps = 1e-8;

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

inline int sgn(double x) {
    if (x < -eps) return -1;
    return x > eps;
}

inline int cmp(double x, double y) {
    return sgn(x - y);
}
```

7.1.2 计算几何点类

成员函数：

read()	输入一个点
norm()	计算向量的模长

相关函数：

double sqr(double x)	计算一个数的平方
double det(const point &a, const point &b)	计算两个向量的叉积
double dot(const point &a, const point &b)	计算两个向量的点积
double dis(const point &a, const point &b)	计算两个点的距离
point rotate_point(const point &p, double A)	$\overrightarrow{OP}$绕原点逆时针旋转 A 弧度

```cpp
struct point {
    double x, y;
    point() : x(0), y(0) {}
    point(double a, double b) : x(a), y(b) {}
    inline void read() {
        scanf("%lf%lf", &x, &y);
    }
    inline friend point operator + (const point &a, const point &b) {
        return point(a.x + b.x, a.y + b.y);
    }
    inline friend point operator - (const point &a, const point &b) {
        return point(a.x - b.x, a.y - b.y);
    }
    inline friend bool operator == (const point &a, const point &b) {
        return cmp(a.x, b.x) == 0 && cmp(a.y, b.y) == 0;
    }
    inline friend point operator * (const double &a, const point &b) {
        return point(a * b.x, a * b.y);
    }
    inline friend point operator / (const point &a, const double &b) {
        return point(a.x / b, a.y / b);
    }
    inline double norm() const {
        return sqrt(sqr(x) + sqr(y));
    }
};

inline double det(const point &a, const point &b) {
    return a.x * b.y - a.y * b.x;
}

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

inline double dis(const point &a, const point &b) {
    return (a - b).norm();
}

inline point rotate_point(const point &p, double A) {
    double tx = p.x, ty = p.y;
    return point(tx * cos(A) - ty * sin(A), tx * sin(A) + ty * cos(A));
}
```

7.1.3 计算几何线段类

相关函数:

bool point_on_segment(const point &p, const segment &l) 判断点 p 是否在线段 l 上 (含端点)

double point_to_segment_dist(const point &p, const segment &l) 求点 p 到线段 l 的距离

point sym_point(const point &p, const segment &l) 求点 p 关于线段 l 的对称点

point point_proj_line(const point &p, const segment &l) 求点 p 到线段 l 的垂足

bool parallel(const segment &a, const segment &b) 判断线段 a 和线段 b 是否平行

point intersect_point(const segment &a, const segment &b) 求直线 a 与直线 b 的交点（如要求线段 a 与线段 b 的交点，应先判断是否有）

bool is_segment_intersect(const segment &l1, const segment &l2) 判断线段 a 与线段 b 是否相交（含端点）（如不含端点，将 $\leq$ 改为 $<$ ）

bool is_line_intersect_segment(const point &p1, const point &p2, const segment &l) 判断直线 p_1p_2 是否与线段 l 相交

bool is_half_line_intersect_segment(const point &p1, const point &p2, const segment &l) 判断射线 p_1p_2 是否与线段 l 相交（含端点 p_1 ）（如不含端点 p_1 ，将 $\geq$ 改为 $>$ ）

```
struct segment {
    point a, b;
    segment() {}
    segment(point x, point y) : a(x), b(y) {}
    void read() {
        a.read(); b.read();
    }
};

// determine whether point p is on segment l
bool point_on_segment(const point &p, const segment &l) {
    if ((cmp(l.a.x, p.x) <= 0 || cmp(l.b.x, p.x) <= 0) &&
        (cmp(l.a.x, p.x) >= 0 || cmp(l.b.x, p.x) >= 0) &&
        (cmp(l.a.y, p.y) <= 0 || cmp(l.b.y, p.y) <= 0) &&
        (cmp(l.a.y, p.y) >= 0 || cmp(l.b.y, p.y) >= 0)) {
        return sgn(det(p - l.a, l.b - l.a)) == 0;
    }
    return 0;
}

// determine the distance from the point p to segment l
double point_to_segment_dist(const point &p, const segment &l) {
    if (dis(l.a, l.b) < eps) return dis(p, l.a);
    if (sgn(dot(l.b - l.a, p - l.a)) < 0) return dis(l.a, p);
    if (sgn(dot(l.a - l.b, p - l.b)) < 0) return dis(l.b, p);
    return fabs(det(l.b - l.a, p - l.a)) / dis(l.b, l.a);
}

// determine the symmetrical point of point p on segment l
point sym_point(const point &p, const segment &l) {
    double a = l.b.x - l.a.x;
    double b = l.b.y - l.a.y;
    double t = ((p.x - l.a.x) * a + (p.y - l.a.y) * b) / (a * a + b * b);
    return point(2 * l.a.x + 2 * a * t - p.x, 2 * l.a.y + 2 * b * t - p.y);
}

```

```cpp
point point_proj_line(const point &p, const segment &l) {
    double r = dot((l.b - l.a), (p - l.a)) / dot(l.b - l.a, l.b - l.a);
    return l.a + r * (l.b - l.a);
}

bool parallel(const segment &a, const segment &b) {
    return sgn(det(a.a - a.b, b.a - b.b)) == 0;
}

point intersect_point(const segment &a, const segment &b) {
    double s1 = det(a.a - b.a, b.b - b.a);
    double s2 = det(a.b - b.a, b.b - b.a);
    return (s1 * a.b - s2 * a.a) / (s1 - s2);
}

// determine whether segment l1 intersects with segment l2
bool is_segment_intersect(const segment &l1, const segment &l2) {
    const point &s1 = l1.a, &e1 = l1.b;
    const point &s2 = l2.a, &e2 = l2.b;
    if ( cmp( min(s1.x, e1.x), max(s2.x, e2.x) ) <= 0 &&
         cmp( min(s1.y, e1.y), max(s2.y, e2.y) ) <= 0 &&
         cmp( min(s2.x, e2.x), max(s1.x, e1.x) ) <= 0 &&
         cmp( min(s2.y, e2.y), max(s1.y, e1.y) ) <= 0 &&
         sgn( det(s2 - s1, e2 - s1) ) * sgn( det(s2 - e1, e2 - e1) ) <= 0 &&
         sgn( det(s1 - s2, e1 - s2) ) * sgn( det(s1 - e2, e1 - e2) ) <= 0)
        return 1;
    return 0;
}

// determine whether line p1p2 intersects with segment l
bool is_line_intersect_segment(const point &p1, const point &p2, const segment &l) {
    assert(!(p1 == p2));
    return sgn( det(p1 - l.a, p2 - l.a) ) * sgn( det(p1 - l.b, p2 - l.b) ) <= 0;
}

// determine whether half-line p1p2 intersects with segment l
bool is_half_line_intersect_segment(const point &p1, const point &p2, const segment
    &l) {
    return is_line_intersect_segment(p1, p2, l) && sgn( det(p1 - l.a, p2 - l.a) ) *
        sgn( det(p1 - l.a, l.b - l.a) ) >= 0;
}
```

7.2 凸包

```cpp
typedef complex<int> point;
#define X real()
#define Y imag()
int n;
long long cross(point a,point b) {
```

```
    return 1ll * a.X * b.Y - 1ll * a.Y * b.X;
}
bool cmp(point a,point b) {
    return make_pair(a.X, a.Y) < make_pair(b.X, b.Y);
}
int convexHull(point p[],int n,point ch[]) {
    sort(p, p + n, cmp);
    int m = 0;
    for(int i = 0; i < n; ++i) {
        while(m > 1 && cross(ch[m-1] - ch[m-2], p[i] - ch[m-2]) <= 0) m--;
        ch[m++] = p[i];
    }
    int k = m;
    for(int i = n - 2; i >= 0; --i) {
        while(m > k && cross(ch[m-1] - ch[m-2], p[i] - ch[m-2]) <= 0) m--;
        ch[m++] = p[i];
    }
    if(n > 1) m--;
    return m;
}
```

7.3 半平面交

输入 vec1 表示所有的半平面 y >= kx + b 的参数 k 和 b 。

输出 vec2 表示下凸壳（对应 y >= kx + b）或者上凸壳（对应 y <= kx + b)。

```
vector< pair< LL, LL > > vec1, vec2;

LL getval(int t, LL x) {
    return vec2[t].first * x + vec2[t].second;
}

void solve() {
    // vec1 stores pair< k, b > for all plane y >=(or <=) kx + b
    sort(vec1.begin(), vec1.end());
    // reverse(vec1.begin(), vec1.end()); // if y <= kx + b
    for (int i = 0; i < vec1.size(); ++i) {
        while (vec2.size() >= 2) {
            LL k1 = vec2[vec2.size() - 2].first;
            LL b1 = vec2[vec2.size() - 2].second;
            LL k2 = vec2[vec2.size() - 1].first;
            LL b2 = vec2[vec2.size() - 1].second;
            LL k3 = vec1[i].first;
            LL b3 = vec1[i].second;
            if ((b2 - b1) * (k2 - k3) >= (b3 - b2) * (k1 - k2))
                vec2.pop_back();
            else
                break;
        }
        vec2.push_back(vec1[i]);
```

```
    }
}
```

8 黑科技和杂项

8.1 找规律

此方法已过时，请参照 “数学 > Berlekamp Messay 算法求线性递推式”。本法使用矩阵快速幂，效率 $O(L^3 \log(m))$，而用 Berlekamp 加多项式快速幂可以做到 $O(L^2 \log(m))$，故不推荐使用本法。

~~有些题目，只给一个正整数n，然后要求输出一个答案。这时，我们可以暴力得到小数据的解，用高斯消元得到递推式，然后用矩阵快速幂求解。~~

~~使用方法：~~

~~首先在 gauss.in 中输入小数据的解（$n = 1$ 时，$n = 2$ 时，$\cdots$），以EOF 结束。~~

~~依次运行 gauss.cpp，matrix.cpp，得到 matrix.out~~

~~将 matrix.out 中的文件粘贴在 main.cpp 中相应的位置中。注意模数一定要是质数。~~

```
//gauss.cpp
#include <bits/stdc++.h>
#define N 102
#define mod 1000000007
//caution: you can use this program iff mod is a prime.

using namespace std;

int n, m, k, a[N], g[N][N];

int power(int base, int times) {
    int ret = 1;
    while (times) {
        if (times & 1) ret = 1ll * ret * base % mod;
        base = 1ll * base * base % mod;
        times >>= 1;
    }
    return ret;
}

int test() {
    for (int i=0;i<m;i++) {
        for (int j=i;j<=m;j++)
            if (g[j][i]) {
                for (int k=i;k<=m;k++)
                    swap(g[i][k], g[j][k]);
                break;
            }
        if (g[i][i] == 0)
            return 0;
        for (int j=i+1;j<n;j++) {
            while (g[j][i]) {
```

```
                int t = 1ll * g[i][i] * power(g[j][i], mod - 2) % mod;
                for (int k=i;k<n;k++)
                    g[i][k] = (g[i][k] + mod - (1ll * t * g[j][k] % mod)) % mod;
                for (int k=i;k<=m;k++)
                    swap(g[i][k], g[j][k]);
            }
        }
        int t = power(g[i][i], mod - 2);
        for (int j = 0; j <= m; ++j)
            g[i][j] = 1ll * g[i][j] * t % mod;
    }
    for (int i = m; i < n; ++i)
        if (g[i][m]) return 0;
    for (int i = m - 1; i >= 0; --i) {
        int t = power(g[i][i], mod - 2);
        g[i][i] = 1;
        g[i][m] = 1ll * g[i][m] * t % mod;
        for (int j = 0; j < i; ++j)
            g[j][m] = (g[j][m] + mod - 1ll * g[i][m] * g[j][i] % mod) % mod;
    }
    printf("%d\n", m);
    for (int i = 0; i < m; ++i)
        printf("%d␣", g[i][m]);
    puts("");
    for (int i = 0; i < m - 1; ++i)
        printf("%d␣", a[i]);
    puts("1");
    return 1;
}

int main() {
    freopen("gauss.in", "r", stdin);
    freopen("gauss.out", "w", stdout);
    k = 0;
    while (~scanf("%d", &a[k++])) ;
    for (int sm = 1; sm <= k - sm; ++sm) {
        n = k - sm - 1;
        m = sm + 1;
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j <= sm; ++j)
                g[i][j] = a[i + j];
            g[i][m] = 1;
            swap(g[i][m - 1], g[i][m]);
        }
        if (test()) return 0;
    }
    puts("no␣solution");
    return 0;
}
```

```cpp
//matrix.cpp
#include <bits/stdc++.h>
#define N 102
using namespace std;

int n, a[N];

int main() {
    freopen("gauss.out", "r", stdin);
    freopen("matrix.out", "w", stdout);
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) scanf("%d", &a[i]);
    printf("#define␣M␣%d\n", n);
    printf("const␣int␣trans[M][M]␣=␣{\n");
    for (int i = 0; i < n; ++i) {
        printf("\t{");
        for (int j = 0; j < n; ++j) {
            int t;
            if (j < n - 2) t = i == j + 1;
            else if (j == n - 2) t = a[i];
            else t = i == n - 1;
            printf("%s%d", j == 0 ? "" : ",␣", t);
        }
        printf("}%s\n", i == n - 1 ? "" : ",");
    }
    printf("};\n");
    printf("const␣int␣pref[M]␣=␣{");
    for (int i = 0; i < n; ++i) {
        int x;
        scanf("%d", &x);
        printf("%d%s", x, i == n - 1 ? "};\n" : ",␣");
    }
    return 0;
}
```

```cpp
//main.cpp
#include <bits/stdc++.h>
using namespace std;

/* paste matrix.out here. */

#define mod 1000000007

struct Matrix {
    int c[M][M];
    void clear() { memset(c, 0, sizeof(c)); }
    void identity() { clear(); for (int i = 0; i < M; ++i) c[i][i] = 1; }
    void base() { memcpy(c, trans, sizeof(trans)); }
    friend Matrix operator * (const Matrix &a, const Matrix &b) {
```

```
        Matrix c; c.clear();
        for (int i = 0; i < M; ++i)
            for (int j = 0; j < M; ++j)
                for (int k = 0; k < M; ++k)
                    c.c[i][j] = (c.c[i][j] + 1ll * a.c[i][k] * b.c[k][j] % mod) %
                        mod;
        return c;
    }
} start, base;

Matrix power(Matrix base, int times) {
    Matrix ret; ret.identity();
    while (times) {
        if (times & 1) ret = ret * base;
        base = base * base;
        times >>= 1;
    }
    return ret;
}

int main() {
    int tot;
    scanf("%d", &tot);
    while (tot--) {
        int n;
        scanf("%d", &n);
        start.clear();
        for (int i = 0; i < M; ++i) start.c[0][i] = pref[i];
        base.base();
        base = power(base, n - 1);
        start = start * base;
        printf("%d\n", start.c[0][0]);
    }
    return 0;
}
```

8.2 分数类

```
#define LL long long

struct frac {
    LL x, y;
    frac(LL _x = 0, LL _y = 1) {
        x = _x;
        y = _y;
        LL g = __gcd(abs(x), abs(y));
        x /= g;
        y /= g;
        if (y < 0) {
```

```
            x = -x;
            y = -y;
        }
    }

    inline friend frac operator + (const frac &lhs, const frac &rhs) {
        return frac(lhs.x * rhs.y + rhs.x * lhs.y, lhs.y * rhs.y);
    }

    inline friend frac operator - (const frac &lhs, const frac &rhs) {
        return frac(lhs.x * rhs.y - rhs.x * lhs.y, lhs.y * rhs.y);
    }

    inline friend frac operator - (const frac &lhs) {
        return frac(-lhs.x, lhs.y);
    }

    inline friend frac operator * (const frac &lhs, const frac &rhs) {
        return frac(lhs.x * rhs.x, lhs.y * rhs.y);
    }

    inline friend frac operator / (const frac &lhs, const frac &rhs) {
        return frac(lhs.x * rhs.y, lhs.y * rhs.x);
    }

    inline friend bool operator == (const frac &lhs, const frac &rhs) {
        return lhs.x * rhs.y == rhs.x * lhs.y;
    }

    inline friend bool operator != (const frac &lhs, const frac &rhs) {
        return lhs.x * rhs.y != rhs.x * lhs.y;
    }

    inline friend bool operator < (const frac &lhs, const frac &rhs) {
        return lhs.x * rhs.y < rhs.x * lhs.y;
    }

    inline friend bool operator > (const frac &lhs, const frac &rhs) {
        return lhs.x * rhs.y > rhs.x * lhs.y;
    }

    inline friend bool operator <= (const frac &lhs, const frac &rhs) {
        return lhs.x * rhs.y <= rhs.x * lhs.y;
    }

    inline friend bool operator >= (const frac &lhs, const frac &rhs) {
        return lhs.x * rhs.y >= rhs.x * lhs.y;
    }

    inline void print() const {
```

```
        printf("%lld/%lld\n", x, y);
    }
};
```

8.3 取模整数类

如果需要用模意义下的除法，需定义常量 D 为除数的最大值，并执行 init_inv() 。

```
struct mod;
mod* inv;

struct mod {
    static constexpr int MOD = 1000 * 1000 * 1000 + 7; // std=c++11
    mod(int x_) : x((x_ % MOD + MOD) % MOD) {}
    mod() = default;
    int x = 0;
    inline mod operator *(mod other) const {
        return ((long long)x * other.x) % MOD;
    }
    inline mod& operator *=(mod other) {
        x = ((long long)x * other.x) % MOD;
        return *this;
    }
    inline mod operator +(mod other) const {
        int res = x + other.x;
        if (res >= MOD) {
            res -= MOD;
        }
        return res;
    }
    inline mod& operator +=(mod other) {
        if ((x += other.x) >= MOD) {
            x -= MOD;
        }
        return *this;
    }
    inline mod operator -(mod other) const {
        int res = x - other.x;
        if (res < 0) {
            res += MOD;
        }
        return res;
    }
    inline mod& operator -=(mod other) {
        if ((x -= other.x) < 0) {
            x += MOD;
        }
        return *this;
    }
    inline mod operator /(mod other) const {
```

```
        return (*this) * inv[other.x];
    }
    inline mod& operator /=(mod other) {
        return *this *= inv[other.x];
    }
    inline bool operator ==(mod other) const {
        return x == other.x;
    }
    inline mod operator -() const {
        return x != 0 ? MOD - x : 0;
    }
};

void init_inv() {
    inv = new mod[D];
    inv[1] = 1;
    for (int i = 2; i < D; i++) {
        inv[i] = (mod::MOD - (long long)mod::MOD / i * inv[mod::MOD % i].x % mod::
            MOD) % mod::MOD;
    }
}
```

8.4 多项式类

```
struct poly {
    vector<mod> C;
    poly() {}
    explicit poly(const vector<mod> &C_) : C(C_) {}
    static const poly zero;
    inline int deg() const {
        return (int)C.size() - 1;
    }
    inline mod operator[](int x) const {
        return (x < 0 || x > deg()) ? mod(0) : C[x];
    }
    inline mod& operator[](int x) {
        if (x > deg()) {
            C.resize(x + 1);
        }
        return C[x];
    }
    inline friend poly operator +(const poly& a, const poly& b) {
        vector<mod> c(max(a.deg(), b.deg()) + 1);
        for (int i = 0; i < c.size(); i++) {
            c[i] = a[i] + b[i];
        }
        return poly(c);
    }
    inline friend poly operator -(const poly& a, const poly& b) {
```

```cpp
        vector<mod> c(max(a.deg(), b.deg()) + 1);
        for (int i = 0; i < c.size(); i++) {
            c[i] = a[i] - b[i];
        }
        return poly(c);
    }
    inline bool isZero() const {
        return C.empty();
    }
    inline friend poly operator *(const poly& a, const poly& b) {
        if (a.isZero() || b.isZero()) {
            return zero;
        }
        vector<mod> c(1 + a.deg() + b.deg());
        for (int i = 0; i <= a.deg(); i++) {
            for (int j = 0; j <= b.deg(); j++) {
                c[i + j] += a[i] * b[j];
            }
        }
        return poly(c);
    }
    inline poly derivative() const {
        if (isZero()) {
            return zero;
        }
        vector<mod> res(deg());
        for (int i = 0; i < deg(); i++) {
            res[i] = C[i + 1] * (i + 1);
        }
        return poly(res);
    }
    inline poly primitive() const {
        if (isZero()) {
            return zero;
        }
        vector<mod> res(2 + deg());
        for (int i = 1; i <= 1 + deg(); i++) {
            res[i] = C[i - 1] / i;
        }
        return poly(res);
    }
    inline mod operator()(mod x) const {
        mod res = 0;
        for (int i = deg(); i >= 0; i--) {
            res = res * x + C[i];
        }
        return res;
    }
    // Expand P(x+t).
    inline poly shift(int t) const {
```

```
        poly res;
        res[deg()];
        vector<mod> binomial(deg() + 1, 0);
        binomial[0] = 1;
        for (int i = 0; i <= deg(); i++) {
            mod cur = 1;
            for (int j = i; j >= 0; j--) {
                res[j] += C[i] * binomial[j] * cur;
                cur *= t;
            }
            if (i == deg()) {
                break;
            }
            for (int j = i + 1; j > 0; j--) {
                binomial[j] += binomial[j - 1];
            }
        }
        return res;
    }
};
```

8.5 高精度计算

```
#include<algorithm>
using namespace std;
const int N_huge=850,base=100000000;
char s[N_huge*10];
struct huge{
    typedef long long value;
    value a[N_huge];int len;
    void clear(){len=1;a[len]=0;}
    huge(){clear();}
    huge(value x){*this=x;}
    huge operator =(huge b){
        len=b.len;for (int i=1;i<=len;++i)a[i]=b.a[i]; return *this;
    }
    huge operator =(value x){
        len=0;
        while (x)a[++len]=x%base,x/=base;
        if (!len)a[++len]=0;
        return *this;
    }
    huge operator +(huge b){
        int L=len>b.len?len:b.len;huge tmp;
        for (int i=1;i<=L+1;++i)tmp.a[i]=0;
        for (int i=1;i<=L;++i){
            if (i>len)tmp.a[i]+=b.a[i];
            else if (i>b.len)tmp.a[i]+=a[i];
            else {
```

```
                tmp.a[i]+=a[i]+b.a[i];
                if (tmp.a[i]>=base){
                    tmp.a[i]-=base;++tmp.a[i+1];
                }
            }
        }
        if (tmp.a[L+1])tmp.len=L+1;
            else tmp.len=L;
        return tmp;
    }
    huge operator -(huge b){
        int L=len>b.len?len:b.len;huge tmp;
        for (int i=1;i<=L+1;++i)tmp.a[i]=0;
        for (int i=1;i<=L;++i){
            if (i>b.len)b.a[i]=0;
            tmp.a[i]+=a[i]-b.a[i];
            if (tmp.a[i]<0){
                tmp.a[i]+=base;--tmp.a[i+1];
            }
        }
        while (L>1&&!tmp.a[L])--L;
        tmp.len=L;
        return tmp;
    }
    huge operator *(huge b){
        int L=len+b.len;huge tmp;
        for (int i=1;i<=L;++i)tmp.a[i]=0;
        for (int i=1;i<=len;++i)
            for (int j=1;j<=b.len;++j){
                tmp.a[i+j-1]+=a[i]*b.a[j];
                if (tmp.a[i+j-1]>=base){
                    tmp.a[i+j]+=tmp.a[i+j-1]/base;
                    tmp.a[i+j-1]%=base;
                }
            }
        tmp.len=len+b.len;
        while (tmp.len>1&&!tmp.a[tmp.len])--tmp.len;
        return tmp;
    }
    pair<huge,huge> divide(huge a,huge b){
        int L=a.len;huge c,d;
        for (int i=L;i;--i){
        c.a[i]=0;d=d*base;d.a[1]=a.a[i];
            int l=0,r=base-1,mid;
            while (l<r){
                mid=(l+r+1)>>1;
                if (b*mid<=d)l=mid;
                    else r=mid-1;
            }
            c.a[i]=l;d-=b*l;
```

```
        }
        while (L>1&&!c.a[L])--L;c.len=L;
        return make_pair(c,d);
    }
    huge operator /(value x){
        value d=0;huge tmp;
        for (int i=len;i;--i){
            d=d*base+a[i];
            tmp.a[i]=d/x;d%=x;
        }
        tmp.len=len;
        while (tmp.len>1&&!tmp.a[tmp.len])--tmp.len;
        return tmp;
    }
    value operator %(value x){
        value d=0;
        for (int i=len;i;--i)d=(d*base+a[i])%x;
        return d;
    }
    huge operator /(huge b){return divide(*this,b).first;}
    huge operator %(huge b){return divide(*this,b).second;}
    huge &operator +=(huge b){*this=*this+b;return *this;}
    huge &operator -=(huge b){*this=*this-b;return *this;}
    huge &operator *=(huge b){*this=*this*b;return *this;}
    huge &operator ++(){huge T;T=1;*this=*this+T;return *this;}
    huge &operator --(){huge T;T=1;*this=*this-T;return *this;}
    huge operator ++(int){huge T,tmp=*this;T=1;*this=*this+T;return tmp;}
    huge operator --(int){huge T,tmp=*this;T=1;*this=*this-T;return tmp;}
    huge operator +(value x){huge T;T=x;return *this+T;}
    huge operator -(value x){huge T;T=x;return *this-T;}
    huge operator *(value x){huge T;T=x;return *this*T;}
    huge operator *=(value x){*this=*this*x;return *this;}
    huge operator +=(value x){*this=*this+x;return *this;}
    huge operator -=(value x){*this=*this-x;return *this;}
    huge operator /=(value x){*this=*this/x;return *this;}
    huge operator %=(value x){*this=*this%x;return *this;}
    bool operator ==(value x){huge T;T=x;return *this==T;}
    bool operator !=(value x){huge T;T=x;return *this!=T;}
    bool operator <=(value x){huge T;T=x;return *this<=T;}
    bool operator >=(value x){huge T;T=x;return *this>=T;}
    bool operator <(value x){huge T;T=x;return *this<T;}
    bool operator >(value x){huge T;T=x;return *this>T;}
    bool operator <(huge b){
        if (len<b.len)return 1;
        if (len>b.len)return 0;
        for (int i=len;i;--i){
            if (a[i]<b.a[i])return 1;
            if (a[i]>b.a[i])return 0;
        }
        return 0;
```

```
    }
    bool operator ==(huge b){
        if (len!=b.len)return 0;
        for (int i=len;i;--i)
            if (a[i]!=b.a[i])return 0;
        return 1;
    }
    bool operator !=(huge b){return !(*this==b);}
    bool operator >(huge b){return !(*this<b||*this==b);}
    bool operator <=(huge b){return (*this<b)||(*this==b);}
    bool operator >=(huge b){return (*this>b)||(*this==b);}
    void str(char s[]){
        int l=strlen(s);value x=0,y=1;len=0;
        for (int i=l-1;i>=0;--i){
            x=x+(s[i]-'0')*y;y*=10;
            if (y==base)a[++len]=x,x=0,y=1;
        }
        if (!len||x)a[++len]=x;
    }
    void read(){
        scanf("%s",s);this->str(s);
    }
    void print(){
        printf("%d",(int)a[len]);
        for (int i=len-1;i;--i){
            for (int j=base/10;j>=10;j/=10){
                if (a[i]<j)printf("0");
                    else break;
            }
            printf("%d",(int)a[i]);
        }
        printf("\n");
    }
}f[1005];
int main(){
    f[1]=f[2]=1;
    for(int i=3;i<=1000;i++)f[i]=f[i-1]+f[i-2];
}
```

8.6 读入优化

8.6.1 普通读入优化

```
#define rd RD<int>
#define rdll RD<long long>
template <typename Type>
inline Type RD() {
    Type x = 0;
    int flag = 0;
```

```
    char c = getchar();
    while (!isdigit(c) && c != '-')
        c = getchar();
    (c == '-') ? (flag = 1) : (x = c - '0');
    while (isdigit(c = getchar()))
        x = x * 10 + c - '0';
    return flag ? -x : x;
}
inline char rdch() {
    char c = getchar();
    while (!isalpha(c)) c = getchar();
    return c;
}
```

8.6.2 HDU 专用读入优化

接口：

int rd(int &x); 读入一个整数，保存在变量 x 中。如正常读入，返回值为 1 ，否则返回 EOF(-1)

int rdll(long long &x);

```
#define rd RD<int>
#define rdll RD<long long>

const int S = 2000000; // 2MB

char s[S], *h = s+S, *t = h;

inline char getchr(void) {
    if(h == t) {
        if (t != s + S) return EOF;
        t = s + fread(s, 1, S, stdin);
        h = s;
    }
    return *h++;
}

template <class T>
inline int RD(T &x) {
    char c = 0;
    int sign = 0;
    for (; !isdigit(c); c = getchr()) {
        if (c == EOF)
            return -1;
        if (c == '-')
            sign ^= 1;
    }
    x = 0;
    for (; isdigit(c); c = getchr())
        x = x * 10 + c - '0';
    if (sign) x = -x;
```

```
    return 1;
}
```

8.7 O2 优化

```
#define OPTIM __attribute__((optimize("-O2")))
```

8.8 位运算及其运用

8.8.1 枚举子集

枚举 i 的非空子集 j

```
for (int j = i; j; j = (j - 1) & i);
```

8.8.2 求 1 的个数

```
int __builtin_popcount(unsigned int x);
```

8.8.3 求前缀 0 的个数

```
int __builtin_clz(unsigned int x);
```

8.8.4 求后缀 0 的个数

```
int __builtin_ctz(unsigned int x);
```

9 Sublime Text

9.1 License

```
-- BEGIN LICENSE --
TwitterInc
200 User License
EA7E-890007
1D77F72E 390CDD93 4DCBA022 FAF60790
61AA12C0 A37081C5 D0316412 4584D136
94D7F7D4 95BC8C1C 527DA828 560BB037
D1EDDD8C AE7B379F 50C9D69D B35179EF
2FE898C4 8E4277A8 555CE714 E1FB0E43
D5D52613 C3D12E98 BC49967F 7652EED2
9D2D2E61 67610860 6D338B72 5CF95C69
E36B85CC 84991F19 7575D828 470A92AB
-- END LICENSE --
```

9.2 Preferences.sublime-settings

```
{
    "font_size": 13,
    "show_encoding": true,
    "update_check": false
}
```

10 Vim

```
syntax on
set cindent
set nu
set tabstop=4
set shiftwidth=4
set background=dark
set relativenumber

inoremap <C-j> <down>
inoremap <C-k> <up>
inoremap <C-h> <left>
inoremap <C-l> <right>
```