ICPC World Finals 2019 Templates

SYSU_Balloon

2019 年 3 月 5 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; }

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

```
struct Query { int l, r, id, l_group; } query[N];
int v[N], ans[N];
bool cmp(const Query &a, const Query &b) { if (a.l_group != b.l_group) return a.
    l_group < b.l_group; return dfn[a.r] < dfn[b.r]; }
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;
}

for (int i=1;i<=m;i++) {
    if (dfn[query[i].l] > dfn[query[i].r]) swap(query[i].l, query[i].r);
    query[i].id = i; query[i].l_group = dfn[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 hash(int l, int r) { return (LL)(f[r]-(LL)f[l-1]*pow[r-l+1]%D+D)%D; }
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

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 扩展 KMP

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, 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;
        }
    }
}
```

2.4 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);
}
```

2.5 AC 自动机

```
struct Node { int next[26]; int terminal, fail; };
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<26;i++)
            if (node[x].next[i]) {
                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];
            }
    }
}
```

2.6 后缀数组

参数 m 表示字符集的大小，即 $0 \le r_i < m$

```
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) { return r[a]==r[b] && a+l<n && b+l<n &&
    r[a+l]==r[b+l]; }
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++;
    }
    for (i=0;i<n;i++) rank[sa[i]]=i; rank[n] = -1;
    for (i=j=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--;
    }
```

```
}
```

2.7 后缀自动机

下面的代码是求两个串的 LCS（最长公共子串）。

```
#define M (N << 1)
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：① Right(str) 表示 str 在母串 S 中所有出现的结束位置集合；② 一个状态 s 表示的所有子串 Right 集合相同，为 Right(s)；③ Parent(s) 满足 Right(s) 是 Right(Parent(s)) 的真子集，并且 Right(Parent(s)) 的大小最小；④ Parent 函数可以表示一个树形结构。不妨叫它 Parent 树；⑤ 一个 Right 集合和一个长度定义了一个子串；⑥ 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)] ；⑦ max(Parent(s)) = min(s) - 1；⑧ 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：① pre[s] 为上述定义中的 Parent(s)；② step[s] 为从初始状态走到 s 状态最多需要多少步；③ refer[s] 为上述定义中的 refer(s)；④ size[s] 为 Right(s) 集合的大小；⑤ topo[s] 为 Parent 树的拓扑序，根（初始状态）在前。

2.8 回文树

① len[i] 表示编号为 i 的节点表示的回文串的长度（一个节点表示一个回文串）② next[i][c] 表示编号为 i 的节点表示的回文串在两边添加字符 c 以后变成的回文串的编号（和字典树类似）。③ fail[i] 表示节点 i 失配以后跳转不等于自身的节点 i 表示的回文串的最长后缀回文串（和 AC 自动机类似）。④ cnt[i] 表示节点 i 表示的本质不同的串的个数（建树时求出的不是完全的，最后 count() 函数跑一遍以后才是正确的）⑤ num[i] 表示以节点 i 表示的最长回文串的最右端点为回文串结尾的回文串个数。⑥ last 指向新添加一个字母后所形成的最长回文串表示的节点。⑦ st[i] 表示第 i 次添加的字符（一开始设 st[0] = -1（可以是任意一个在串 S 中不会出现的字符））。⑧ tot 表示添加的节点个数。⑨ n 表示添加的字符个数。

【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 $\{a, b\}$

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

```cpp
            next[cur][c]=tot; num[tot] = num[fail[tot]] + 1; answer[n]='1';
        } else answer[n]='0';
        last=next[cur][c]; cnt[last] ++;
    }
    void count () { for (int i=tot-1; i>=0; --i) cnt[fail[i]] += cnt[i]; }
    //父亲累加儿子的cnt，因为如果fail[v]=u，则u一定是v的子回文串。
} pam;
n=strlen(st+1); pam.init();
for (int i=1;i<=n;i++) pam.insert(st[i]-'a');
```

3 数据结构

3.1 ST 表

```cpp
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]); }
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

① change 将编号为 x 的点的权值增加 p；② euclid_lower_bound 欧几里得距离的平方，下界；③ euclid_upper_bound 欧几里得距离的平方，上界；④ manhattan_lower_bound 曼哈顿距离，下界；⑤ manhattan_upper_bound 曼哈顿距离，上界；⑥ add 添加一个点（注意此处的添加可能导致这棵树不平衡，慎用!）；⑦ ask(p, X, Y, ans) 询问距离点 (X, Y) 最远的一个点的距离，ans 需传入无穷小；⑧ ask(p, x1, y1, x2, y2) 查询矩形范围内所有点的权值和。

```cpp
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);
```

```cpp
    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;
}
void change(int x, int p) {
    x = id[x]; // 将点的编号映成排序后的编号
    for (t[x].val += p; x; x = t[x].f) t[x].sum += p;
}
inline long long sqr(long long x) { return x * x; }
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)); }
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])); }
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);
}
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));
}
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, 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); }
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
        ); }
}
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;
}
for (int i = 1; i <= n; ++i) 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：① push 插入一个元素；② merge 合并两个堆，注意，合并后原来那个堆将不可访问；③ top 返回堆顶元素；④ pop 删除堆顶元素；⑤ size 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 $i \leq y$ 且满足一些条件（如 $a[i] \geq w$，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 $i \geq x$ 且满足一些条件（如 $a[i] \geq w$ ，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：① ADD x y d 将 $[x, y]$ 的所有数加上 d ；② REVERSE x y 将 $[x, y]$ 翻转；③ INSERT x p 将 p 插入到第 x 个数的后面；④ DEL x 将第 x 个数删除。

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

```
void rotate(int x) {
    if (!x) return; int y = father[x], w = son[y][1]==x;
    sc(y, son[x][w^1], w); sc(father[y], x, son[father[y]][1]==y); sc(x, y, w^1);
}
void flushdown(int x) {
    int t=0; for (; x; x=father[x]) q[++t]=x;
    for (; t; t--) pushdown(q[t]);
}
void Splay(int x, int root=0) {
    flushdown(x);
    while (father[x] != root) { int y=father[x], w=son[y][1]==x;
        if (father[y] != root && son[father[y]][w]==y) rotate(y);
        rotate(x); }
}
int find(int k) {
    Splay(rt);
    while (1) { pushdown(rt);
        if (size[son[rt][0]]+1==k) Splay(rt), return rt;
        else if (size[son[rt][0]]+1<k) k-=size[son[rt][0]]+1, rt=son[rt][1];
        else rt=son[rt][0]; }
}
int split(int x, int y) {
    int fx = find(x), fy = find(y+2); Splay(fx); Splay(fy, fx); return son[fy][0]; }
void add(int x, int y, int d) { //add d to each number in a[x]...a[y]
    int t = split(x, y); lazy[t] += d; Splay(t); rt=t; }
void reverse(int x, int y) { // reverse the x-th to y-th elements
    int t = split(x, y); rev[t] ^= 1; Splay(t); rt=t; }
void insert(int x, int p) { // insert p after the x-th element
    int fx = find(x+1), 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); son[fy][0] = 0; Splay(fy); rt=fy; }
```

3.6 可持久化 Treap

接口：① insert 在当前第 x 个字符后插入 c；② del 删除第 x 个字符到第 y 个字符；③ copy 复制第 l 个字符到第 r 个字符，然后粘贴到第 x 个字符后；④ reverse 翻转第 x 个到第 y 个字符；⑤ query 表示询问当前第 x 个字符是什么。

```
char key[N];
bool rev[N];
int lc[N], rc[N], size[N]; // if size is long long, remember here
int n, root;
LL Rand() { return rd = (rd * 20372052ll + 25022087ll) % mod; }
void init() { n = root = 0; }
inline int copy(int x) { ++ n; key[n] = key[x]; (copy rev, lc, rc, size); return n;
    }
inline void pushdown(int x) {
```

```
    if (!rev[x]) return;
    if (lc[x]) lc[x] = copy(lc[x]); if (rc[x]) rc[x] = copy(rc[x]);
    swap(lc[x], rc[x]); rev[lc[x]] ^= 1; rev[rc[x]] ^= 1; rev[x] = 0;
}
inline void pushup(int x) { size[x] = size[lc[x]] + size[rc[x]] + 1; }
int merge(int u, int v) {
    if (!u || !v) return u+v; pushdown(u); pushdown(v);
    int t = Rand() % (size[u] + size[v]), r;  // if size is long long, remember here
    if (t < size[u]) r = copy(u), rc[r] = merge(rc[u], v);
    else r = copy(v), lc[r] = merge(u, lc[v]);
    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 == size[u]) return copy(u);
    if (y <= size[lc[u]]) return split(lc[u], x, y);
    int t = size[lc[u]] + 1; // if size is long long, remember here
    if (x > t) return split(rc[u], x-t, y-t);
    int num = copy(u); lc[num]=split(lc[u], x, t-1); rc[num]=split(rc[u], 1, y-t);
    pushup(num); return num;
}
void insert(int x, char c) {
    int t1 = split(root, 1, x), t2 = split(root, x+1, size[root]);
    key[++n] = c; lc[n] = rc[n] = rev[n] = 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, size[root]); 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, size[
        root]);
    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, size
        [root]);
    rev[t2] ^= 1; root = merge(merge(t1, t2), t3); }
char query(int k) {
    int x = root;
    while (1) { pushdown(x);
        if (k <= size[lc[x]]) x = lc[x];
        else if (k == size[lc[x]] + 1) return key[x];
        else k -= size[lc[x]] + 1, x = rc[x]; }
}
```

3.7 可持久化并查集

接口：① merge 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 ② GetFather 询问 time 时刻及以前的连边状态中，x 所属的集合

```
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]=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 点分治

```
void getsize(int x, int root = 0) {
    size[x] = 1; son[x] = 0; int dd = 0;
    for (int p = gh[x]; p; p = edge[p].next) {
        int y = edge[p].adj;
        if (y == root || !vis[y]) continue;
        size[x] += size[y];
        if (size[y] > dd) dd = size[y], son[x] = y;
    }
}
int getroot(int x) {
    int sz = size[x];
    while (size[son[x]] > sz/2) x = son[x]; return x;
}
void dc(int x) {
    getsize(x); x = getroot(x);
    vis[x] = 1;
    for (int p = gh[x]; p; p = edge[p].next) {
        int y = edge[p].adj;
        if (vis[y]) continue;
        dc(y);
    }
    vis[x] = 0;
}
```

4.2 Link Cut Tree

① 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 。② minval 维护的是链上 val 最小值。③ sumval2 维护的是子树 val2 的和。

```
int f[N], son[N][2], sz[N], rev[N], tot;
int val[N], minid[N], minval[N];
```

```cpp
int val2[N], sumval2[N]; // 记得开 long long 。注意两个都要开 long long ，因为 val2 还包含了虚儿子的子树和。
stack<int> s;
void init(int i) {
    tot = max(tot, i); son[i][0] = son[i][1] = 0; f[i] = rev[i] = 0;
    if (i == 0) sz[i] = 0, val[i] = minval[i] = inf, minid[i] = i, val2[i] = sumval2[i] = 0;
    else sz[i] = 1, val[i] = minval[i] = VAL, minid[i] = i, val2[i] = sumval2[i] = VAL2;
}
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]];
    sumval2[x] = sumval2[son[x][0]] + sumval2[son[x][1]] + val2[x];
}
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), val2[x] += sumval2[son[x][1]], son[x][1] = y, val2[x] -= sumval2[son[x][1]], 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);
    // 如果需要维护子树和 val2, sumval2 ，这样是不够的。因为增加了虚边，所以需要修改 val2 值。将上面的代码替换为以下代码：
    // makeroot(x); makeroot(y); f[x] = y; val2[y] += sumval2[x]; up(y); access(x);
}
void cutf(int x) { access(x); splay(x); f[son[x][0]] = 0; son[x][0] = 0; up(x); } // 它和父亲的边
void cut(int x, int y) { makeroot(x); cutf(y); } // 切断 x 与 y 之间的边（须保证 x 与 y 相邻）
int ask(int x, int y) { makeroot(x); access(y); splay(y); return minid[y]; } // 询问 x 到 y 之间取得最小值的点
int querymin_cut(int x, int y) { int m = ask(x, y); makeroot(x); cutf(m); makeroot(y); cutf(m); return val[m]; } // 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）
void link(int x, int y, int w) { init(++tot); val[tot] = minval[tot] = w; link(x, tot); link(y, tot); } // 在 x 和 y 之间添加一条权值为 w 的边（将边视为点插入）
int getsumval2(int x, int y) { makeroot(x); access(y); return val2[y]; } // 令 x 为根，求 y 子树的 val2 的和
```

4.3 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;
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 Tarjan 有向图强联通分量

① 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 $dfn[u] \le low[v]$ 。② 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;
    }
}
```

5.2 Tarjan 双联通分量

以下代码为点双联通分量。若要更改为边双联通，在第 8 行将 $low[next] \ge dfn[x]$ 改为 $low[next] > dfn[x]$ ，并将 14 行 $vec[tot].push\_back(x)$ 删除。

```
1  void DFS(int x,int fa) {
2      vis[x]=true; dfn[x]=low[x]=++times; sk[++tp]=x;
3      for (int pt=first[x];pt;pt=e[pt].next) {
4          int next=e[pt].to; if (e[pt].id==fa) continue;
5          if (!vis[next]) {
6              DFS(next,e[pt].id);
7              low[x]=min(low[x],low[next]);
8              if (low[next]>=dfn[x]) {
9                  vec[++tot].clear();
10                 while (tp) {
11                     vec[tot].push_back(sk[tp--]);
12                     if (sk[tp+1]==next) break;
13                 }
14                 vec[tot].push_back(x);
15             }
16         } else if (dfn[next]>last) low[x]=min(low[x],dfn[next]);
17     }
18 }
19 for (i=1;i<=n;i++) if (!vis[i]) {
20     DFS(i,0); last=times;
21     if (tp) {
22         tot++; vec[tot].clear();
23         for (i=1;i<=tp;i++) vec[tot].push_back(sk[i]);
24         tp=0;
```

```
    }
}
```

5.3 2-SAT

记 $x \to y$ 的有向边表示选了 x 就要选 y 。

```
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) { 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) {}
    void work() { for (i) if (!dfn[i]) tarjan(i); }
} merge;
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; ans[ops[x]] = 1; //-1 NO, 1 YES
        }
    }
} ts;
merge.init(); merge.addedge(); merge.work();
for (int i = 1; i <= n; ++i) {
    int x = merge.belong[U(i, 0)], y = merge.belong[U(i, 1)];
    if (x==y) NO(); ts.ops[x]=y; ts.ops[y]=x;
}
ts.init(); ts.work();
puts("YES"); for (int i = 1; i <= n; ++i) select(ts.ans[merge.belong[U(i,1)] == 1);
```

5.4 网络流

5.4.1 最大流

注意：*top 要初始化为 1*

```
struct EDGE { int adj, w, next; } edge[M];
int n, top, gh[N], nrl[N], dist[N], q[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 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 ans = 0; while (bfs()) { memcpy(nrl, gh, sizeof(gh)); int t; while (t = dinic(S, inf)) ans += t; } return ans;
```

5.4.2 上下界有源汇网络流

①T 向 S 连容量为正无穷的边，将有源汇转化为无源汇。②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。③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。④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。⑤费用流：建完图后等价于求以 SS,TT 为源汇的费用流。

5.4.3 费用流

注意：*top 要初始化为 1*

```
struct EDGE { int adj, w, cost, next; } edge[M*2];
int gh[N], q[N], dist[N], v[N], pre[N], prev[N], top, S, T;
void addedge(int x, int y, int w, int cost) {x->y(w,cost); y->x(0,-cost);}
void clear() { top = 1; memset(gh, 0, sizeof(gh)); }
```

```
bool spfa() {} // 从S出发, 返回dist[T] != inf
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;
```

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 FFT

6.3.1 普通 FFT

```
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;
}
```

6.4 求原根

```
vector <LL> a;
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; }
}
```

6.5 NTT

```
#define mod 998244353
#define gg 3
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], 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; }
}
```

6.6 线性基

```
#define B 30
const int allset=(1<<B)-1;
struct LB {
    int mat[B],cnt;
    multiset<int> st;
    LB(){}
    void clear() { st.clear(); cnt=0; memset(mat,0,sizeof(mat)); }
    void add(int x) {
        for (int i=B-1;i>=0;i--) if ((x>>i)&1) {
            if (mat[i]) x^=mat[i];
            else { cnt++; mat[i]=x; break; }
        }
    }
    void fix() {
        for (int i=0;i<B;i++) if (mat[i])
            for (int j=i+1;j<B;j++) if ((mat[j]>>i)&1) mat[j]^=mat[i];
    }
    void preset() { //正确性待定
        fix(); for (int i=0;i<B;i++) if (mat[i]) st.insert(mat[i]);
    }
    int kth(int k) { //正确性待定
```

```
        int i=0,ans=0; if (k<=0||k>(1<<cnt)-1) return 0;//无解
        for (multiset<int>::iterator it=st.begin();it!=st.end();it++,i++)
            if ((k>>i)&1) ans^=(*it);
        return ans;
    }
    int getmax() {
        fix(); int ans=0;
        for (int i=B-1;i>=0;i--) if (ans^mat[i]>ans) ans^=mat[i];
        return ans;
    }
} tree[N*10];
```

6.7 高斯消元法实数方程

```
void Gauss(int n,int m) {
    int i,j,k,t;
    double mul;
    for (i=j=1;i<=n&&j<=m;i++,j++) {
        for (k=i+1;k<=n;k++)
            if (abs(mat[k][j])>abs(mat[i][j]))
                for (t=1;t<=m+1;t++) swap(mat[i][t],mat[k][t]);
        if (abs(mat[i][j])<eps) { i--; continue; }
        for (k=i+1;k<=n;k++) {
            mul=mat[k][j]/mat[i][j];
            for (t=1;t<=m+1;t++) mat[k][t]-=mat[i][t]*mul;
        }
    }
    for (i=n;i>=1;i--) { //solved表示那个变量是否确定
        for (j=1;j<=m;j++) if (abs(mat[i][j])>eps) break;
        if  (j>m) continue; solved[j]=true; ans[j]=mat[i][m+1];
        for (k=j+1;k<=m;k++)
            if (abs(mat[i][k])>eps&&!solved[k]) solved[j]=false;
        for (k=j+1;k<=m;k++) ans[j]-=ans[k]*mat[i][k];
        ans[j]/=mat[i][j];
    }
}
```

6.8 高斯消元法模方程

```
void Gauss(LL n,LL m) {
    LL i,j,k,t,lcm,muli,mulk;
    for (i=j=1;i<=n&&j<=m;i++,j++) {
        for (k=i;k<=n;k++) if (mat[k][j]) {
            for (t=1;t<=m+1;t++) swap(mat[k][t],mat[i][t]);
            break;
        }
        if (mat[i][j]==0) { i--; continue; }
        for (k=i+1;k<=n;k++) if (mat[k][j]) {
```

```
            lcm=mat[k][j]*mat[i][j]/__gcd(mat[k][j],mat[i][j]);
            muli=lcm/mat[i][j]; mulk=lcm/mat[k][j];
            for (t=1;t<=m+1;t++) {
                mat[k][t]=mat[k][t]*mulk-mat[i][t]*muli;
                mat[k][t]=(mat[k][t]%mod+mod)%mod;
            }
        }
    }
    for (i=n;i>=1;i--) {
        for (j=1;j<=m;j++) if (mat[i][j]) break;
        if  (j>m) continue; ans[j]=mat[i][m+1];
        for (k=j+1;k<=m;k++) ans[j]-=ans[k]*mat[i][k];
        ans[j]=(ans[j]*power(mat[i][j],mod-2)%mod+mod)%mod;
    }
}
```

6.9 蔡勒公式

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

① w ： 0 星期日，1 星期一，$\cdots$，6 星期六；② c ： 年份前两位数；③ y ： 年份后两位数；④ m ：月（$3 \le m \le 14$ ，即在蔡勒公式中，1、2 月要看作上一年的 13、14 月来计算）；⑤ d ： 日。

6.10 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：$S = n + \frac{s}{2} + 1$ 。

7 计算几何

7.1 凸包

```
bool cmp(const Point &a,const Point &b) {
    return F(a.x-b.x)<0||F(a.x-b.x)==0&&a.y<b.y; }
void Gram(int id[], int n) {
    int i,mid; sort(id,id+n,cmp); tp=0; //凸包从x最小的点出发，逆时针方向
     for (i=0;i<n;i++) {
         for (;tp>=2&&Cross(p[sk[tp-1]]-p[sk[tp-2]],p[id[i]]-p[sk[tp-1]])<=0;tp--);
        //有重点必须用<=不留共线点，无重点用<=不留共线点，无重点用<留共线点
         sk[tp++]=id[i];
     }
     mid=tp;
     for (i=n-2;i>=0;i--) {
         for (;tp>mid&&Cross(p[sk[tp-1]]-p[sk[tp-2]],p[id[i]]-p[sk[tp-1]])<=0;tp--);
        //有重点必须用<=不留共线点，无重点用<=不留共线点，无重点用<留共线点
         sk[tp++]=id[i];
     }
```

```
    if  (n>1) tp--;
}
```

7.2 定义

```
struct Point { double x,y; Point(){} Point(double _x,double _y):x(_x),y(_y){} };
struct Seg { Point a,b; Seg(){} Seg(Point _a,Point _b):a(_a),b(_b){} };
struct Circle { double x,y,r;
    Point pt() { return Point(x,y); }
    double Area() { return pi*r*r; }
};
Point operator +(const Point &a,const Point &b);
Point operator -(const Point &a,const Point &b);
Point operator *(const Point &a,double b);
Point operator /(const Point &a,double b);
int F(double x) { if (x>eps) return 1; if (x<-eps) return -1; return 0; }
bool operator ==(const Point &a,const Point &b) {
    return F(a.x-b.x)==0&&F(a.y-b.y)==0; }
double Dist(const Point &a) { return sqrt(a.x*a.x+a.y*a.y); }
double Dot(const Point &a,const Point &b) { return a.x*b.x+a.y*b.y; }
double Cross(const Point &a,const Point &b) { return a.x*b.y-a.y*b.x; }
Point Rotate(const Point &p,double a) { // 逆时针旋转
    return Point(p.x*cos(a)-p.y*sin(a),p.x*sin(a)+p.y*cos(a)); }
Point Inter(Seg a,Seg b) { // 两线段相交(前提有交点)
    double s=Cross(a.b-a.a,b.a-a.a),t=Cross(a.b-a.a,b.b-a.a);
    return b.a+(b.b-b.a)*s/(s-t); }
vector<Point> SegCir(Seg seg,Point pt,double r) { //线圆
    vector<Point> ans; double mul; Point vec,mid;
    ans.clear(); vec=Rotate(seg.b-seg.a,pi/2);
    mid=Inter(seg,Seg(pt,pt+vec));
    if (F(Dist(pt-mid)-r)>0) return ans;
    if (F(Dist(pt-mid)-r)==0) {
        ans.push_back(mid); ans.push_back(mid); return ans;
    }
    vec=seg.b-seg.a; mul=sqrt(r*r-Dist2(mid-pt))/Dist(vec);
    ans.push_back(mid+vec*mul); ans.push_back(mid-vec*mul);
    return ans;
}
vector<Point> Circir(Circle a,Circle b) { //圆圆相交
    vector<Point> ans; double dis,dis2,alpha; Point pa,pb,vec;
    ans.clear(); if (a.r<b.r) swap(a,b);
    pa=a.pt(); pb=b.pt(); vec=pb-pa;
    dis=Dist(vec); dis2=Dist2(vec);
    if (F(dis-(a.r+b.r))>0||F(dis-(a.r-b.r))<0) return ans;
    if (F(dis-(a.r+b.r))==0) {
        ans.push_back(pa+vec*a.r/(a.r+b.r)); return ans;
    }
    if (F(dis-(a.r-b.r))==0) {
        ans.push_back(pa+vec*a.r/(a.r-b.r)); return ans;
```

```
    }
    alpha=acos((a.r*a.r+dis2-b.r*b.r)/2/a.r/dis);
    ans.push_back(pa+Rotate(vec,alpha)*a.r/dis);
    ans.push_back(pa+Rotate(vec,-alpha)*a.r/dis);
    return ans;
}
double Bing(double ra,double rb,double dis) {
    double alpha,beta; if (ra<rb) swap(ra,rb);
    if (F(dis-(ra-rb))<=0) return pi*ra*ra;
    if (F(dis-(ra+rb))>=0) return pi*ra*ra+pi*rb*rb;
    alpha=acos((ra*ra+dis*dis-rb*rb)/2/dis/ra);
    beta=acos((rb*rb+dis*dis-ra*ra)/2/dis/rb);
    return (pi-alpha)*ra*ra+(pi-beta)*rb*rb+ra*dis*sin(alpha);
}
double Jiao(double ra,double rb,double dis) {
    return pi*ra*ra+pi*rb*rb-Bing(ra,rb,dis); }
Point Gongmid(Circle a,Circle b) { //正确性待定
    Point pa=a.pt(),pb=b.pt();
    return pa+(pb-pa)*a.r/(a.r+b.r); }
Point Gongright(Circle a,Circle b) {
    Point pa=a.pt(),pb=b.pt();
    return pa+(pb-pa)*a.r/(a.r-b.r); }
int Ptinpol(Point pt) {
    int wn=0;
    for(int i=0;i<n;i++) {
        if(Ins(pt,Seg(p[i],p[(i+1)%n]))) return 2;
        int k=F(Cross(p[(i+1)%n]-p[i],pt-p[i]));
        int d1=F(p[i].y-pt.y), d2=F(p[(i+1)%n].y-pt.y);
        if(k>0&&d1<=0&&d2>0)wn++;
        if(k<0&&d2<=0&&d1>0)wn--;
    }
    return wn!=0;
}
bool Cirinpol(Point pt) { //需要点在多边形内的前提
    double nearest=inf;
    for (int i=0;i<n;i++) {
        nearest=min(nearest,Dist(p[i]-pt));
        if  (F(Dot(pt-p[i],p[(i+1)%n]-p[i]))>0&&
            F(Dot(pt-p[(i+1)%n],p[i]-p[(i+1)%n]))>0)
        nearest=min(nearest,abs(Cross(p[i]-pt,p[(i+1)%n]-pt))/dis[i]);
    }
    return F(nearest-r)>=0;
}
bool Ins(const Point &p,const Seg &s) {
    return F(Cross(s.a-p,s.b-p))==0&&F(p.x-min(s.a.x,s.b.x))>=0&&
        F(p.x-max(s.a.x,s.b.x))<=0&&F(p.y-min(s.a.y,s.b.y))>=0&&
        F(p.y-max(s.a.y,s.b.y))<=0; }
double PS(const Point &p,const Seg &s) { // 点到线段最短距离
    if (F(Dot(p-s.a,s.b-s.a))<0||F(Dot(p-s.b,s.a-s.b))<0)
        return min(Dist(p-s.a),Dist(p-s.b));
```

```
    return abs(Cross(s.a-p,s.b-p))/Dist(s.a-s.b); }
double SS(const Seg &a,const Seg &b) { // 线段到线段最短距离
    return min(min(PS(a.a,b),PS(a.b,b)),min(PS(b.a,a),PS(b.b,a))); }
double Alpha(Point a,Point b) {
    double ans=atan2(b.y,b.x)-atan2(a.y,a.x);
    if (ans<0) ans=-ans; if (ans>pi) ans=2*pi-ans; return ans; }
double Shan(Circle c,double a) { return c.r*c.r*a/2; }
```

7.3 半平面交

```
bool Cmphp(Seg a,Seg b) {
    Point va=a.b-a.a, vb=b.b-b.a;
    double dega=atan2(va.y,va.x), degb=atan2(vb.y,vb.x);
    return F(dega-degb)<0||F(dega-degb)==0&&Cross(a.b-a.a,b.a-a.a)<0;
}
void HalfPlane(Seg hp[], int n, Point pol[], int &pols) {
    Point mid;
    hp[n++]=Seg(Point(-oo,-oo),Point(oo,-oo));
    hp[n++]=Seg(Point(oo,-oo),Point(oo,oo));
    hp[n++]=Seg(Point(oo,oo),Point(-oo,oo));
    hp[n++]=Seg(Point(-oo,oo),Point(-oo,-oo));
    sort(hp,hp+n,Cmphp);
    int tp=0, low=0, high=-1; //sk 0~tp-1
    for (int i=0;i<n;i++)
    if (high-low+1==0||F(Cross(sk[high].b-sk[high].a,hp[i].b-hp[i].a))) {
        for (;low<high;high--) {
            mid=Inter(sk[high],sk[high-1]);
            if (F(Cross(hp[i].b-hp[i].a,mid-hp[i].a))>0) break;
        }
        for (;low<high;low++) {
            mid=Inter(sk[low],sk[low+1]);
            if (F(Cross(hp[i].b-hp[i].a,mid-hp[i].a))>0) break;
        }
        sk[++high]=hp[i];
    }
    for (;low<high;high--) {
        mid=Inter(sk[high],sk[high-1]);
        if (Cross(sk[low].b-sk[low].a,mid-sk[low].a)>0) break;
    }
    tp=high-low+1; for (int i=0;i<tp;i++) sk[i]=sk[low+i];
    pols=0; if  (tp<=2) return;
    for (int i=0;i<tp;i++) pol[pols++]=Inter(sk[i],sk[(i+1)%tp]);
}
```

7.4 圆与多边形交集

```
double CT(Circle c,Point a,Point b) { // 圆与三角形交 (多边形)
    double da=Dist(a-c.pt()), db=Dist(b-c.pt());
```

```
    if (da>db) { swap(a,b); swap(da,db); }
    Seg s=Seg(a,b);
    vector<Point> temp=CS(c,s);
    if (F(db-c.r)<=0) return 0.5*abs(Cross(a-c.pt(),b-c.pt()));
    if (F(da-c.r)<0) {
        if (F(Dot(a-temp[1],b-temp[1]))<0) swap(temp[0],temp[1]);
        return Shan(c,Alpha(temp[0]-c.pt(),b-c.pt()))+
            0.5*abs(Cross(a-c.pt(),temp[0]-c.pt()));
    }
    if (!temp.size()) return Shan(c,Alpha(a-c.pt(),b-c.pt()));
    if (Ins(temp[1],s)&&Dist2(a-temp[1])<Dist2(a-temp[0])) swap(temp[0],temp[1]);
    if (Ins(temp[0],s)&&Ins(temp[1],s)) {
        return Shan(c,Alpha(a-c.pt(),temp[0]-c.pt()))+
            Shan(c,Alpha(b-c.pt(),temp[1]-c.pt()))+
            0.5*abs(Cross(temp[0]-c.pt(),temp[1]-c.pt()));
    }
    return Shan(c,Alpha(a-c.pt(),b-c.pt()));
}
```

7.5 三角形面积并

```
#define pr pair<ld,ld>
typedef long double ld;
const ld EPS=1e-8, INF=1e100;
struct Point {
    ld x,y; Point(){} Point(ld _,ld __):x(_),y(__){}
    void read() { double _x,_y; scanf("%lf%lf",&_x,&_y); x=_x,y=_y; }
    friend bool operator<(Point a,Point b) {
        if(fabs(a.x-b.x)<EPS) return a.y<b.y;
        return a.x<b.x;
    }
    friend Point operator +(Point a,Point b) { return Point(a.x+b.x,a.y+b.y); }
    friend Point operator -(Point a,Point b) { return Point(a.x-b.x,a.y-b.y); }
    friend Point operator *(ld a,Point b) { return Point(a*b.x,a*b.y); }
    friend ld operator *(Point a,Point b) { return a.x*b.x+a.y*b.y; }
    friend ld operator ^(Point a,Point b) { return a.x*b.y-a.y*b.x; }
} a[N][3],Poi[N*N];
struct Line {
    Point p,v; Line(){} Line(Point x,Point y){p=x,v=y-x;}
    Point operator [](int k) { if(k) return p+v; else return p; }
    friend bool Cross(Line a,Line b) {
        return (a.v^b[0]-a.p)*(a.v^b[1]-a.p)<-EPS &&
            (b.v^a[0]-b.p)*(b.v^a[1]-b.p)<-EPS;
    }
    friend Point getP(Line a,Line b) {
        Point u=a.p-b.p; ld temp=(b.v^u)/(a.v^b.v);
        return a.p+temp*a.v;
    }
}l[N][3],T;
```

```
pr p[N];
int main() {
    int n,m,i,j,k,x,y,cnt,tot;
    ld ans,last,A,B,sum;
    scanf("%d",&n);
    for(i=1,tot=0;i<=n;i++) {
        a[i][0].read(),a[i][1].read(),a[i][2].read();
        Poi[++tot]=a[i][0],Poi[++tot]=a[i][1],Poi[++tot]=a[i][2];
        sort(a[i],a[i]+3);
        if((a[i][2]-a[i][0]^a[i][1]-a[i][0])>EPS)
            l[i][0]=Line(a[i][0],a[i][2]),l[i][1]=Line(a[i][2],a[i][1]),l[i][2]=Line(a[i][1],a[i][0]);
        else
            l[i][0]=Line(a[i][2],a[i][0]),l[i][1]=Line(a[i][1],a[i][2]),l[i][2]=Line(a[i][0],a[i][1]);
    }
    for(i=1;i<=n;i++) for(j=1;j<i;j++) for(x=0;x<3;x++) for(y=0;y<3;y++)
        if(Cross(l[i][x],l[j][y])) Poi[++tot]=getP(l[i][x],l[j][y]);
    sort(Poi+1,Poi+tot+1);
    ans=0,last=Poi[1].x; T=Line(Point(0,-INF),Point(0,INF));
    for(i=2;i<=tot;i++) {
        T.p.x=(last+Poi[i].x)/2;
        for(j=1,cnt=0;j<=n;j++)
            if(Cross(l[j][0],T)) {
                if(Cross(l[j][1],T)) B=getP(l[j][1],T).y;
                else B=getP(l[j][2],T).y;
                A=getP(l[j][0],T).y; if (A>B) swap(A,B);
                p[++cnt]=pr(A,B);
            }
        sort(p+1,p+cnt+1);
        for(j=1,sum=0,A=-INF;j<=cnt;j++) {
            if(p[j].first>A) sum+=p[j].second-p[j].first, A=p[j].second;
            else if(p[j].second>A) sum+=p[j].second-A, A=p[j].second;
        }
        ans+=(Poi[i].x-last)*sum; last=Poi[i].x;
    }
    printf("%.2lf\n",(double)ans);
}
```

7.6 K 圆并

```
#define sqr(x) ((x)*(x))
const double eps = 1e-8;
double area[N]; int n;
int dcmp(double x) { if (x < -eps) return -1; else return x > eps; }
struct cp { double x, y, r, angle; int d;
    cp(){} cp(double xx, double yy, double ang = 0, int t = 0) {
        x = xx;  y = yy;  angle = ang;  d = t; }
    void get() { scanf("%lf%lf%lf", &x, &y, &r); d = 1; }
```

```
}cir[N], tp[N * 2];
double dis(cp a, cp b) { return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y)); }
double cross(cp p0, cp p1, cp p2) {
    return (p1.x - p0.x) * (p2.y - p0.y) - (p1.y - p0.y) * (p2.x - p0.x);
}
int CirCrossCir(cp p1, double r1, cp p2, double r2, cp &cp1, cp &cp2) {
    double mx = p2.x - p1.x, sx = p2.x + p1.x, mx2 = mx * mx;
    double my = p2.y - p1.y, sy = p2.y + p1.y, my2 = my * my;
    double sq = mx2 + my2, d = -(sq - sqr(r1 - r2)) * (sq - sqr(r1 + r2));
    if (d + eps < 0) return 0; if (d < eps) d = 0; else d = sqrt(d);
    double x = mx * ((r1 + r2) * (r1 - r2) + mx * sx) + sx * my2;
    double y = my * ((r1 + r2) * (r1 - r2) + my * sy) + sy * mx2;
    double dx = mx * d, dy = my * d; sq *= 2;
    cp1.x = (x - dy) / sq; cp1.y = (y + dx) / sq;
    cp2.x = (x + dy) / sq; cp2.y = (y - dx) / sq;
    if (d > eps) return 2; else return 1;
}
bool circmp(const cp& u, const cp& v) { return dcmp(u.r - v.r) < 0; }
bool cmp(const cp& u, const cp& v) {
    if (dcmp(u.angle - v.angle)) return u.angle < v.angle;
    return u.d > v.d;
}
double calc(cp cir, cp cp1, cp cp2) {
    double ans = (cp2.angle - cp1.angle) * sqr(cir.r)
        - cross(cir, cp1, cp2) + cross(cp(0, 0), cp1, cp2);
    return ans / 2;
}
void CirUnion(cp cir[], int n) {
    cp cp1, cp2; sort(cir, cir + n, circmp);
    for (int i = 0; i < n; ++i) for (int j = i + 1; j < n; ++j)
        if (dcmp(dis(cir[i], cir[j]) + cir[i].r - cir[j].r) <= 0) cir[i].d++;
    for (int i = 0; i < n; ++i) {
        int tn = 0, cnt = 0;
        for (int j = 0; j < n; ++j) {
            if (i == j) continue;
            if (CirCrossCir(cir[i],cir[i].r,cir[j],cir[j].r,cp2,cp1)<2) continue;
            cp1.angle = atan2(cp1.y - cir[i].y, cp1.x - cir[i].x);
            cp2.angle = atan2(cp2.y - cir[i].y, cp2.x - cir[i].x);
            cp1.d = 1; tp[tn++] = cp1; cp2.d = -1; tp[tn++] = cp2;
            if (dcmp(cp1.angle - cp2.angle) > 0) cnt++;
        }
        tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, pi, -cnt);
        tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, -pi, cnt);
        sort(tp, tp + tn, cmp);
        int p, s = cir[i].d + tp[0].d;
        for (int j = 1; j < tn; ++j) {
            p = s;  s += tp[j].d;
            area[p] += calc(cir[i], tp[j - 1], tp[j]);
        }
    }
```

```
}
void solve() {
    for (int i = 0; i < n; ++i) cir[i].get();
    memset(area, 0, sizeof(area));
    CirUnion(cir, n);
    for (int i = 1; i <= n; ++i) {
        area[i] -= area[i + 1];
        printf("[%d] = %.3lf\n", i, area[i]);
    }
}
```

7.7 三维计算几何

```
Point Cross(Point a,Point b) {
    return Point(a.y*b.z-a.z*b.y,a.z*b.x-a.x*b.z,a.x*b.y-a.y*b.x); }
double Crossxy(Point a,Point b) { return a.x*b.y-a.y*b.x; }
vector<Point> SegPlane(Seg seg,Plane p) {
    vector<Point> ans; ans.clear();
    Point fa=Cross(p.b-p.a,p.c-p.a);
    if  (F(Dot(fa,seg.b-seg.a))==0) return ans;
    double s=Dot(p.a-seg.a,fa)/Dist(fa), t=Dot(p.a-seg.b,fa)/Dist(fa);
    ans.push_back(seg.a+(seg.b-seg.a)*s/(s-t));
    return ans;
}
// mixed product
double Mix(Point3 a,Point3 b,Point3 c) { return Dot(Cross(a,b),c); }
double PP(Point3 pt,Plane pl) { // distance from point to plane
    Point3 fa=Cross(pl.b-pl.a,pl.c-pl.a);
    return abs(Dot(fa,pt-pl.a))/Dist(fa);
}
// get the center point from 3D(need plane well prepared)
Point3 Getcenter(Point3 p[],int n,Plane pp[],int nn) {
    double sumv=0;
    Point3 sum=Point3(0,0,0);
    for (int i=0;i<nn;i++)
    {
        double tempv=Mix(pp[i].b-pp[i].a,pp[i].c-pp[i].a,Point3(0,0,0)-pp[i].a);
        sum=sum+(pp[i].a+pp[i].b+pp[i].c)*tempv/4.0;
        sumv+=tempv;
    }
    return sum/sumv;
}
```

8 黑科技和杂项

8.1 读入优化

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)

```cpp
#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;
}
```

9 Vim

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

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

10 Java