# CSUST-acm 模板(2015 年 11 月 8 日)

----By:qwb

# 数据结构

1.BIT

```
int S[MX], A[MX], n;
/*返回的是 1~x 的和*/
int query(int x) {
    int ret = 0;
    for(; x; x -= x & -x) {
        ret += S[x];
    }
    return ret;
}

void update(int x, int d) {
    for(; x <= n; x += x & -x) {
        S[x] += d;
    }
}
```

2.ST 表

```
/*可以从 0 也可以从 1 开始,30 应该能使用 100W 以内的*/
int A[MX];
int MIN[MX][30], MAX[MX][30];

void RMQ_init(int n) {
    for(int i = 0; i < n + 1; i++) {
        MAX[i][0] = MIN[i][0] = A[i];
    }
    for(int j = 1; (1 << j) <= n + 1; j++) {
        for(int i = 0; i + (1 << j) - 1 < n + 1; i++) {
            MAX[i][j] = max(MAX[i][j - 1], MAX[i + (1 << (j - 1))][j - 1]);
            MIN[i][j] = min(MIN[i][j - 1], MIN[i + (1 << (j - 1))][j - 1]);
        }
    }
}

int RMQ_min(int L, int R) {
    int k = 0;
    while((1 << (k + 1)) <= R - L + 1) k++;
    return min(MIN[L][k], MIN[R - (1 << k) + 1][k]);
}

int RMQ_max(int L, int R) {
    int k = 0;
    while((1 << (k + 1)) <= R - L + 1) k++;
    return max(MAX[L][k], MAX[R - (1 << k) + 1][k]);
}
```

3.并查集

```
int P[MX], Rank[MX];

int find(int x) {
    return P[x] == x ? x : (P[x] = find(P[x]));
}

void Union(int u, int v) {
    int x = find(u), y = find(v);
    if(x == y) return;

    if(Rank[x] < Rank[y]) {
        P[x] = y;
    } else {
        P[y] = x;
        if(Rank[x] == Rank[y]) Rank[x]++;
    }
}

void find_init(int n) {
    memset(Rank, 0, sizeof(Rank));
    for(int i = 1; i <= n; i++) P[i] = i;
}
```

4.字符串哈希

```
const int MX = 2e4 + 5;
const int HS = 1000007;

char word[MX][35];
int H_head[HS], H_next[MX], H_rear;

int Hash(char*S) {
    int len = strlen(S), ret = 0;
    for(int i = 0; i < len; i++) {
        ret = ret * 131 + S[i];
    }
    return (ret & 0x7fffffff) % HS;
}

void Hash_init() {
    H_rear = 0;
    memset(H_head, -1, sizeof(H_head));
    memset(H_next, -1, sizeof(H_next));
}

/*返回-1 表示不存在*/
int Hash_query(char*S) {
    int h = Hash(S);

    for(int i = H_head[h]; ~i; i = H_next[i]) {
        if(strcmp(word[i], S) == 0) return i;
    }
    return -1;
}

/*返回添加的下标,从 0 开始*/
int Hash_add(char*S) {
    int h = Hash(S);
```

```
    for(int i = H_head[h]; ~i; i = H_next[i]) {
        if(strcmp(word[i], S) == 0) return i;
    }

    strcpy(word[H_rear], S);
    H_next[H_rear] = H_head[h];
    H_head[h] = H_rear;
    return H_rear++;
}
```

5.长整数哈希

```
const int MX = 2e4 + 5;
const int HS = 1000007;

LL H_T[MX];
int H_head[HS], H_next[MX], H_rear;

int Hash(LL S) {
    return S % HS;
}

void Hash_init() {
    H_rear = 0;
    memset(H_head, -1, sizeof(H_head));
    memset(H_next, -1, sizeof(H_next));
}

int Hash_add(LL S) {
    int h = Hash(S);

    for(int i = H_head[h]; ~i; i = H_next[i]) {
        if(H_T[i] == S) return i;
    }

    H_T[H_rear] = S;
    H_next[H_rear] = H_head[h];
    H_head[h] = H_rear;
    return H_rear++;
}

int Hash_query(LL S) {
    int h = Hash(S);

    for(int i = H_head[h]; ~i; i = H_next[i]) {
        if(H_T[i] == S) return i;
    }
    return -1;
}
```

6.面积并

```
const int MX = 1e3 + 5;
#define lson l,m,rt<<1
#define rson m+1,r,rt<<1|1
#define root 1,rear,1

int rear, cnt[MX << 2];
double S[MX << 2], A[MX];
```

```
struct Que {
    int sign;
    double top, L, R;
    bool operator<(const Que &b)const {
        return top < b.top;
    }
    Que() {}
    Que(double _top, double _L, double _R, int _sign) {
        top = _top; L = _L; R = _R; sign = _sign;
    }
} Q[MX];

int BS(double x) {
    int l = 1, r = rear, m;
    while(l <= r) {
        m = (l + r) >> 1;
        if(A[m] == x) return m;
        if(A[m] < x)  l = m + 1;
        else       r = m - 1;
    }
    return -1;
}

/*每次只需要改 push_up 即可*/
void push_up(int l, int r, int rt) {
    if(cnt[rt]) S[rt] = A[r + 1] - A[l];
    else S[rt] = S[rt << 1] + S[rt << 1 | 1];
}

void update(int L, int R, int d, int l, int r, int rt) {
    if(L <= l && r <= R) {
        cnt[rt] += d;
        push_up(l, r, rt);
        return;
    }

    int m = (l + r) >> 1;
    if(L <= m) update(L, R, d, lson);
    if(R > m) update(L, R, d, rson);
    push_up(l, r, rt);
}

int main() {
    int n, ansk = 0;
    while(~scanf("%d", &n), n) {
        rear = 0;
        memset(cnt, 0, sizeof(cnt));
        memset(S, 0, sizeof(S));

        for(int i = 1; i <= n; i++) {
            double x1, y1, x2, y2;
            scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);

            A[++rear] = x1, A[++rear] = x2;
            Q[i] = Que(y1, x1, x2, 1);
            Q[i + n] = Que(y2, x1, x2, -1);
        }
        sort(Q + 1, Q + 1 + 2 * n);
```

```
        sort(A + 1, A + 1 + rear);
        rear = unique(A + 1, A + 1 + rear) - A - 1;

        double last = 0, ans = 0;
        for(int i = 1; i <= 2 * n; i++) {
            ans += (Q[i].top - last) * S[1];
            update(BS(Q[i].L), BS(Q[i].R) - 1, Q[i].sign, root);
            last = Q[i].top;
        }
        printf("Test case #%d\n", ++ansk);
        printf("Total explored area: %.2lf\n\n", ans);
    }
    return 0;
}
```

7.莫队算法

```
const int MX = 1e5 + 5;
const int MQ = 1e5 + 5;

LL ans[MQ];
int n, unit, Qt;
int num[MX], A[MX];

struct Que {
    int L, R, id;
    bool operator<(const Que &b)const {
        if(L / unit == b.L / unit) {
            if(R == b.R) return L < b.L;
            return R < b.R;
        }
        return L / unit < b.L / unit;
    }
} Q[MQ];

/*Qt 表示有多少次查询，下标从 1 开始,最后答案保存在 ans 里*/
void solve(int Qt) {
    unit = sqrt(n + 0.5);
    memset(num, 0, sizeof(num));
    sort(Q + 1, Q + 1 + Qt);

    LL sum = 0;
    int L = 1, R = 0, c = 1;
    while(c <= Qt) {
        while(Q[c].L < L) {
            LL s = num[A[--L]]++;
            sum += -s * s * s + (s + 1) * (s + 1) * (s + 1);
        }
        while(Q[c].R > R) {
            LL s = num[A[++R]]++;
            sum += -s * s * s + (s + 1) * (s + 1) * (s + 1);
        }
        while(Q[c].L > L) {
            LL s = num[A[L++]]--;
            sum += -s * s * s + (s - 1) * (s - 1) * (s - 1);
        }
        while(Q[c].R < R) {
            LL s = num[A[R--]]--;
            sum += -s * s * s + (s - 1) * (s - 1) * (s - 1);
```

```
        }
        ans[Q[c++].id] = sum;
    }
}
```

8.离线操作求第 k 大可单点修改

```
/*
测试样例：
5
1 2 3 4 5
3
2 2 4 2
1 3 6
2 2 4 2

输出
3
4
*/
const int MX = 1e5 + 5;

int nq;
int n, m, val[MX];
int L[MX << 2], R[MX << 2];
int ty[MX * 3], ql[MX * 3], qr[MX * 3], qk[MX * 3], rep[MX * 3];
vector<int>dp[MX << 2], V;
int cur, rear, via[MX], a[MX];

int getid(int x) {
    return lower_bound(V.begin(), V.end(), x) - V.begin();
}

inline int low(int x) {
    return x & (-x);
}

int ask(int loc) {
    int ret = 0;
    for(int i = loc; i >= 1; i -= low(i))if(via[i] == cur)ret += a[i];
    return ret;
}

void upd(int loc, int w) {
    for(int i = loc; i <= n; i += low(i)) {
        if(via[i] < cur) {
            via[i] = cur;
            a[i] = w;
        } else a[i] += w;
    }
}

int main() {
    while(~scanf("%d", &n)) {
        dp[0].clear();
        V.clear();
        nq = 0;
        for(int i = 1; i <= n; i++) {
            scanf("%d", &qr[nq]);
```

```
    val[i] = qr[nq];
    V.push_back(val[i]);
    ql[nq] = i;
    ty[nq] = 1;
    dp[0].push_back(nq++);
}
scanf("%d", &m);
for(int i = 0; i < m; i++) {
    scanf("%d%d%d", &ty[nq], &ql[nq], &qr[nq]);
    if(ty[nq] == 2) scanf("%d", &qk[nq]);
    else {
        V.push_back(qr[nq]);
        dp[0].push_back(nq++);
        ty[nq] = -1;
        ql[nq] = ql[nq - 1];
        qr[nq] = val[ql[nq]];
        val[ql[nq]] = qr[nq - 1];
    }
    dp[0].push_back(nq++);
}
sort(V.begin(), V.end());
V.erase(unique(V.begin(), V.end()), V.end());

for(int i = 0; i < nq; i++)if(ty[i] != 2)qr[i] = getid(qr[i]);
for(int i = 1; i <= n; i++)a[i] = 0, via[i] = 0;
L[0] = 0, R[0] = V.size() - 1;

cur = 0, rear = 1;
while(cur < rear) {
    if(L[cur] == R[cur]) {
        for(int i = 0; i < dp[cur].size(); i++) {
            int u = dp[cur][i];
            if(ty[u] == 2)rep[u] = V[L[cur]];
        }
    } else {
        int mid = (L[cur] + R[cur]) >> 1;
        L[rear] = L[cur], R[rear] = mid;
        dp[rear].clear();

        int ls = rear++;
        L[rear] = mid + 1, R[rear] = R[cur];
        dp[rear].clear();

        int rs = rear++;
        for(int i = 0; i < dp[cur].size(); i++) {
            int u = dp[cur][i];
            if(ty[u] == 2) {
                int t = ask(qr[u]) - ask(ql[u] - 1);
                if(t >= qk[u])dp[ls].push_back(u);
                else qk[u] -= t, dp[rs].push_back(u);
            } else {
                if(qr[u] <= mid)dp[ls].push_back(u), upd(ql[u], ty[u]);
                else dp[rs].push_back(u);
            }
        }
    }
    cur++;
}
```

```
        for(int i = 0; i < nq; i++){
            if(ty[i] == 2) printf("%d\n", rep[i]);
        }
    }
    return 0;
}
```

9.主席树

```
const int MX = 1e5 + 5;

int n, m, tot, num;
int T[MX], a[MX], b[MX * 2];
int lson[MX * 40], rson[MX * 40], c[MX * 40];

int build(int l, int r) {
    int root = tot++;
    c[root] = 0;
    if(l != r) {
        int mid = (l + r) >> 1;
        lson[root] = build(l, mid);
        rson[root] = build(mid + 1, r);
    }
    return root;
}
void hase(int now) {
    sort(b, b + now);
    num = unique(b, b + now) - b;
}
int get_hase(int now) {
    return (lower_bound(b, b + num, now) - b);
}
int update(int root, int pos, int val) {
    int newroot = tot++;
    int temp = newroot;
    c[newroot] = c[root] + val;
    int l = 0, r = num - 1;
    while(l < r) {
        int mid = (l + r) >> 1;
        if(pos <= mid) {
            lson[newroot] = tot++;
            rson[newroot] = rson[root];
            newroot = lson[newroot];
            root = lson[root];
            r = mid;
        } else {
            lson[newroot] = lson[root];
            rson[newroot] = tot++;
            newroot  = rson[newroot];
            root = rson[root];
            l = mid + 1;
        }
        c[newroot] = c[root] + val;
    }
    return temp;
}
int query(int lroot, int rroot, int k) {
    int l = 0, r = num - 1;
    while(l < r) {
```

```
        int mid = (l + r) >> 1;
        int sum = c[lson[rroot]] - c[lson[lroot]];
        if(sum >= k) {
            r = mid;
            lroot = lson[lroot];
            rroot = lson[rroot];
        } else {
            k -= sum;
            l = mid + 1;
            lroot = rson[lroot];
            rroot = rson[rroot];
        }
    }
    return l;
}
struct node {
    int l, r, k;
} Q[MX];
int main() {//FIN;
    while(~scanf("%d%d", &n, &m)) {
        int i;
        num = 0;
        tot = 0;
        for(i = 0; i < n; i ++) {
            scanf("%d", &a[i]);
            b[num++] = a[i];
        }
        for(i = 0; i < m; i++) {
            scanf("%d%d%d", &Q[i].l, &Q[i].r, &Q[i].k);
        }
        hase(num);
        T[0] = build(0, num - 1);
        for(i = 0; i < n; i++) {
            T[i + 1] = update(T[i], get_hase(a[i]), 1);
        }
        for(i = 0; i < m; i++) {
            printf("%d\n", b[query(T[Q[i].l - 1], T[Q[i].r], Q[i].k)]);
        }
    }
    return 0;
}
```

10.DLX 精确覆盖

```
const int MX = 1000 + 5;
const int MN = 1000000 + 5;
const int INF = 0x3f3f3f3f;

int ans[MX][MX];

struct DLX {
    int m, n;
    int H[MX], S[MX];
    int Row[MN], Col[MN], rear;
    int L[MN], R[MN], U[MN], D[MN];

    void Init(int _m, int _n) {
        m = _m; n = _n;
        rear = n;
```

```
    for(int i = 0; i <= n; i++) {
        S[i] = 0;
        L[i] = i - 1;
        R[i] = i + 1;
        U[i] = D[i] = i;
    }
    L[0] = n; R[n] = 0;
    for(int i = 1; i <= m; i++) {
        H[i] = -1;
    }
}

void Link(int r, int c) {
    int rt = ++rear;
    Row[rt] = r; Col[rt] = c; S[c]++;

    D[rt] = D[c]; U[D[c]] = rt;
    U[rt] = c; D[c] = rt;
    if(H[r] == -1) {
        H[r] = L[rt] = R[rt] = rt;
    } else {
        int id = H[r];
        R[rt] = R[id]; L[R[id]] = rt;
        L[rt] = id; R[id] = rt;
    }
}

void Remove(int c) {
    R[L[c]] = R[c]; L[R[c]] = L[c];
    for(int i = D[c]; i != c; i = D[i]) {
        for(int j = R[i]; j != i; j = R[j]) {
            D[U[j]] = D[j]; U[D[j]] = U[j];
            S[Col[j]]--;
        }
    }
}

void Resume(int c) {
    for(int i = U[c]; i != c; i = U[i]) {
        for(int j = L[i]; j != i; j = L[j]) {
            D[U[j]] = U[D[j]] = j;
            S[Col[j]]++;
        }
    }
    R[L[c]] = L[R[c]] = c;
}

bool Dance(int cnt) {
    if(R[0] == 0) return true;

    int c = R[0];
    for(int i = R[0]; i != 0; i = R[i]) {
        if(S[i] < S[c]) c = i;
    }

    Remove(c);
    for(int i = D[c]; i != c; i = D[i]) {
        for(int j = R[i]; j != i; j = R[j]) Remove(Col[j]);
```

```
            int r = Row[i];
            ans[(r - 1) / 81 + 1][((r - 1) % 81) / 9 + 1] = ((r - 1) % 81) % 9 + 1;
            if(Dance(cnt + 1)) return true;

            for(int j = L[i]; j != i; j = L[j]) Resume(Col[j]);
        }
        Resume(c);
        return false;
    }
} G;

int S[MX][MX], vis[10];

void check(int x, int y) {
    for(int i = 1; i <= 9; i++) vis[i] = 0;
    for(int i = 1; i <= 9; i++) {
        vis[S[i][y]] = vis[S[x][i]] = 1;
    }
    int tx = (x - 1) / 3 + 1, ty = (y - 1) / 3 + 1;
    for(int i = (tx - 1) * 3 + 1; i <= tx * 3; i++) {
        for(int j = (ty - 1) * 3 + 1; j <= ty * 3; j++) {
            vis[S[i][j]] = 1;
        }
    }
}

int main() {
    int T; //FIN;
    scanf("%d", &T);
    while(T--) {
        G.Init(9 * 9 * 9, 4 * 9 * 9);
        for(int i = 1; i <= 9; i++) {
            for(int j = 1; j <= 9; j++) {
                scanf("%1d", &S[i][j]);
            }
        }

        for(int i = 1; i <= 9; i++) {
            for(int j = 1; j <= 9; j++) {
                int tx = (i - 1) / 3 + 1, ty = (j - 1) / 3 + 1, tp = (tx - 1) * 3 + ty;

                if(!S[i][j]) {
                    check(i, j);
                    for(int k = 1; k <= 9; k++) {
                        if(vis[k]) continue;

                        int id = ((i - 1) * 9 + j - 1) * 9 + k;
                        G.Link(id, (i - 1) * 9 + j);
                        G.Link(id, 9 * 9 + (i - 1) * 9 + k);
                        G.Link(id, 2 * 9 * 9 + (j - 1) * 9 + k);
                        G.Link(id, 3 * 9 * 9 + (tp - 1) * 9 + k);
                    }
                } else {
                    int k = S[i][j];
                    int id = ((i - 1) * 9 + j - 1) * 9 + k;
                    G.Link(id, (i - 1) * 9 + j);
                    G.Link(id, 9 * 9 + (i - 1) * 9 + k);
```

```
                G.Link(id, 2 * 9 * 9 + (j - 1) * 9 + k);
                G.Link(id, 3 * 9 * 9 + (tp - 1) * 9 + k);
            }
        }
    }
    int ret = G.Dance(0);
    for(int i = 1; i <= 9; i++) {
        for(int j = 1; j <= 9; j++) {
            if(S[i][j]) printf("%d", S[i][j]);
            else printf("%d", ans[i][j]);
        }
        printf("\n");
    }

}
return 0;
}
```

11.DLX 重复覆盖

```
const int MX = 300 + 5;
const int MN = 90000 + 5;
const int INF = 0x3f3f3f3f;

struct DLX {
    int m, n, ans;
    int H[MX], S[MX], vis[MX];
    int Row[MN], Col[MN], rear;
    int L[MN], R[MN], U[MN], D[MN];

    void Init(int _m, int _n) {
        m = _m; n = _n;
        rear = n; ans = INF;
        for(int i = 0; i <= n; i++) {
            S[i] = 0;
            L[i] = i - 1;
            R[i] = i + 1;
            U[i] = D[i] = i;
        }
        L[0] = n; R[n] = 0;
        for(int i = 1; i <= m; i++) {
            H[i] = -1;
        }
    }

    void Link(int r, int c) {
        int rt = ++rear;
        Row[rt] = r; Col[rt] = c; S[c]++;

        D[rt] = D[c]; U[D[c]] = rt;
        U[rt] = c; D[c] = rt;
        if(H[r] == -1) {
            H[r] = L[rt] = R[rt] = rt;
        } else {
            int id = H[r];
            R[rt] = R[id]; L[R[id]] = rt;
            L[rt] = id; R[id] = rt;
        }
    }
```

```
    void Remove(int c) {
        for(int i = D[c]; i != c; i = D[i]) {
            R[L[i]] = R[i]; L[R[i]] = L[i];
        }
    }

    void Resume(int c) {
        for(int i = U[c]; i != c; i = U[i]) {
            R[L[i]] = L[R[i]] = i;
        }
    }

    int h() {
        int ret = 0;
        memset(vis, 0, sizeof(vis));
        for(int c = R[0]; c != 0; c = R[c]) {
            if(!vis[c]) {
                ret++;
                vis[c] = 1;
                for(int i = D[c]; i != c; i = D[i]) {
                    for(int j = R[i]; j != i; j = R[j]) {
                        vis[Col[j]] = 1;
                    }
                }
            }
        }
        return ret;
    }

    void Dance(int cnt) {
        if(cnt + h() >= ans) return;
        if(R[0] == 0) {
            ans = min(ans, cnt);
            return;
        }

        int c = R[0];
        for(int i = R[0]; i != 0; i = R[i]) {
            if(S[i] < S[c]) c = i;
        }

        for(int i = D[c]; i != c; i = D[i]) {
            Remove(i);
            for(int j = R[i]; j != i; j = R[j]) Remove(j);
            Dance(cnt + 1);
            for(int j = L[i]; j != i; j = L[j]) Resume(j);
            Resume(i);
        }
    }
} G;
```

12.单调队列

```
const int MX = 1e6 + 5;
const int INF = 0x3f3f3f3f;

int A[MX];
int Q_1[MX], cur_1, tail_1;//Min
```

```
int Q_2[MX], cur_2, tail_2;//Max
int MIN[MX], MAX[MX];

=
int main() {
    int n, k; //FIN;
    while(~scanf("%d%d", &n, &k)) {
        k = min(n, k);
        cur_1 = tail_1 = cur_2 = tail_2 = 0;
        for(int i = 1; i <= n; i++) {
            A[i] = read();
        }

        for(int i = 1; i <= n; i++) {
            while(cur_1 < tail_1 && A[Q_1[tail_1 - 1]] > A[i]) tail_1--; Q_1[tail_1++] = i;
            while(cur_2 < tail_2 && A[Q_2[tail_2 - 1]] < A[i]) tail_2--; Q_2[tail_2++] = i;
            if(i >= k) {
                while(cur_1 < tail_1 && Q_1[cur_1] < i - k + 1) cur_1++;
                while(cur_2 < tail_2 && Q_2[cur_2] < i - k + 1) cur_2++;
                MIN[i - k + 1] = A[Q_1[cur_1]];
                MAX[i - k + 1] = A[Q_2[cur_2]];
            }
        }

        for(int i = 1; i <= n - k + 1; i++) {
            printf("%d%c", MIN[i], i == n - k + 1 ? '\n' : ' ');
        }
        for(int i = 1; i <= n - k + 1; i++) {
            printf("%d%c", MAX[i], i == n - k + 1 ? '\n' : ' ');
        }
    }
    return 0;
}
```

13.左偏树

```
/*复杂度
取最小 O(1)
合并 O(logn)
这个是最小堆，求最大堆改 merge 即可
*/
const int MX = 1000 + 5;

struct Data {
    int l, r, key, dist;
} D[MX << 1];
int rear;

int lt_init() {
    rear = 0;
    D[0].dist = -1;
}
int lt_new(int _key = 0) {
    rear++;
    D[rear].l = D[rear].r = 0;
    D[rear].key = _key;
    D[rear].dist = 0;
    return rear;
}
```

```
int lt_merge(int r1, int r2) {
    if(!r1) return r2;
    if(!r2) return r1;
    if(D[r1].key > D[r2].key) {
        swap(r1, r2);
    }
    D[r1].r = lt_merge(D[r1].r, r2);
    if(D[D[r1].l].dist < D[D[r1].r].dist) {
        swap(D[r1].l, D[r1].r);
    }
    D[r1].dist = D[D[r1].r].dist + 1;
    return r1;
}
int lt_pop(int &rt) {
    int ret = D[rt].key;
    rt = lt_merge(D[rt].l, D[rt].r);
    return ret;
};
void lt_push(int &rt, int key) {
    rt = lt_merge(rt, lt_new(key));
}

/*使用的时候
lt_init();
int rt=0;
lt_push(rt,1);
*/
```

14.单调区间求最大子矩阵

```
const int MX = 1e5 + 5;

int A[MX], L[MX], R[MX];
int S[MX], rear;

int main() {
    int n; //FIN;
    while(~scanf("%d", &n), n) {
        for(int i = 1; i <= n; i++) {
            scanf("%d", &A[i]);
        }

        rear = 0;
        for(int i = 1; i <= n; i++) {
            L[i] = 1;
            while(rear && A[S[rear - 1]] >= A[i]) L[i] += L[S[--rear]];
            S[rear++] = i;
        }

        rear = 0;
        for(int i = n; i >= 1; i--) {
            R[i] = 1;
            while(rear && A[S[rear - 1]] >= A[i]) R[i] += R[S[--rear]];
            S[rear++] = i;
        }

        LL ans = 0;
        for(int i = 1; i <= n; i++) {
            ans = max(ans, (LL)A[i] * (L[i] + R[i] - 1));
```

```
        }
        printf("%I64d\n", ans);
    }
    return 0;
}
```

## 图论

1.邻接表

```
/*MX 要开 2 倍大小*/
int Head[MX], Next[MX], rear;

struct Edge {
    int u, v,cost;
} E[MX];

void edge_init() {
    rear = 0;
    memset(Head, -1, sizeof(Head));
}

void edge_add(int u, int v, int cost) {
    E[rear].u = u;
    E[rear].v = v;
    E[rear].cost = cost;
    Next[rear] = Head[u];
    Head[u] = rear++;
}
/*遍历的时候*/
for(int id = Head[u]; ~id; id = Next[id]) {
    int v = E[id].v;
}
```

2.spfa 判负环

```
/*如果存在负环，会返回负环中其中一个的节点，否则会返回-1*/
int spfa_dfs(int u) {
    vis[u] = 1;
    for(int i = Head[u]; ~i; i = Next[i]) {
        int v = E[i].v, w = E[i].cost;
        if(d[u] + w < d[v]) {
            d[v] = d[u] + w;
            if(!vis[v]) {
                int ret = spfa_dfs(v);
                if(ret != -1)  return ret;
            } else return v;
        }
    }
    vis[u] = 0;
    return -1;
}
```

3.二分图判定

```
int vis[MX];
bool Bgraph_check(int u) {
    queue<PII>work;
    work.push(PII(u, 0));

    while(!work.empty()) {
```

```
        PII f = work.front();
        work.pop();

        int u = f.first, c = f.second;

        vis[u] = c;
        for(int id = 0; ~id; id = Next[id]) {
            int v = G[u][id];
            if(vis[v] == c) return false;
            if(vis[v] == -1) {
                vis[v] = c ^ 1;
                work.push(PII(v, c ^ 1));
            }
        }
    }
    return true;
}
```

4.求割边和割点

```
int Low[MX], DFN[MX], cut[MX], dfs_clock;

void tarjan_init(){
    dfs_clock = 0;
    memset(DFN, 0, sizeof(DFN));
    memset(cut, 0, sizeof(cut));
}

/*这样写的前提是，必须是没有重边的，。
如果有重边，from 要改成无向图的边的编号，这样就可以避免重边了~*/
int tarjan(int u, int from) {
    Low[u] = DFN[u] = ++dfs_clock;

    int child = 0;
    for(int id = Head[u]; ~id; id = Next[id]) {
        int v = E[id].v;

        if(!DFN[v]) {
            int lowv = tarjan(v, u);
            Low[u] = min(Low[u], lowv);

            if(lowv >= DFN[u]) {
                cut[u] = 1;
            }
            if(lowv > DFN[u]) {
                E[id].sign = 1;
                E[id ^ 1].sign = 1;
            }

            child++;
        } else if(v != from && DFN[v] < DFN[u]) {
            Low[u] = min(Low[u], DFN[v]);
        }
    }

    if(from == -1 && child == 1) cut[u] = 0;
    return Low[u];
}
```

```
void solve(){
    tarjan(1, -1);//如果已经确定是连通图，就这样写
    /* 否则要这样
    for(int i = 1; i <= n; i++) {
        if(!DFN[i]) tarjan(i, -1);
    }*/
}
```

5.Dijistra

```
LL d[MX];
void dijistra(int Begin){
    memset(d, INF, sizeof(d));
    d[Begin] = 0;

    priority_queue<PLI,vector<PLI>,greater<PLI> >work;
    work.push(PLI(0, Begin));

    while(!work.empty()) {
        PLI f = work.top();
        work.pop();

        LL dist = f.first;
        int u = f.second;
        if(d[u] < dist) continue;

        for(int id = Head[u]; ~id; id = Next[id]) {
            int cost = E[id].cost, v = E[id].v;
            if(dist + cost < d[v]) {
                d[v] = dist + cost;
                work.push(PLI(dist + cost, v));
            }
        }
    }
}
```

6.费用流

```
const int MX = 400 + 5;
const int MM = 400 + 5;
const int INF = 0x3f3f3f3f;

struct Edge {
    int to, next, cap, flow, cost;
    Edge() {}
    Edge(int _to, int _next, int _cap, int _flow, int _cost) {
        to = _to; next = _next; cap = _cap; flow = _flow; cost = _cost;
    }
} E[MM];

int Head[MX], tol;
int pre[MX];
int dis[MX];
bool vis[MX];
int N;
void init(int n) {
    tol = 0;
    N = n + 2;
    memset(Head, -1, sizeof(Head));
```

```
}
void edge_add(int u, int v, int cap, int cost) {
    E[tol] = Edge(v, Head[u], cap, 0, cost);
    Head[u] = tol++;

    E[tol] = Edge(u, Head[v], 0, 0, -cost);
    Head[v] = tol++;
}
bool spfa(int s, int t) {
    queue<int>q;
    for (int i = 0; i < N; i++) {
        dis[i] = INF;
        vis[i] = false;
        pre[i] = -1;
    }
    dis[s] = 0;
    vis[s] = true;
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        vis[u] = false;
        for (int i = Head[u]; i != -1; i = E[i].next) {
            int v = E[i].to;
            if (E[i].cap > E[i].flow && dis[v] > dis[u] + E[i].cost) {
                dis[v] = dis[u] + E[i].cost;
                pre[v] = i;
                if (!vis[v]) {
                    vis[v] = true;
                    q.push(v);
                }
            }
        }
    }
    if (pre[t] == -1) return false;
    else return true;
}
//返回的是最大流， cost 存的是最小费用
int minCostMaxflow(int s, int t, int &cost) {
    int flow = 0;
    cost = 0;
    while (spfa(s, t)) {
        int Min = INF;
        for (int i = pre[t]; i != -1; i = pre[E[i ^ 1].to]) {
            if (Min > E[i].cap - E[i].flow)
                Min = E[i].cap - E[i].flow;
        }
        for (int i = pre[t]; i != -1; i = pre[E[i ^ 1].to]) {
            E[i].flow += Min;
            E[i ^ 1].flow -= Min;
            cost += E[i].cost * Min;
        }
        flow += Min;
    }
    return flow;
}
```

7.求树中最长路

```
int solve(int u, int from, int &ans) {
    int Max1 = 0, Max2 = 0;
    for(int id = Head[u]; ~id; id = Next[id]) {
        int v = E[id].v;
        if(v == from) continue;

        int t = solve(v, u, ans) + 1;

        if(t > Max1) {
            Max2 = Max1;
            Max1 = t;
        } else if(t > Max2) Max2 = t;
    }

    ans = max(ans, Max1 + Max2);
    return Max1;
}
```

8.字典序的拓扑排序

```
const int MX = 1e3 + 5;
/*邻接表省略*/
int IN[MX], A[MX], r;

int main() {
    int n, m;
    while(~scanf("%d%d", &n, &m)) {
        r = 0;
        edge_init();
        memset(IN, 0, sizeof(IN));

        for(int i = 1; i <= m; i++) {
            int u, v;
            scanf("%d%d", &u, &v);
            edge_add(u, v);
            IN[v]++;
        }

        priority_queue<int, vector<int>, greater<int> >work;
        for(int i = 1; i <= n; i++) {
            if(!IN[i]) work.push(i);
        }

        while(!work.empty()) {
            int u = work.top();
            work.pop();
            A[++r] = u;

            for(int i = Head[u]; ~i; i = Next[i]) {
                int v = E[i].v;
                IN[v]--;
                if(!IN[v]) work.push(v);
            }
        }

        for(int i = 1; i <= r; i++) {
            printf("%d%c", A[i], i == r ? '\n' : ' ');
        }
    }
```

```
    return 0;
}
```

9.欧拉回路

```
int IN[MX], P[MX], cur;
/*必须是 IN 为奇数，或者全部为偶数，才能构成欧拉回路*/
void Fleury(int u) {
    while(~Head[u]) {
        int id = Head[u], v = E[id].v;

        Head[u] = Next[id];
        E[id ^ 1].sign = true;
        if(!E[id].sign) Fleury(v);
    }
    P[++cur] = u;
}
```

10.二分图匈牙利匹配

```
/*邻接表版复杂度 O(nm)*/
int match[MX], vis[MX];

bool DFS(int u) {
    vis[u] = true;
    for(int i = Head[u]; ~i; i = Next[i]) {
        int v = E[i].v, w = match[v];
        if(w < 0 || !vis[w] && DFS(w)) {
            match[v] = u;
            match[u] = v;
            return true;
        }
    }
    return false;
}

int BM(int n) {
    int ret = 0;
    memset(match, -1, sizeof(match));
    for(int i = 1; i <= n; i++) {
        if(match[i] < 0) {
            memset(vis, 0, sizeof(vis));
            if(DFS(i)) ret++;
        }
    }
    return ret;
}



/*邻接矩阵版复杂度最坏 O(n^3)*/
int match[MX];
bool vis[MX], G[MX][MX];

bool DFS(int n, int u) {
    vis[u] = true;
    for(int i = 1; i <= n; i++) {
        if(!G[u][i]) continue;
        int v = i, w = match[v];
        if(w < 0 || !vis[w] && DFS(n, w)) {
```

```
            match[v] = u;
            match[u] = v;
            return true;
        }
    }
    return false;
}

int BM(int n) {
    int ret = 0;
    memset(match, -1, sizeof(match));
    for(int i = 1; i <= n; i++) {
        if(match[i] < 0) {
            memset(vis, 0, sizeof(vis));
            if(DFS(n, i)) ret++;
        }
    }
    return ret;
}
```

11.离线 LCA

```
const int MQ = 40000 + 5;
const int MX = 80000 + 5;

struct Edge {
    int v, d;
    Edge(int _v, int _d) {
        v = _v; d = _d;
    }
};
struct Que {
    int id, u, v;
    Que() {}
    Que(int _u, int _v, int _id) {
        u = _u; v = _v; id = _id;
    }
} A[MQ];

int D[MX];
struct LCA {
    int n, ans[MQ];//答案按照 id 保存在 ans 中
    int P[MX]; bool vis[MX];

    vector<Edge>E[MX];
    vector<Que>Q[MQ];

    void Init(int _n) {
        n = _n;
        memset(ans, -1, sizeof(ans));
        memset(vis, false, sizeof(vis));
        for(int i = 1; i <= n; i++) {
            E[i].clear();
            Q[i].clear();
            P[i] = i;
        }
    }

    void AddQue(int u, int v, int id) {
```

```
        Q[u].push_back(Que(u, v, id));
        Q[v].push_back(Que(v, u, id));
    }

    void AddEdge(int u, int v, int d) {
        E[u].push_back(Edge(v, d));
        E[v].push_back(Edge(u, d));
    }

    int Find(int x) {
        return P[x] == x ? x : (P[x] = Find(P[x]));
    }

    void Union(int u, int v) {
        int p1 = Find(u), p2 = Find(v);
        P[p1] = p2;
    }

    /*初始 DFS(root,root)*/
    void DFS(int u, int f, int d) {
        D[u] = d;
        for(int i = 0; i < E[u].size(); i++) {
            int v = E[u][i].v, cost = E[u][i].d;
            if(v != f) DFS(v, u, d + cost);
        }

        vis[u] = 1;
        for(int i = 0; i < Q[u].size(); i++) {
            int v = Q[u][i].v, id = Q[u][i].id;
            if(vis[v]) {
                ans[id] = Find(v);
            }
        }
        Union(u, f);
    }
} lca;
```

## 字符串

1.文本全局替换
/*Match 为 true 表示不区分大小写*/

```
string Replace(string s, string a, string b, bool Match = false) {
    string tmp = s;
    if(!Match) {
        transform(tmp.begin(), tmp.end(), tmp.begin(), ::tolower);
        transform(a.begin(), a.end(), a.begin(), ::tolower);
    }

    int pos;
    while(true) {
        if((pos = tmp.find(a)) != -1) {
            tmp.replace(pos, a.length(), b);
            s.replace(pos, a.length(), b);
        } else break;
    }
    return s;
```

```
}
```

2.KMP

```
int Next[MX];
int KMP(char *A, char *B) {
    int m = strlen(A), n = strlen(B);

    Next[0] = 0;
    for(int i = 1; i < n; i++) {
        int k = Next[i - 1];
        while(B[i] != B[k] && k) k = Next[k - 1];
        Next[i] = B[i] == B[k] ? k + 1 : 0;
    }

    int ans = 0, j = 0;
    for(int i = 0; i < m; i++) {
        while(A[i] != B[j] && j) j = Next[j - 1];
        if(A[i] == B[j]) j++;
        if(j == n) ans++;//会有重叠部分
        //if(j == n) ans++, j = 0;//不会有重叠部分
    }
    return ans;
}
```

3.manacher 求最长回文串

```
const int MAX = 110000 + 10;
char s[MAX * 2];//记得要开两倍
int p[MAX * 2];

int manacher(char *s){
    int len = strlen(s), id = 0, ans = 0;
    for(int i = len; i >= 0; i--) {
        s[i + i + 2] = s[i];
        s[i + i + 1] = '#';
    }
    s[0] = '*';//防越界,很重要!!
    for(int i = 2; i < 2 * len + 1; ++i) {
        if(p[id] + id > i) p[i] = min(p[2 * id - i], p[id] + id - i);
        else p[i] = 1;
        while(s[i - p[i]] == s[i + p[i]]) p[i]++;
        if(id + p[id] < i + p[i]) id = i;
        ans = max(ans, p[i] - 1);
    }
    return ans;
}
```

4.字符串同位异构最小字典序表示法

```
int solve(char*s) {
    int i = 0, j = 1, k = 0, t, l = strlen(s);
    while(i < l && j < l && k < l) {
        t = s[(i + k) >= l ? i + k - 1 : i + k] - s[(j + k) >= l ? j + k - l : j + k];
        if(!t) k++;
        else {
            if(t > 0) i = i + k + 1;
            else j = j + k + 1;
            if(i == j) j++;
            k = 0;
```

```
        }
    }
    return min(i, j);
}
```

5.base64 解码

```
int base64_decode(char *A, unsigned char *B) {
    int n = strlen(A), r = 0;
    for(int t = 0; t < n / 4 - 1; t++) {
        int ret = 0;
        for(int j = t * 4; j < (t + 1) * 4; j++) {
            ret = ret << 6 | ID(A[j]);
        }
        B[r++] = ret >> 16 & 255;
        B[r++] = ret >> 8 & 255;
        B[r++] = ret & 255;
    }

    int ret = 0;
    for(int j = n - 4; j <= n - 1; j++) {
        if(A[j] != '=') ret = ret << 6 | ID(A[j]);
    }
    if(A[n - 2] == '=' && A[n - 1] == '=') {
        B[r++] = ret >> 4 & 255;
    } else if(A[n - 1] == '=') {
        B[r++] = ret >> 10 & 255;
        B[r++] = ret >> 2 & 255;
    } else {
        B[r++] = ret >> 16 & 255;
        B[r++] = ret >> 8 & 255;
        B[r++] = ret & 255;
    }
    B[r] = 0;
    return r;
}
```

6.trie 树

```
/*MX 为总长度*/
const int MX = 3e6 + 5;

struct Trie {
    int rear, root;
    int Next[MX][26], End[MX];

    void Init() {
        rear = 0;
        root = New();
    }

    int New() {
        rear++;
        End[rear] = 0;
        for(int i = 0; i < 26; i++) {
            Next[rear][i] = -1;
        }
        return rear;
    }
```

```
    void Add(char *A) {
        int n = strlen(A), now = root;
        for(int i = 0; i < n; i++) {
            int id = A[i] - 'a';
            if(Next[now][id] == -1) {
                Next[now][id] = New();
            }
            now = Next[now][id];
            End[now]++;
        }
    }

    int Query(char *S) {
        int n = strlen(S), now = root, ret = 0;
        for(int i = 0; i < n; i++) {
            now = Next[now][S[i] - 'a'];
            if(now == -1) return 0;
        }
        return End[now];
    }
} trie;
```

7.AC 自动机

```
/*MX 为总长度*/
const int MX = 500000 + 5;

struct AC_machine {
    int rear, root;
    int Next[MX][26], Fail[MX], End[MX];

    void Init() {
        rear = 0;
        root = New();
    }

    int New() {
        rear++;
        End[rear] = 0;
        for(int i = 0; i < 26; i++) {
            Next[rear][i] = -1;
        }
        return rear;
    }

    void Add(char*A) {
        int n = strlen(A), now = root;
        for(int i = 0; i < n; i++) {
            int id = A[i] - 'a';
            if(Next[now][id] == -1) {
                Next[now][id] = New();
            }
            now = Next[now][id];
        }
        End[now]++;
    }

    void Build() {
        queue<int>Q;
```

```
        Fail[root] = root;
        for(int i = 0; i < 26; i++) {
            if(Next[root][i] == -1) {
                Next[root][i] = root;
            } else {
                Fail[Next[root][i]] = root;
                Q.push(Next[root][i]);
            }
        }

        while(!Q.empty()) {
            int u = Q.front();
            Q.pop();

            for(int i = 0; i < 26; i++) {
                if(Next[u][i] == -1) {
                    Next[u][i] = Next[Fail[u]][i];
                } else {
                    Fail[Next[u][i]] = Next[Fail[u]][i];
                    Q.push(Next[u][i]);
                }
            }
        }
    }

    int Query(char *S) {
        int n = strlen(S), now = root, ret = 0;
        for(int i = 0; i < n; i++) {
            now = Next[now][S[i] - 'a'];
            int temp = now;

            while(temp != root) {
                ret += End[temp];
                End[temp] = 0;
                temp = Fail[temp];
            }
        }
        return ret;
    }
} AC;
```

## 动态规划

1.TSP

```
//W 是距离,n 是除了起点以外的数量，0 为原点
int TSP() {
    memset(dp, 0x3f, sizeof(dp));

    for(int S = 0; S <= (1 << n) - 1; S++) {
        for(int i = 1; i <= n; i++) {
            if(S & (1 << (i - 1))) {
                if(S == (1 << (i - 1))) dp[i][S] = W[0][i];
                else for(int j = 1; j <= n; j++) {
                        if(S & (1 << (j - 1)) && j != i) {
                            dp[i][S] = min(dp[i][S], dp[j][S ^ (1 << (i - 1))] + W[j][i]);
                        }
                    }
            }
```

```
        }
    }

    int ret = INF;
    for(int i = 1; i <= n; i++) {
        ret = min(ret, dp[i][(1 << n) - 1] + W[0][i]);
    }

    /*
    若不需要回到起点，只需要全部走完，那么直接这样写
    int ret=INF;
    for(int i=1;i<=n;i++){
        ret=min(ret,dp[i][(1<<n)-1]);
    }
    也就是说不需要加上了那 W[0][i]而已
    */
    return ret;
}
```

2.四边形不等式优化的石子合并

```
int S[MX], dp[MX][MX], K[MX][MX];

int main() {
    int T, n, t;
    scanf("%d", &T);
    while(T--) {
        memset(dp, 0x3f, sizeof(dp));
        scanf("%d", &n);

        S[0] = 0;
        for(int i = 1; i <= n; i++) {
            scanf("%d", &t);
            dp[i][i] = 0;
            K[i][i] = i;
            S[i] = S[i - 1] + t;
        }

        for(int l = 2; l <= n; l++) {
            for(int i = 1; i <= n - l + 1; i++) {
                for(int j = K[i][i + l - 2]; j <= K[i + 1][i + l - 1]; j++) {
                    int temp = dp[i][j] + dp[j + 1][i + l - 1] + S[i + l - 1] - S[i - 1];
                    if(temp < dp[i][i + l - 1]) {
                        dp[i][i + l - 1] = temp;
                        K[i][i + l - 1] = j;
                    }
                }
            }
        }
        printf("%d\n", dp[1][n]);
    }
    return 0;
}
```

## 数论

1.凸包

```
const int MX = 10000 + 5;

struct Node {
```

```
    double x, y;
    bool operator<(const Node&b) const {
        if(x == b.x) return y < b.y;
        return x < b.x;
    }
} P[MX], R[MX];

double cross(Node a, Node b, Node c) {
    return ((b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y - a.y));
}

int convex(int n) {
    int m = 0, k;
    sort(P, P + n);
    for(int i = 0; i < n; i++) {
        while(m > 1 && cross(R[m - 1], P[i], R[m - 2]) <= 0) m--;
        R[m++] = P[i];
    }

    k = m;
    for(int i = n - 2; i >= 0; i--) {
        while(m > k && cross(R[m - 1], P[i], R[m - 2]) <= 0) m--;
        R[m++] = P[i];
    }
    if(n > 1) m--;
    return m;
}

double length(Node a, Node b) {
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}

int main() {
    int n;
    while(~scanf("%d", &n)) {
        for(int i = 0; i < n; i++) {
            scanf("%lf%lf", &P[i].x, &P[i].y);
        }
        if(n == 0 || n == 1) {
            printf("0.00\n");
            continue;
        }

        int rear = convex(n);
        double ans = 0;
        for(int i = 1; i < rear; i++) {
            ans += length(R[i], R[i - 1]);
        }
        ans += length(R[rear - 1], R[0]);
        printf("%.2lf\n", ans);
    }
    return 0;
}
```

2.质数筛

```
/*O(nlogn)*/
int prime[100000], vis[MX], rear = 0;
void init() {
```

```
    rear = 0;
    vis[1] = 1;
    for(int i = 2; i < MX; i++) {
        if(vis[i]) continue;
        prime[++rear] = i;

        if((LL)i * i >= MX) continue;
        for(int j = i * i; j < MX; j += i) {
            vis[j] = 1;
        }
    }
}

/*O(n)*/
int vis[MX], Prime[800000], rear;
void init() { //vis[i]==1 表示不是质数
    rear = 0; vis[1] = 1;
    for(int i = 2; i < MX; i++) {
        if(!vis[i]) Prime[++rear] = i;

        for(int j = 1; j <= rear && i * Prime[j] < MX; j++) {
            vis[i * Prime[j]] = 1;
            if(!i % Prime[j]) break;
        }
    }
}

/*点试法*/
bool is_prime(LL x) {
    for(int i = 2; (LL)i * i <= x; i++) {
        if(!(x % i)) return false;
    }
    return true;
}
```

3.求约数

```
/*可以用 14000KB，140MS 求出 10W 内的所有约数并打表*/
void init() {
    for(int i = 1; i < MX; i++) {
        for(int j = i; j < MX; j += i) {
            G[j].push_back(i);
        }
    }
}

/*可以单个求约数*/
LL n, s[MX];
int e = sqrt(n + 0.5), rear = 0;
for(int i = 1; i <= e; i++) {
    if(n % i) continue;

    s[++rear] = i;
    s[++rear] = n / i;
}
sort(s + 1, s + rear + 1);
```

4.求组合数

```
/*利用递推公式*/
```

```
const int MX=100;
LL C[MX][MX];

C[0][0]=1;
for(int i=1;i<MX;i++){
    C[i][0]=C[i][i]=1;
    for(int j=1;j<i;j++){
        C[i][j]=(C[i-1][j-1]+C[i-1][j])%mod;
    }
}

/*利用费马小定理*/
const int MX = 1000000 + 5;
const int mod = 1e9 + 7;

LL F[MX], invF[MX];

LL power(LL a, LL b) {
    LL ret = 1;
    while(b) {
        if(b & 1) ret = (ret * a) % mod;
        a = (a * a) % mod;
        b >>= 1;
    }
    return ret;
}

void init() {
    F[0] = 1;
    for(int i = 1; i < MX; i++) {
        F[i] = (F[i - 1] * i) % mod;
    }
    invF[MX - 1] = power(F[MX - 1], mod - 2);
    for(int i = MX - 2; i >= 0; i--) {
        invF[i] = invF[i + 1] * (i + 1) % mod;
    }
}

LL C(int n, int m) {
    if(n < 0 || m < 0 || m > n) return 0;
    if(m == 0 || m == n)    return 1;
    return F[n] * invF[n - m] % mod * invF[m] % mod;
}

LL A(int n, int m) {
    if(n < 0 || m < 0 || m > n) return 0;
    return F[n] * invF[n - m] % mod;
}

5.欧拉函数
/*单点求欧拉函数值*/
LL eular(LL n) {
    LL ans = n;
    for(int i = 2; i * i <= n; i++) {
        if(n % i == 0) {
            ans -= ans / i;
            while(n % i == 0) n /= i;
        }
```

```
    }
    if(n > 1) ans -= ans / n;
    return ans;
}
```

6.莫比乌斯数筛法

```
bool vis[MX];
int prime[MX], mu[MX], tot;

void miu_init() {
    memset(vis, 0, sizeof(vis));
    mu[1] = 1; tot = 0;
    for(int i = 2; i < MX; i++) {
        if(!vis[i]) {
            prime[tot++] = i;
            mu[i] = -1;
        }
        for(int j = 0; j < tot; j++) {
            if(i * prime[j] >= MX) break;
            vis[i * prime[j]] = 1;
            if(i % prime[j] == 0) {
                mu[i * prime[j]] = 0;
                break;
            } else {
                mu[i * prime[j]] = -mu[i];
            }
        }
    }
}
```

7.扩展欧几里德

```
/*可以得到 x>=bound 时的 x 和 y，返回 true 表示有解
否则无解，我只想问这个模板无脑调用有木有~
但是不同的题目特判不同，有的地方记得还是特判，比如 a 和 b 的正负和是否为 0~*/
LL exgcd(LL a, LL b, LL &x, LL &y) {
    if(b == 0) {
        x = 1; y = 0;
        return a;
    }
    LL r = exgcd(b, a % b, x, y);
    LL t = y;
    y = x - a / b * y;
    x = t;
    return r;
}
bool solve(LL a, LL b, LL c, LL bound, LL &x, LL &y) {
    LL xx, yy, d = exgcd(a, b, xx, yy);
    if(c % d) return false;

    xx = xx * c / d; yy = yy * c / d;
    LL t = (bound - xx) * d / b;

    x = xx + b / d * t;
    if(x < bound) {
        t++;
        x = xx + b / d * t;
    }
    y = yy - a / d * t;
```

```
    return true;
}
```

8.快速幂和快速乘

```
LL multi(LL a, LL b) {
    LL ret = 0;
    while(b) {
        if(b & 1) ret = (ret + a) % mod;
        a = (a + a) % mod;
        b >>= 1;
    }
    return ret;
}

LL power(LL a, LL b) {
    LL ret = 1;
    while(b) {
        if(b & 1) ret = (ret * a) % mod;
        a = (a * a) % mod;
        b >>= 1;
    }
    return ret;
}
```

9.快速矩阵幂

```
/*矩阵下标从 0 开始*/
const int matMX = 6;
const int MX = 200000 + 5;
const int INF = 0x3f3f3f3f;
const LL mod = 1e18 + 7;

LL power(LL a, LL b) {
    LL ret = 1;
    while(b) {
        if(b & 1) ret = ret * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return ret;
}

struct Mat {
    int m, n;
    LL S[matMX][matMX];
    Mat(int a, int b) {
        m = a;
        n = b;
        memset(S, 0, sizeof(S));
    }
    Mat(int a, int b, LL w[][matMX]) {
        m = a;
        n = b;
        for(int i = 0; i < m; i++) {
            for(int j = 0; j < n; j++) {
                S[i][j] = w[i][j];
            }
        }
    }
```

```
};

Mat mat_mul(Mat A, Mat B) {
    Mat C(A.m, B.n);
    for(int i = 0; i < A.m; i++) {
        for(int j = 0; j < B.n; j++) {
            for(int k = 0; k < A.n; k++) {
                C.S[i][j] = (C.S[i][j] + A.S[i][k] * B.S[k][j]) % mod;
            }
        }
    }
    return C;
}

Mat Blank(int m, int n) {
    Mat ret(m, n);
    for(int i = 0; i < m; i++) {
        ret.S[i][i] = 1;
    }
    return ret;
}

Mat mat_pow(Mat A, LL b) {
    Mat ret = Blank(A.m, A.n);
    while(b) {
        if(b & 1) ret = mat_mul(ret, A);
        A = mat_mul(A, A);
        b >>= 1;
    }
    return ret;
}

int main() {
    int n;
    while(~scanf("%d", &n)) {
        LL table1[][matMX] = {{0, 0, 1, 0}, {0, 0, 0, 1}, {0, 0, 0, 1}, {0, 0, 2, 2}};
        LL table2[][matMX] = {{1}, {2}, {2}, {6}};

        Mat s(4, 4, table1);
        Mat ans(4, 1, table2);

        Mat res = mat_mul(mat_pow(s, n - 1), ans);
        printf("%I64d\n", (res.S[0][0] + res.S[1][0]) % mod);
    }

    return 0;
}
```

10.高斯消元浮点数

```
const int MX = 100 + 5;
const int INF = 0x3f3f3f3f;
typedef double Matrix[MX][MX];

Matrix A, S;

void gauss(Matrix A, int n) {
    int i, j, k, r;
    for(i = 0; i < n; i++) {
```

```
        r = i;
        for(j = i + 1; j < n; j++) {
            if(fabs(A[j][i]) > fabs(A[r][i])) r = j;
        }
        if(r != i) for(j = 0; j <= n; j++) swap(A[r][j], A[i][j]);

        for(k = i + 1; k < n; k++) {
            double f = A[k][i] / A[i][i];
            for(j = i; j <= n; j++) A[k][j] -= f * A[i][j];
        }
    }

    for(i = n - 1; i >= 0; i--) {
        for(j = i + 1; j < n; j++) {
            A[i][n] -= A[j][n] * A[i][j];
        }
        A[i][n] /= A[i][i];
    }
}
```

11.高斯消元 xor

```
int gauss(int equ, int var) {
    int max_r, col, k;
    for(k = 0, col = 0; k < equ && col < var; k++, col++) {
        max_r = k;
        for(int i = k + 1; i < equ; i++) {
            if(A[i][col] > A[max_r][col]) {
                max_r = i;
            }
        }
        if(A[max_r][col] == 0) {
            k--;
            continue;
        }
        if(max_r != k) {
            for(int j = col; j < var + 1; j++) {
                swap(A[k][j], A[max_r][j]);
            }
        }
        for(int i = k + 1; i < equ; i++) {
            if(A[i][col] != 0) {
                for(int j = col; j < var + 1; j++) {
                    A[i][j] ^= A[k][j];
                }
            }
        }
    }
    for(int i = k; i < equ; i++) {
        if(A[i][col] != 0) return -1;
    }
    if(k < var) return var - k;

    for(int i = var - 1; i >= 0; i--) {
        for(int j = i + 1; j < var; j++) {
            A[i][var] ^= (A[i][j] && A[j][var]);
        }
    }
    return 0;
```

```
}
```

12.pell 方程(C#)

```
/*完全平方数无解*/
struct PellAns {
    public BigInteger p, q;
};

struct Node {
    public BigInteger g, h;
};

static PellAns Solve(int _n) {
    PellAns[] s = new PellAns[4];
    Node[] w = new Node[4];
    BigInteger[] a = new BigInteger[4];
    BigInteger n = _n, zero = 0;

    s[0].p = 0; s[0].q = 1;
    s[1].p = 1; s[1].q = 0;
    a[0] = (int)Math.Sqrt((double)_n);
    a[2] = a[0];
    w[1].g = 0; w[1].h = 1;
    while (true) {
        w[2].g = zero - w[1].g + a[2] * w[1].h;
        w[2].h = (n - w[2].g * w[2].g) / w[1].h;
        a[3] = (w[2].g + a[0]) / w[2].h;
        s[2].p = a[2] * s[1].p + s[0].p;
        s[2].q = a[2] * s[1].q + s[0].q;
        if ((s[2].p * s[2].p - n * s[2].q * s[2].q) == 1 && s[2].p > 0 && s[2].q > 0) return s[2];

        w[0] = w[1]; w[1] = w[2];
        a[2] = a[3];
        s[0] = s[1]; s[1] = s[2];
    }
}
```

13.大质数判定

```
/*power 里乘法可能要用快速乘*/
bool Miller_Rabin(LL n) {
    LL u = n - 1, pre, x;
    int i, j, k = 0;
    if(n == 2 || n == 3 || n == 5 || n == 7 || n == 11)  return true;
    if(n == 1 || (!(n % 2)) || (!(n % 3)) || (!(n % 5)) || (!(n % 7)) || (!(n % 11)))   return false;
    for(; !(u & 1); k++, u >>= 1);
    srand(time(NULL));
    for(i = 0; i < 5; i++) {
        x = rand() % (n - 2) + 2;
        x = power(x, u, n);
        pre = x;
        for(j = 0; j < k; j++) {
            x = multi(x, x, n);
            if(x == 1 && pre != 1 && pre != (n - 1))
                return false;
            pre = x;
        }
        if(x != 1)  return false;
    }
```

```
    return true;
}
```

14.简单大数模板

```
struct BigInteger{
    int A[25];
    enum{MOD = 10000};
    BigInteger(){memset(A, 0, sizeof(A)); A[0]=1;}
    void set(int x){memset(A, 0, sizeof(A)); A[0]=1; A[1]=x;}
    void print(){
        printf("%d", A[A[0]]);
        for (int i=A[0]-1; i>0; i--){
            if (A[i]==0){printf("0000"); continue;}
            for (int k=10; k*A[i]<MOD; k*=10) printf("0");
            printf("%d", A[i]);
        }
        printf("\n");
    }
    int& operator [] (int p) {return A[p];}
    const int& operator [] (int p) const {return A[p];}
    BigInteger operator + (const BigInteger& B){
        BigInteger C;
        C[0]=max(A[0], B[0]);
        for (int i=1; i<=C[0]; i++)
            C[i]+=A[i]+B[i], C[i+1]+=C[i]/MOD, C[i]%=MOD;
        if (C[C[0]+1] > 0) C[0]++;
        return C;
    }
    BigInteger operator * (const BigInteger& B){
        BigInteger C;
        C[0]=A[0]+B[0];
        for (int i=1; i<=A[0]; i++)
            for (int j=1; j<=B[0]; j++){
                C[i+j-1]+=A[i]*B[j], C[i+j]+=C[i+j-1]/MOD, C[i+j-1]%=MOD;
            }
        if (C[C[0]] == 0) C[0]--;
        return C;
    }
};
```

15.自适式高斯消元

```
const double exps = 1e-8;
typedef vector<double> vec;
typedef vector<vec> mat;

int dcmp(double x) {
    if(fabs(x) < exps) return 0;
    return x < 0 ? -1 : 1;
}

void guass(mat &A, int m, int n) {
    for(int i = 0; i < m; i++) {
        int pv = i, id;
        for(int j = 0; j <= n; j++) {
            for(int k = i + 1; k < m; k++) {
                if(fabs(A[k][j]) > fabs(A[pv][j])) {
                    pv = k;
                }
```

```
            }
            if(dcmp(A[pv][j])) break;
        }
        swap(A[i], A[pv]);

        for(id = 0; id <= n && !dcmp(A[i][id]); id++);
        if(id > n) return;

        for(int j = i + 1; j < m; j++) {
            if(!dcmp(A[j][id])) continue;

            double f = A[j][id] / A[i][id];
            for(int k = id + 1; k <= n; k++) A[j][k] -= A[i][k] * f;
            A[j][id] = 0;
        }
    }
}

int solve(mat &A) {
    int m = A.size(), n = A[0].size() - 1;
    guass(A, m, n);

    int r1 = 0, r2 = 0;
    for(int i = 0; i < m; i++) {
        bool sign = true;
        for(int j = 0; j <= n; j++) {
            if(dcmp(A[i][j])) {
                r2++;
                if(j < n) r1++;
                sign = false;
                break;
            }
        }
        if(sign) break;
    }

    if(r1 != r2) return -1;
    if(r1 == r2 && r1 != n) return 0;
    for(int i = n - 1; i >= 0; i--) {
        A[i][n] /= A[i][i];
        for(int j = i - 1; j >= 0; j--) A[j][n] -= A[i][n] * A[j][i];
    }
    return 1;
}
```

## 数学公式

GCD(a,b,c)=1,则必然有 ax+by+cz=1，与扩展欧几里德的原理是一样的
若有 GCD(x,n)=1,那么在一个圈中隔点报数必能全部报完
x<=1e9,则说明最多只会由 9 个质数组成

## 其他

1.模拟退火

```
const int MX = 1500 + 5;
const int INF = 0x3f3f3f3f;
```

```
const double exps = 1e-3;//比要求精度低 2 个就行
const double pi = acos(-1.0);

double fx[MX], fy[MX], best[MX];
double PX[MX], PY[MX];

double Rand(double L, double R) {//区间内随机数生成函数
    return (rand() % 10000) / 10000.0 * (R - L) + L;
}

double dist(double x1, double y1, double x2, double y2) {
    return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
}

int main() {
    int T, X, Y, n;
    scanf("%d", &T);
    srand(time(NULL));
    while(T--) {
        scanf("%d%d%d", &X, &Y, &n);
        for(int i = 1; i <= n; i++) {
            scanf("%lf%lf", &PX[i], &PY[i]);
        }

        for(int i = 1; i <= 30; i++) {
            fx[i] = Rand(1, X);
            fy[i] = Rand(1, Y);
            best[i] = INF;
            for(int j = 1; j <= n; j++) {
                //评估函数，靠它来评估整个退火过程的好坏
                best[i] = min(best[i], dist(fx[i], fy[i], PX[j], PY[j]));
            }
        }

        double step = max(X, Y);//一般是最长的跨度
        while(step > exps) {
            for(int i = 1; i <= 30; i++) {//初始状态一般 30 个
                for(int j = 1; j <= 30; j++) {//一般循环 30 次即可
                    double angel = Rand(0, 2 * pi);//枚举任何角度，使得到的新点向四周扩散
                    double nx = fx[i] + cos(angel) * step;
                    double ny = fy[i] + sin(angel) * step;
                    if(nx < 0 || nx > X || ny < 0 || ny > Y) continue;

                    double d = INF;
                    for(int k = 1; k <= n; k++) {
                        d = min(d, dist(nx, ny, PX[k], PY[k]));
                    }
                    if(d > best[i]) {
                        best[i] = d;
                        fx[i] = nx;
                        fy[i] = ny;
                    }
                }
            }
            step *= 0.85;//退火，常数，不管
        }
        int t = 1;
        for(int i = 1; i <= 30; i++) {
```

```
            if(best[i] >= best[t]) {//找到退火后的状态中，最优的
                t = i;
            }
        }

        printf("The safest point is (%.1lf, %.1lf).\n", fx[t], fy[t]);
    }
    return 0;
}
```

2.二分查找

```
int BS(int A[], int L, int R, int x) {
    int l = L, r = R, m;
    while(l <= r) {
        m = (l + r) >> 1;
        if(A[m] == x) return m;
        if(A[m] < x)  l = m + 1;
        else       r = m - 1;
    }
    return -1;
}
```

3.long double

```
/*#include<iomanip>
cout<<fixed<<setprecision(15)<<a;设置精度为 15

typedef long double LDB;//ldb 只能用 cout 输出*/
```

4.加速挂

```
inline void read(int &x) {
    char c = getchar();
    while(!isdigit(c)) c = getchar();

    x = 0;
    while(isdigit(c)) {
        x = x * 10 + c - '0';
        c = getchar();
    }
}
```

5.快速排序

```
void qsort(int A[], int L, int R) {
    if(L >= R) return;

    int s = A[L], l = L, r = R;
    while(l < r) {
        while(l < r && A[r] >= s) r--;
        A[l] = A[r];

        while(l < r && A[l] <= s) l++;
        A[r] = A[l];
    }
    A[l] = s;

    qsort(A, L, l - 1);
    qsort(A, l + 1, R);
}
```

6.手动扩栈

```
#pragma comment(linker, "/STACK:102400000,102400000")
```

7.尺取法处理连续区间

```
int L, R, ans = 0;
for(L = 1; L <= n; L = R + 1) {
    for(R = L; R + 1 <= n && B[L] == B[R + 1]; R++);
    ans = max(ans, R - L + 1);
}
```

8.头文件

```
#include<map>
#include<set>
#include<cmath>
#include<stack>
#include<queue>
#include<cstdio>
#include<string>
#include<vector>
#include<cstring>
#include<iostream>
#include<algorithm>
#include<functional>
#define MEM0(a) memset(a, 0, sizeof(a))
#define MEM1(a) memset(a, -1, sizeof(a))
#define FIN freopen("input.txt","r",stdin)
#define FOUT freopen("output.txt","w+",stdout)
#define MEMINF(a) memset(a, 0x3f3f3f3f, sizeof(a))
#define UFOR(i, a, b) for(int i = (int)a; i <= (int)b; i++)
#define DFOR(i, a, b) for(int i = (int)a; i >= (int)b; i--)

using namespace std;
typedef long long LL;
typedef pair<int, int>PII;
```