Contents

1 BASIC

1.1 對拍

```
@echo off
g++ ac.cpp -o ac.exe
g++ wa.cpp -o wa.exe
set /a num=1
:loop
   echo %num%
   C:/msys64/ucrt64/bin/python.exe gen.py > input
   ac.exe < input > ac
   wa.exe < input > wa
   fc ac wa
   set /a num=num+1
if not errorlevel 1 goto loop
```

1.2 C++ 聖經

```
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
```

1.3 random

```
int randint(int l, int r){
    mt19937 mt(chrono::steady_clock::now().
        time_since_epoch().count());
    uniform_int_distribution<> dis(l, r); return dis(mt
        );
}
```

2 資料結構

2.1 線段樹

```
#define cl(x) (x << 1)
#define cr(x) (x << 1) + 1

struct SEG2 {
#define MXN 200500
    int n;
    // vector<int> seg;
    // vector<int> arr, tag;
    int seg[MXN], arr[MXN], tag[MXN];
    void init(int a) {
        n = a;
        // seg.resize(4 * (n + 5), 0);
        // tag.resize(4 * (n + 5), 0);
        // arr.resize(n + 5, 0);
        for (int i = 0; i < n + 5; i++)
            arr[i] = 0;
        for (int i = 0; i < 4 * (n + 5); i++)
            seg[i] = tag[i] = 0;
    }
    void push(int id, int l, int r) {
        if (tag[id] != 0) {
            seg[id] += tag[id] * (r - l + 1);
            if (l != r) {
                tag[cl(id)] += tag[id];
                tag[cr(id)] += tag[id];
            }
            tag[id] = 0;
        }
    }
    void pull(int id, int l, int r) {
        int mid = (l + r) >> 1;
        push(cl(id), l, mid);
        push(cr(id), mid + 1, r);
        int a = seg[cl(id)];
        int b = seg[cr(id)];
        seg[id] = a + b;
    }
    void build(int id, int l, int r) {
        if (l == r) {
            seg[id] = arr[l];
            return;
        }
        int mid = (l + r) >> 1;
        build(cl(id), l, mid);
        build(cr(id), mid + 1, r);
        pull(id, l, r);
    }
    void update(int id, int l, int r, int ql, int qr,
        int v) {
        push(id, l, r);
        if (ql <= l && r <= qr) {
            tag[id] += v;
            return;
        }
        int mid = (l + r) >> 1;
        if (ql <= mid)
            update(cl(id), l, mid, ql, qr, v);
        if (qr > mid)
            update(cr(id), mid + 1, r, ql, qr, v);
        pull(id, l, r);
    }
    int query(int id, int l, int r, int ql, int qr) {
        push(id, l, r);
        if (ql <= l && r <= qr) {
            return seg[id];
        }
        int mid = (l + r) >> 1;
        int ans1, ans2;
        bool f1 = 0, f2 = 0;
        if (ql <= mid) {
            ans1 = query(cl(id), l, mid, ql, qr);
            f1 = 1;
        }
        if (qr > mid) {
            ans2 = query(cr(id), mid + 1, r, ql, qr);
            f2 = 1;
        }
        if (f1 && f2)
            return ans1 + ans2;
        if (f1)
            return ans1;
        return ans2;
    }
    void build() { build(1, 1, n); }
    int query(int ql, int qr) { return query(1, 1, n,
        ql, qr); }
    void update(int ql, int qr, int val) { update(1, 1,
         n, ql, qr, val); }
};
```

2.2 BIT

```
struct STRUCT_BIT {
    int n;
    vector<int> bit;
    int lowbit(int x) { return x & -x; }
    void init(int _n) {
        n = _n + 1;
        bit = vector<int>(n, 0);
    }
    void update(int x, int v) {
        for (; x < n; x += lowbit(x)) {
            bit[x] += v;
        }
    }
    int query(int x) {
        int ret = 0;
        for (; x > 0; x -= lowbit(x)) {
            ret += bit[x];
```

```
        }
        return ret;
    }
};
```

2.3 Fenwicktree

```
struct fenwick{
  #define lowbit(x) (x&-x)
  int n;
  vector<int> v;
  fenwick(int _n) : n(_n+1),v(_n+2){}
  void add(int x,int u){
    ++x;
    for(;x < n; x += lowbit(x)) v[x] += u;
  }
  int qry(int x){
    ++x; int ret = 0;
    for(; x ; x -= lowbit(x)) ret += v[x];
    return ret;
  }
  int qry(int l,int r) { return query(r) - query(l-1);
      }
  int kth(int k){ // lower_bound(k)
    int x = 0; --k;
    for(int i = (1<<__lg(n)); i;i >>= 1){
      if(x + i <= n and k >= v[x + i]) x += i; k -= v[x
          ];
    }
    return x;
  }

};
```

3 樹

3.1 LCA

```
int n, q;
int anc[MAXN][25], in[MAXN], out[MAXN];
vector<int> edge[MAXN];
int timing = 1;
void dfs(int cur, int fa) {
    anc[cur][0] = fa;
    in[cur] = timing++;
    for (int nex : edge[cur]) {
        if (nex == fa) continue;
        dfs(nex, cur);
    }
    out[cur] = timing++;
}
void init() {
    dfs(1, 0);
    for (int i = 1; i < 25; i++) {
        for (int cur = 1; cur <= n; cur++) {
            anc[cur][i] = anc[anc[cur][i - 1]][i - 1];
        }
    }
}
bool isanc(int u, int v) { return (in[u] <= in[v] &&
    out[v] <= out[u]); }
int lca(int a, int b) {
    if (isanc(a, b)) return a;
    if (isanc(b, a)) return b;
    for (int i = 24; i >= 0; i--) {
        if (anc[a][i] == 0) continue;
        if (!isanc(anc[a][i], b)) a = anc[a][i];
    }

    return anc[a][0];
}
```

3.2 DSUONTREE

```
int ans[MXN], color[MXN], son[MXN];
map<int, int> mp[MXN];
void dfs(int x, int f){
    if(son[x]){
        dfs(son[x], x);
        swap(mp[x], mp[son[x]]);
        ans[x] = ans[son[x]];
    }
    mp[x][color[x]]++;
    ans[x] = max(ans[x], mp[x][color[x]]);
    for(int i : edge[x]){
        if(i == f || i == son[x])    continue;
        dfs(i, x);
        for(auto j : mp[i]){
            mp[x][j.first] += j.second;
            ans[x] = max(ans[x], mp[x][j.first]);
        }
    }
}
```

4 計算幾何

4.1 基本定義

```
const ld eps = 1e-8, PI = acos(-1);
struct PT {  // 定義點
    int x, y;
    PT(int _x = 0, int _y = 0) : x(_x), y(_y) {}
    bool operator==(const PT& a) const { return a.x ==
        x && a, y == y; }
    PT operator+(const PT& a) const { return PT(x + a.x
        , y + a.y); }
    PT operator-(const PT& a) const { return PT(x - a.x
        , y - a.y); }
    PT operator*(const int& a) const { return PT(x * a,
         y * a); }
    PT operator/(const int& a) const { return PT(x / a,
         y / a); }
    int operator*(const PT& a) const {  // 計算幾何程式
        碼中內積通常用*表示
        return x * a.x + y * a.y;
    }
    int operator^(const PT& a) const {  // 計算幾何程式
        碼中外積通常用^表示
        return x * a.y - y * a.x;
    }
    int length2() { return x * x + y * y; }          //
         回傳距離平方
    double length() { return sqrt(x * x + y * y); }  //
         回傳距離
    bool operator<(const PT& a) const {  // 判斷兩點座
        標 先比 x 再比 y
        return x < a.x || (x == a.x && y < a.y);
    }
    friend int cross(const PT& o, const PT& a, const PT
        & b) {
        PT lhs = o - a, rhs = o - b;
        return lhs.x * rhs.y - lhs.y * rhs.x;
    }
};
struct CIRCLE {  // 圓心，半徑
    PT o;
    ld r;
};
struct LINE {  // 點， 向量
    PT p, v;
};
int judge(ld a, ld b) {  // 判斷浮點數大小
    // 等於回傳0，小於回傳-1，大於回傳1
    if (fabs(a - b) < eps)
        return 0;
    if (a < b)
        return -1;
    return 1;
}
PT zhixianjiaodian(LINE a, LINE b) {  // 求兩直線交點
    PT u = a.p - b.p;
    ld t = (b.v ^ u) / (a.v ^ b.v);
    return a.p + (a.v * t);
}
PT zhuanzhuan(PT a, ld angle) {  // 向量旋轉
    return {a.x * cos(angle) + a.y * sin(angle),
            -a.x * sin(angle) + a.y * cos(angle)};
}
LINE bisector(PT a, PT b) {  // 中垂線
    PT p = (a + b) / 2;
    PT v = zhuanzhuan(b - a, PI / 2);
    return {p, v};
}
CIRCLE getcircle(PT a, PT b, PT c) {  // 三點求外接圓
```

```
    auto n = bisector(a, b), m = bisector(a, c);
    PT o = zhixianjiaodian(n, m);
    ld r = (o - a).length();
    return {o, r};
}
bool collinearity(const PT& a, const PT& b, const PT& c
    ) {  // 是否三點共線
    return ((b - a) ^ (c - a)) == 0;
}
bool inLine(const PT& p, const LINE& li) {  // 是否在線
    段上
    PT st, ed;
    st = li.p, ed = st + li.v;
    return collinearity(st, ed, p) && (st - p) * (ed -
        p) < 0;
}
int dcmp(ld x) {
    if (abs(x) < eps)
        return 0;
    else
        return x < 0 ? -1 : 1;
}
Pt LLIntersect(Line a, Line b) {
    Pt p1 = a.s, p2 = a.e, q1 = b.s, q2 = b.e;
    ld f1 = (p2 - p1) ^ (q1 - p1), f2 = (p2 - p1) ^ (p1
         - q2), f;
    if (dcmp(f = f1 + f2) == 0)
        return dcmp(f1) ? Pt(NAN, NAN) : Pt(INFINITY,
            INFINITY);
    return q1 * (f2 / f) + q2 * (f1 / f);
}
int ori(const Pt& o, const Pt& a, const Pt& b) {
    LL ret = (a - o) ^ (b - o);
    return (ret > 0) - (ret < 0);
}
// p1 == p2 || q1 == q2 need to be handled
bool banana(const Pt& p1, const Pt& p2, const Pt& q1,
    const Pt& q2) {
    if (((p2 - p1) ^ (q2 - q1)) == 0) {  // parallel
        if (ori(p1, p2, q1))
            return false;
        return ((p1 - q1) * (p2 - q1)) <= 0 || ((p1 -
            q2) * (p2 - q2)) <= 0 ||
               ((q1 - p1) * (q2 - p1)) <= 0 || ((q1 -
                   p2) * (q2 - p2)) <= 0;
    }
    return (ori(p1, p2, q1) * ori(p1, p2, q2) <= 0) &&
           (ori(q1, q2, p1) * ori(q1, q2, p2) <= 0);
}
```

4.2 凸包

```
vector<PT> convex_hull(vector<PT> &hull) {
    sort(hull.begin(), hull.end());
    int top = 0;
    vector<PT> stk;
    for (int i = 0; i < hull.size(); i++) {
        while (top >= 2 && cross(stk[top - 2], stk[top
            - 1], hull[i]) <= 0)
            stk.pop_back(), top--;
        stk.push_back(hull[i]);
        top++;
    }
    for (int i = hull.size() - 2, t = top + 1; i >= 0;
        i--) {
        while (top >= t && cross(stk[top - 2], stk[top
            - 1], hull[i]) <= 0)
            stk.pop_back(), top--;
        stk.push_back(hull[i]);
        top++;
    }
    stk.pop_back();
    return stk;
}
```

4.3 極腳排序

```
bool cmp(const Pt& lhs, const Pt rhs){
    return atan2(lhs.y, lhs.x) < atan2(rhs.y, rhs.x);
}
sort(P.begin(), P.end(), cmp);

bool cmp(const Pt& lhs, const Pt rhs){
    if((lhs < Pt(0, 0)) ^ (rhs < Pt(0, 0)))
        return (lhs < Pt(0, 0)) < (rhs < Pt(0, 0));
    return (lhs ^ rhs) > 0;
} // 從 270 度開始逆時針排序

sort(P.begin(), P.end(), cmp);
```

4.4 最遠點對

```
int RoatingCalipers(vector<PT> &tubao) { // 最遠點對 回
    傳距離平方
    int nn = tubao.size();
    int ret = 0;
    if (tubao.size() <= 2) {
        return (tubao[0] - tubao[1]).length2();
    }
    for (int i = 0, j = 2; i < nn; i++) {
        PT a = tubao[i], b = tubao[(i + 1) % nn];
        while (((a - tubao[j]) ^ (b - tubao[j])) <
               ((a - tubao[(j + 1) % nn]) ^ (b - tubao
                   [(j + 1) % nn])))
            j = (j + 1) % nn;
        ret = max(ret, (a - tubao[j]).length2());
        ret = max(ret, (b - tubao[j]).length2());
    }
    return ret;
}
```

4.5 最小圓包覆-隨機增量

```
CIRCLE getmec(vector<PT> &p) {
    int n = p.size();
    random_shuffle(p.begin(), p.end());
    CIRCLE c = {p[0], 0};
    for (int i = 1; i < n; i++) {
        if (judge(c.r, (c.o - p[i]).length()) == -1) {
            c = {p[i], 0};
            for (int j = 0; j < i; j++) {
                if (judge(c.r, (c.o - p[j]).length())
                    == -1) {
                    c = {(p[i] + p[j]) / 2, (p[i] - p[j
                        ]).length() / 2};
                    for (int k = 0; k < j; k++) {
                        if (judge(c.r, (c.o - p[k]).
                            length()) == -1)
                            c = getcircle(p[i], p[j], p
                                [k]);
                    }
                }
            }
        }
    }
    return c;
}
```

4.6 最小圓包覆-三分搜尋

```
PT arr[MXN];
int n = 10;
double checky(double x, double y) {
    double cmax = 0;
    for (int i = 0; i < n; i++) {  // 過程中回傳距離^2
        避免不必要的根號運算
        cmax = max(cmax, (arr[i].x - x) * (arr[i].x - x
            ) + (arr[i].y - y) * (arr[i].y - y));
    }
    return cmax;
}
double checkx(double x) {
    double yl = -1e9, yr = 1e9;
    while (yr - yl > EPS) {
        double ml = (yl + yl + yr) / 3, mr = (yl + yr +
             yr) / 3;
        if (checky(x, ml) < checky(x, mr))
            yr = mr;
        else
            yl = ml;
    }
}
signed main() {
    double xl = -1e9, xr = 1e9;
```

```
    while (xr - xl > EPS) {
        double ml = (xl + xl + xr) / 3, mr = (xl + xr +
             xr) / 3;
        if (checkx(ml) < checkx(mr))
            xr = mr;
        else
            xl = ml;
    }
}
```