Sharif University of Technology - Kolompeh - Notebook

Contents

1 Geometry

1.1 Rotating Calipers

```
vector <pair<pt, pt>> get_antipodals(vector <pt> &p){
        int n = sz(p);
        sort(p.begin(), p.end());
        vector <pt> U, L;
        for (int i = 0; i < n; i++){
                while (sz(U) > 1 && side(U[sz(U)-2], U[sz(U)-1], p[i])
                     >= 0)
                        U.pop_back();
                while (sz(L) > 1 && side(L[sz(L)-2], L[sz(L)-1], p[i])
                     <= 0)
                        L.pop_back();
                U.pb(p[i]);
                L.pb(p[i]);
        }
        vector <pair<pt, pt>> res;
        int i = 0, j = sz(L)-1;
        while (i+1 < sz(U) || j > 0){
                res.pb({U[i], L[j]});
                if (i+1 == sz(U))
                        j--;
                else if (j == 0)
                        i++;
                else if (cross(L[j]-L[j-1], U[i+1]-U[i]) >= 0)
                        i++;
                else
                        j--;
        }
        return res;
}
```

1.2 Delaunay Triangulation O(n2)

```
struct Delaunay{
        vector <pt> p;
        vector <pt> to;
        vector <int> nxt;

        int add_edge(pt q, int bef=-1){
                int cnt = sz(to);
                to.pb(q);
                nxt.pb(-1);
                if (bef != -1){
                        nxt[bef] = cnt;
                        to.pb(to[bef]);
                        nxt.pb(-1);
                }
                return cnt;
        }

        int before(int e){
                int cur = e, last = -1;
                do{
                        last = cur;
                        cur = nxt[cur^1];
                }while (cur != e);
                return last^1;
        }

        void easy_triangulate(){
                to.clear();
                nxt.clear();
                sort(p.begin(), p.end());
                if (dir(p[0], p[1], p[2]) > 0)
                        swap(p[1], p[2]);
                int to0 = add_edge(p[0]), to0c = add_edge(p[2]),
                        to1 = add_edge(p[1]), to1c = add_edge(p[0]),
                        to2 = add_edge(p[2]), to2c = add_edge(p[1]);

                nxt[to1] = to2; nxt[to2] = to0;
                nxt[to0] = to1; nxt[to0c] = to2c;
                nxt[to2c] = to1c; nxt[to1c] = to0c;

                int e = to0;
                for (int i = 3; i < sz(p); i++){
                        pt q = p[i];
                        while (dir(q, to[e^1], to[e]) < 0 || dir(q, to
                            [e^1], to[before(e)^1]) < 0)
                                e = nxt[e];
                        vector <int> vis;
                        while (dir(q, to[e^1], to[e]) > 0 || dir(q, to
                            [e^1], to[before(e)^1]) > 0){
                                vis.pb(e);
                                e = nxt[e];
                        }
                        int ex = add_edge(q, before(vis[0]));
                        int last = ex^1;
                        for (int edge : vis){
                                nxt[last] = edge;
                                int eq = add_edge(q, edge);
                                nxt[edge] = eq;
                                nxt[eq] = last;
                                last = eq^1;
                        }
                        nxt[ex] = last;
                        nxt[last] = e;
                }
        }

        bool incircle(pt a, pt b, pt c, pt d) {
                return a.z() * (b.x * (c.y - d.y) - c.x * (b.y - d.y)
                    + d.x * (b.y - c.y))
                        - b.z() * (a.x * (c.y - d.y) - c.x * (a.y - d.
                            y) + d.x * (a.y - c.y))
```

```
                + c.z() * (a.x * (b.y - d.y) - b.x * (a.y - d.
                    y) + d.x * (a.y - b.y))
                - d.z() * (a.x * (b.y - c.y) - b.x * (a.y - c.
                    y) + c.x * (a.y - b.y)) > 0;
}


bool locally(int e){
        pt a = to[e^1], b = to[e], c = to[nxt[e]], d = to[nxt[
            e^1]];
        if (dir(a, b, c) < 0) return true;
        if (dir(b, a, d) < 0) return true;
        if (incircle(a, b, c, d)) return false;
        if (incircle(b, a, d, c)) return false;
        return true;
}

void flip(int e){
        int a = nxt[e], b = nxt[a],
                c = nxt[e^1], d = nxt[c];
        nxt[d] = a;
        nxt[b] = c;
        to[e] = to[c];
        nxt[a] = e;
        to[e^1] = to[a];
        nxt[c] = e^1;
}

void delaunay_triangulate(){
        if (sz(to) == 0)
                easy_triangulate();
        bool *mark = new bool[sz(to)];
        fill(mark, mark + sz(to), false);
        vector <int> bad;
        for (int e = 0; e < sz(to); e++){
                if (!mark[e/2] && !locally(e)){
                        bad.pb(e);
                        mark[e/2] = true;
                }
        }
        while (sz(bad)){
                int e = bad.back();
                bad.pop_back();
                mark[e/2] = false;
                if (!locally(e)){
                        flip(e);
                        int to_check[4] = {nxt[e], nxt[nxt[e
                            ]], nxt[e^1], nxt[nxt[e^1]]};
                        for (int i = 0; i < 4; i++)
                                if (!mark[to_check[i]/2] &&
                                    locally(to_check[i])){
                                        bad.pb(to_check[i]);
                                        mark[to_check[i]/2] =
                                            true;
                                }
                }
        }
}

vector <tri> get_triangles(){
        vector <tri> res;
        for (int e = 0; e < sz(to); e++){
                pt a = to[e^1], b = to[e], c = to[nxt[e]];
                if (dir(a, b, c) < 0) continue;
                res.pb(tri(a, b, c));
        }
        return res;
}
Delaunay(vector <pt> p):p(p){}
};
```