Gungnir's Standard Code Library

Shanghai Jiao Tong University

Dated: July 29, 2016

Contents

Chapter 1 计算几何

1.1 二维

1.1.1 基础

```cpp
typedef double DB;
const DB eps = 1e-8;

__inline int sign(DB x) {
    return x < -eps ? -1 : ( x > eps ? 1 : 0 );
}
__inline DB msqrt(DB x) {
    return sign(x) > 0 ? sqrt(x) : 0;
}

struct Point {
    DB x, y;
    __inline Point(): x(0), y(0) {}
    __inline Point(DB x, DB y): x(x), y(y) {}
    __inline Point operator+(const Point &rhs) const {
        return Point(x + rhs.x, y + rhs.y);
    }
    __inline Point operator-(const Point &rhs) const {
        return Point(x - rhs.x, y - rhs.y);
    }
    __inline Point operator*(DB k) const {
        return Point(x * k, y * k);
    }
    __inline Point operator/(DB k) const {
        assert(sign(k));
        return Point(x / k, y / k);
    }
};

__inline DB dot(const P& a, const P& b) {
    return a.x * b.x + a.y * b.y;
}

__inline DB det(const P& a, const P& b) {
    return a.x * b.y - a.y * b.x;
}
```

1.1.2 凸包

```cpp
__inline void clear(std::vector<Point>& v) {
    v.clear();
    std::vector<Point>(v).swap(v);
}

struct Convex {
    int n;
    std::vector<Point> a, upper, lower;
    void make_shell(const std::vector<Point>& p,
            std::vector<Point>& shell) {  // p needs to be sorted.
        clear(shell); int n = p.size();
        for (int i = 0, j = 0; i < n; i++, j++) {
            for (; j >= 2 && sign(det(shell[j-1] - shell[j-2],
                            p[i] - shell[j-2])) <= 0; --j) shell.pop_back();
            shell.push_back(p[i]);
        }
    }
    void make_convex() {
        std::sort(a.begin(), a.end());
        make_shell(a, lower);
        std::reverse(a.begin(), a.end());
        make_shell(a, upper);
        a = lower;
        for (std::vector<Point>::iterator it = upper.begin(); it != upper.end(); it++)
            if (!(*it == *a.rbegin()) && !(*it == *a.begin()))
                a.push_back(*it);
        n = a.size();
    }
    void init(const std::vector<Point>& _a) {
        clear(a); a = _a; n = a.size();
        make_convex();
    }
    void read(int _n) {  // Won't make convex.
        clear(a); n = _n; a.resize(n);
        for (int i = 0; i < n; i++)
            a[i].read();
    }
    std::pair<DB, int> get_tangent(
            const std::vector<Point>& convex, const Point& vec) {
        int l = 0, r = (int)convex.size() - 2;
        assert(r >= 0);
        for (; l + 1 < r; ) {
            int mid = (l + r) / 2;
            if (sign(det(convex[mid + 1] - convex[mid], vec)) > 0)
                r = mid;
            else l = mid;
        }
        return std::max(std::make_pair(det(vec, convex[r]), r),
                std::make_pair(det(vec, convex[0]), 0));
    }
    int binary_search(Point u, Point v, int l, int r) {
        int s1 = sign(det(v - u, a[l % n] - u));
        for (; l + 1 < r; ) {
            int mid = (l + r) / 2;
            int smid = sign(det(v - u, a[mid % n] - u));
            if (smid == s1) l = mid;
            else r = mid;
        }
        return l % n;
    }
    // 求凸包上和向量 vec 叉积最大的点，返回编号，共线的多个切点返回任意一个
    int get_tangent(Point vec) {
        std::pair<DB, int> ret = get_tangent(upper, vec);
        ret.second = (ret.second + (int)lower.size() - 1) % n;
        ret = std::max(ret, get_tangent(lower, vec));
        return ret.second;
    }
    // 求凸包和直线 u, v 的交点，如果不相交返回 false，如果有则是和 (i, next(i)) 的交点，交在点上不确定返回前后两条边其中之一
    bool get_intersection(Point u, Point v, int &i0, int &i1) {
        int p0 = get_tangent(u - v), p1 = get_tangent(v - u);
        if (sign(det(v - u, a[p0] - u)) * sign(det(v - u, a[p1] - u)) <= 0) {
            if (p0 > p1) std::swap(p0, p1);
            i0 = binary_search(u, v, p0, p1);
            i1 = binary_search(u, v, p1, p0 + n);
            return true;
        }
        else return false;
    }
};
```

Chapter 2 图论

2.1 基础

```cpp
struct Graph {  // Remember to call .init()!
    int e, nxt[M], v[M], adj[N], n;
    bool base;
    __inline void init(bool _base, int _n = 0) {
        assert(n < N);
        n = _n; base = _base;
        e = 0; memset(adj + base, -1, sizeof(*adj) * n);
    }
    __inline int new_node() {
        adj[n + base] = -1;
        assert(n + base + 1 < N);
        return n++ + base;
    }
```

```
    __inline void ins(int u0, int v0) {  // directional
        assert(u0 < n + base && v0 < n + base);
        v[e] = v0; nxt[e] = adj[u0]; adj[u0] = e++;
        assert(e < M);
    }
    __inline void bi_ins(int u0, int v0) {  // bi-directional
        ins(u0, v0); ins(v0, u0);
    }
};
```

2.2 KM

```
struct KM {
    // Truly O(n^3)
    // 邻接矩阵，不能连的边设为 -INF，求最小权匹配时边权取负，但不能连的还是 -INF，
    ↪ 使用时先对 1 -> n 调用 hungary() ，再 get_ans() 求值
    int w[N][N];
    int lx[N], ly[N], match[N], way[N], slack[N];
    bool used[N];
    void initialization() {
        for(int i = 1; i <= n; i++) {
            match[i] = 0;
            lx[i] = 0;
            ly[i] = 0;
            way[i] = 0;
        }
    }
    void hungary(int x) {  // for i(1 -> n) : hungary(i);
        match[0] = x;
        int j0 = 0;
        for(int j = 0; j <= n; j++){
            slack[j] = INF;
            used[j] = false;
        }

        do {
            used[j0] = true;
            int i0 = match[j0], delta = INF, j1;
            for(int j = 1; j <= n; j++) {
                if(used[j] == false) {
                    int cur = -w[i0][j] - lx[i0] - ly[j];
                    if(cur < slack[j]) {
                        slack[j] = cur;
                        way[j] = j0;
                    }
                    if(slack[j] < delta) {
                        delta = slack[j];
                        j1 = j;
                    }
                }
            }
            for(int j = 0; j <= n; j++) {
                if(used[j]) {
                    lx[match[j]] += delta;
                    ly[j] -= delta;
                }
                else slack[j] -= delta;
            }
            j0 = j1;
        } while (match[j0] != 0);

        do {
            int j1 = way[j0];
            match[j0] = match[j1];
            j0 = j1;
        } while(j0);
    }

    int get_ans() {  // maximum ans
        int sum = 0;
        for(int i = 1; i<= n; i++)
            if(match[i] > 0) sum += -w[match[i]][i];
        return sum;
    }
};
```

2.3 点双连通分量

dcc.forest is a set of connected tree whose vertices are chequered with cut-vertex and DCC.

```
const bool DCC_VERTEX = 0, DCC_EDGE = 1;
struct DCC {  // N = N0 + M0. Remember to call init(&raw_graph).
    Graph *g, forest; // g is raw graph ptr.
    int dfn[N], DFN, low[N];
    int stack[N], top;
    int expand_to[N];      // Where edge i is expanded to in expaned graph.
    // Vertex i expaned to i.
    int compress_to[N];  // Where vertex i is compressed to.
    bool vertex_type[N], cut[N], compress_cut[N], branch[M];
    //std::vector<int> DCC_component[N];  // Cut vertex belongs to none.
    __inline void init(Graph *raw_graph) {
        g = raw_graph;
    }
    void DFS(int u, int pe) {
        dfn[u] = low[u] = ++DFN; cut[u] = false;
        if (!~g->adj[u]) {
            cut[u] = 1;
            compress_to[u] = forest.new_node();
            compress_cut[compress_to[u]] = 1;
        }
        for (int e = g->adj[u]; ~e; e = g->nxt[e]) {
            int v = g->v[e];
            if ((e ^ pe) > 1 && dfn[v] > 0 && dfn[v] < dfn[u]) {
                stack[top++] = e;
                low[u] = std::min(low[u], dfn[v]);
            }
            else if (!dfn[v]) {
                stack[top++] = e; branch[e] = 1;
                DFS(v, e);
                low[u] = std::min(low[v], low[u]);
                if (low[v] >= dfn[u]) {
                    if (!cut[u]) {
                        cut[u] = 1;
                        compress_to[u] = forest.new_node();
                        compress_cut[compress_to[u]] = 1;
                    }
                    int cc = forest.new_node();
                    forest.bi_ins(compress_to[u], cc);
                    compress_cut[cc] = 0;
                    //DCC_component[cc].clear();
                    do {
                        int cur_e = stack[--top];
                        compress_to[expand_to[cur_e]] = cc;
                        compress_to[expand_to[cur_e^1]] = cc;
                        if (branch[cur_e]) {
                            int v = g->v[cur_e];
                            if (cut[v])
                                forest.bi_ins(cc, compress_to[v]);
                            else {
                                //DCC_component[cc].push_back(v);
                                compress_to[v] = cc;
                            }
                        }
                    } while (stack[top] != e);
                }
            }
        }
    }
    void solve() {
        forest.init(g->base);
        int n = g->n;
```

```
        for (int i = 0; i < g->e; i++) {
            expand_to[i] = g->new_node();
        }
        memset(branch, 0, sizeof(*branch) * g->e);
        memset(dfn + g->base, 0, sizeof(*dfn) * n); DFN = 0;
        for (int i = 0; i < n; i++)
            if (!dfn[i + g->base]) {
                top = 0;
                DFS(i + g->base, -1);
            }
    }
} dcc;

dcc.init(&raw_graph);
dcc.solve();
// Do something with dcc.forest ...
```

Chapter 3 技巧

3.1 释放 STL 容器内存空间

```
// vectors for example.
std::vector<int> v;
// Do something with v...
v.clear(); // Or having erased many.
std::vector<int>(v).swap(v);
```

3.2 大整数相乘取模

Time complexity $O(1)$.

```
// 需要保证 x 和 y 非负
long long mult(long long x, long long y, long long MODN) {
    long long t = (x * y - (long long)((long double)x / MODN * y + 1e-3) * MODN) % MODN;
    return t < 0 ? t + MODN : t;
}
```