Gungnir's Standard Code Library

Shanghai Jiao Tong University

Dated: July 25, 2016

Contents

Chapter 1

Computational Geometry

1.1 2D

1.1.1 Basis

```
1  typedef double DB;
2  const DB eps = 1e-8;
3
4  __inline int sign(DB x) {
5      return x < -eps ? -1 : ( x > eps ? 1 : 0 );
6  }
7  __inline DB msqrt(DB x) {
8      return sign(x) > 0 ? sqrt(x) : 0;
9  }
10
11 struct Point {
12     DB x, y;
13     __inline Point(): x(0), y(0) {}
14     __inline Point(DB x, DB y): x(x), y(y) {}
15     __inline Point operator+(const Point &rhs) const {
16         return Point(x + rhs.x, y + rhs.y);
17     }
18     __inline Point operator-(const Point &rhs) const {
19         return Point(x - rhs.x, y - rhs.y);
20     }
21     __inline Point operator*(DB k) const {
22         return Point(x * k, y * k);
23     }
24     __inline Point operator/(DB k) const {
25         assert(sign(k));
26         return Point(x / k, y / k);
27     }
28 };
29
```

```
30  __inline DB dot(const P& a, const P& b) {
31      return a.x * b.x + a.y * b.y;
32  }
33
34  __inline DB det(const P& a, const P& b) {
35      return a.x * b.y − a.y * b.x;
36  }
```

Chapter 2

Graph Theory

2.1 Basis

```
1  struct Graph {  // Remember to call .init()!
2      int e, nxt[M], v[M], adj[N], n;
3      bool base;
4      __inline void init(bool _base, int _n = 0) {
5          assert(n < N);
6          n = _n; base = _base;
7          e = 0; memset(adj + base, −1, sizeof(*adj) * n);
8      }
9      __inline int new_node() {
10         adj[n + base] = −1;
11         assert(n + base + 1 < N);
12         return n++ + base;
13     }
14     __inline void ins(int u0, int v0) {  // directional
15         assert(u0 < n + base && v0 < n + base);
16         v[e] = v0; nxt[e] = adj[u0]; adj[u0] = e++;
17         assert(e < M);
18     }
19     __inline void bi_ins(int u0, int v0) {  // bi−directional
20         ins(u0, v0); ins(v0, u0);
21     }
22  };
```

2.2 Double Connected Graph (vertex)

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

```
1  const bool DCC_VERTEX = 0, DCC_EDGE = 1;
2  struct DCC {  // N = N0 + M0. Remember to call init(&raw_graph).
```

```
 3        Graph *g, forest; // g is raw graph ptr.
 4        int dfn[N], DFN, low[N];
 5        int stack[N], top;
 6        int expand_to[N];    // Where edge i is expanded to in expaned graph.
 7        // Vertex i expaned to i.
 8        int compress_to[N];  // Where vertex i is compressed to.
 9        bool vertex_type[N], cut[N], compress_cut[N], branch[M];
10        //std::vector<int> DCC_component[N];  // Cut vertex belongs to none.
11        __inline void init(Graph *raw_graph) {
12            g = raw_graph;
13        }
14        void DFS(int u, int pe) {
15            dfn[u] = low[u] = ++DFN; cut[u] = false;
16            if (!~g->adj[u]) {
17                cut[u] = 1;
18                compress_to[u] = forest.new_node();
19                compress_cut[compress_to[u]] = 1;
20            }
21            for (int e = g->adj[u]; ~e; e = g->nxt[e]) {
22                int v = g->v[e];
23                if ((e ^ pe) > 1 && dfn[v] > 0 && dfn[v] < dfn[u]) {
24                    stack[top++] = e;
25                    low[u] = std::min(low[u], dfn[v]);
26                }
27                else if (!dfn[v]) {
28                    stack[top++] = e; branch[e] = 1;
29                    DFS(v, e);
30                    low[u] = std::min(low[v], low[u]);
31                    if (low[v] >= dfn[u]) {
32                        if (!cut[u]) {
33                            cut[u] = 1;
34                            compress_to[u] = forest.new_node();
35                            compress_cut[compress_to[u]] = 1;
36                        }
37                        int cc = forest.new_node();
38                        forest.bi_ins(compress_to[u], cc);
39                        compress_cut[cc] = 0;
40                        //DCC_component[cc].clear();
41                        do {
42                            int cur_e = stack[--top];
43                            compress_to[expand_to[cur_e]] = cc;
44                            if (branch[cur_e]) {
45                                int v = g->v[cur_e];
46                                if (cut[v])
47                                    forest.bi_ins(cc, compress_to[v]);
48                                else {
49                                    //DCC_component[cc].push_back(v);
50                                    compress_to[v] = cc;
51                                }
52                            }
```

```
53                    } while (stack[top] != e);
54                }
55            }
56        }
57    }
58    void solve() {
59        forest.init(g->base);
60        int n = g->n;
61        for (int i = 0; i < g->e; i++) {
62            expand_to[i] = g->new_node();
63            branch[i] = 0;
64        }
65        memset(dfn + g->base, 0, sizeof(*dfn) * n); DFN = 0;
66        for (int i = 0; i < n; i++)
67            if (!dfn[i + g->base]) {
68                top = 0;
69                DFS(i + g->base, -1);
70            }
71    }
72 } dcc;
73
74 dcc.init(&raw_graph);
75 dcc.solve();
76 // Do something with dcc.forest ...
```