

# Gungnir's Standard Code Library

*Shanghai Jiao Tong University*

Dated: July 27, 2016

# Contents

<b>1</b>	<b>计算几何</b>	<b>2</b>
1.1	二维	2
1.1.1	基础	2
1.1.2	凸包	2
<b>2</b>	<b>图论</b>	<b>2</b>
2.1	基础	2
2.2	点双连通分量	3
<b>3</b>	<b>技巧</b>	<b>3</b>
3.1	释放 STL 容器内存空间	3

## Chapter 1 计算几何

### 1.1 二维

#### 1.1.1 基础

```
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
8 __inline DB msqrt(DB x) {
9     return sign(x) > 0 ? sqrt(x) : 0;
10 }
11
12 struct Point {
13     DB x, y;
14     __inline Point(): x(0), y(0) {}
15     __inline Point(DB x, DB y): x(x), y(y) {}
16     __inline Point operator+(const Point &rhs) const {
17         return Point(x + rhs.x, y + rhs.y);
18     }
19     __inline Point operator-(const Point &rhs) const {
20         return Point(x - rhs.x, y - rhs.y);
21     }
22     __inline Point operator*(DB k) const {
23         return Point(x * k, y * k);
24     }
25     __inline Point operator/(DB k) const {
26         assert(sign(k));
27         return Point(x / k, y / k);
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 }
```

#### 1.1.2 凸包

```
1 __inline void clear(std::vector<Point>& v) {
2     v.clear();
3     std::vector<Point>(v).swap(v);
4 }
5
6 struct Convex {
7     int n;
8     std::vector<Point> a, upper, lower;
9     void make_shell(const std::vector<Point>& p,
10         std::vector<Point>& shell) { // p needs to be sorted.
11         clear(shell); int n = p.size();
12         for (int i = 0, j = 0; i < n; i++, j++) {
13             for (; j >= 2 && sign(det(shell[j-1] - shell[j-2],
14                 p[i] - shell[j-2])) <= 0; --j) shell.pop_back();
15             shell.push_back(p[i]);
16         }
17     }
18     void make_convex() {
19         std::sort(a.begin(), a.end());
20         make_shell(a, lower);
21         std::reverse(a.begin(), a.end());
22         make_shell(a, upper);
23         a = lower;
24         for (std::vector<Point>::iterator it = upper.begin(); it != upper.end(); it++)
25             if (!(it == *a.rbegin()) && !(it == *a.begin()))
26                 a.push_back(*it);
27         n = a.size();
28     }
29 }
```

```
28 }
29 void init(const std::vector<Point>& _a) {
30     clear(a); a = _a; n = a.size();
31     make_convex();
32 }
33 void read(int _n) { // Won't make convex.
34     clear(a); n = _n; a.resize(n);
35     for (int i = 0; i < n; i++)
36         a[i].read();
37 }
38 std::pair<DB, int> get_tangent(
39     const std::vector<Point>& convex, const Point& vec) {
40     int l = 0, r = (int)convex.size() - 2;
41     assert(r >= 0);
42     for (; l + 1 < r; ) {
43         int mid = (l + r) / 2;
44         if (sign(det(convex[mid + 1] - convex[mid], vec)) > 0)
45             r = mid;
46         else l = mid;
47     }
48     return std::max(std::make_pair(det(vec, convex[r]), r),
49         std::make_pair(det(vec, convex[0]), 0));
50 }
51 int binary_search(Point u, Point v, int l, int r) {
52     int s1 = sign(det(v - u, a[l % n] - u));
53     for (; l + 1 < r; ) {
54         int mid = (l + r) / 2;
55         int smid = sign(det(v - u, a[mid % n] - u));
56         if (smid == s1) l = mid;
57         else r = mid;
58     }
59     return l % n;
60 }
61 // 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
62 int get_tangent(Point vec) {
63     std::pair<DB, int> ret = get_tangent(upper, vec);
64     ret.second = (ret.second + (int)lower.size() - 1) % n;
65     ret = std::max(ret, get_tangent(lower, vec));
66     return ret.second;
67 }
68 // 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
69 // 交点, 交在点上不确定返回前后两条边其中之一
70 bool get_intersection(Point u, Point v, int &i0, int &i1) {
71     int p0 = get_tangent(u - v), p1 = get_tangent(v - u);
72     if (sign(det(v - u, a[p0] - u)) * sign(det(v - u, a[p1] - u)) <= 0) {
73         if (p0 > p1) std::swap(p0, p1);
74         i0 = binary_search(u, v, p0, p1);
75         i1 = binary_search(u, v, p1, p0 + n);
76         return true;
77     }
78     else return false;
79 }
80 }
```

## Chapter 2 图论

### 2.1 基础

```
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 }
```

```

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 KM

```

1  struct KM {
2      // Truly O(n^3)
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,
   使用先对 1 -> n 调用 hungary(), 再 get_ans() 求值
4      int w[N][N];
5      int lx[N], ly[N], match[N], way[N], slack[N];
6      bool used[N];
7      void initialization() {
8          for(int i = 1; i <= n; i++) {
9              match[i] = 0;
10             lx[i] = 0;
11             ly[i] = 0;
12             way[i] = 0;
13         }
14     }
15     void hungary(int x) { // for i(1 -> n) : hungary(i);
16         match[x] = x;
17         int j0 = 0;
18         for(int j = 0; j <= n; j++){
19             slack[j] = INF;
20             used[j] = false;
21         }
22         do {
23             used[j0] = true;
24             int i0 = match[j0], delta = INF, j1;
25             for(int j = 1; j <= n; j++) {
26                 if(used[j] == false) {
27                     int cur = -w[i0][j] - lx[i0] - ly[j];
28                     if(cur < slack[j]) {
29                         slack[j] = cur;
30                         way[j] = j0;
31                     }
32                 }
33                 if(slack[j] < delta) {
34                     delta = slack[j];
35                     j1 = j;
36                 }
37             }
38             for(int j = 0; j <= n; j++) {
39                 if(used[j]) {
40                     lx[match[j]] += delta;
41                     ly[j] -= delta;
42                 }
43                 else slack[j] -= delta;
44             }
45             j0 = j1;
46         } while (match[j0] != 0);
47         do {
48             int j1 = way[j0];
49             match[j0] = match[j1];
50             j0 = j1;
51         } while(j0);
52     }
53     int get_ans() { // maximum ans
54         int sum = 0;
55         for(int i = 1; i <= n; i++)

```

```

59         if(match[i] > 0) sum += -w[match[i]][i];
60         return sum;
61     }
62 };

```

## 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.

```

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 expanded graph.
7      // Vertex i expanded 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[u], low[v]);
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                         compress_to[expand_to[cur_e^1]] = cc;
45                         if (branch[cur_e]) {
46                             int v = g->v[cur_e];
47                             if (cut[v])
48                                 forest.bi_ins(cc, compress_to[v]);
49                             else {
50                                 //DCC_component[cc].push_back(v);
51                                 compress_to[v] = cc;
52                             }
53                         }
54                     } while (stack[top] != e);
55                 }
56             }
57         }
58     }
59     void solve() {
60         forest.init(g->base);
61         int n = g->n;

```

```

62     for (int i = 0; i < g->e; i++) {
63         expand_to[i] = g->new_node();
64     }
65     memset(branch, 0, sizeof(*branch) * g->e);
66     memset(dfn + g->base, 0, sizeof(*dfn) * n); DFN = 0;
67     for (int i = 0; i < n; i++)
68         if (!dfn[i + g->base]) {
69             top = 0;
70             DFS(i + g->base, -1);
71         }
72     }
73 } dcc;
74
75 dcc.init(&raw_graph);

```

```

76 dcc.solve();
77 // Do something with dcc.forest ...

```

## Chapter 3 技巧

### 3.1 释放 STL 容器内存空间

```

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

```