

Gungnir's Standard Code Library

Shanghai Jiao Tong University

Dated: August 14, 2016

Contents

1	计算几何	2
1.1	二维	2
1.1.1	基础	2
1.1.2	凸包	2
2	数论	3
2.1	求逆元	3
2.2	中国剩余定理	3
3	字符串	3
3.1	后缀自动机	3
4	图论	3
4.1	基础	3
4.2	KM	4
4.3	点双连通分量	4
4.4	边双连通分量	5
5	技巧	5
5.1	真正的释放 STL 容器内存空间	5
5.2	无敌的大整数相乘取模	5
5.3	无敌的读入优化	5
5.4	控制 cout 输出实数精度	5

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 int sign(DB x) {
5     return x < -eps ? -1 : (x > eps ? 1 : 0);
6 }
7 DB msqrt(DB x) {
8     return sign(x) > 0 ? sqrt(x) : 0;
9 }
10
11 struct Point {
12     DB x, y;
13     Point(): x(0), y(0) {}
14     Point(DB x, DB y): x(x), y(y) {}
15     Point operator+(const Point &rhs) const {
16         return Point(x + rhs.x, y + rhs.y);
17     }
18     Point operator-(const Point &rhs) const {
19         return Point(x - rhs.x, y - rhs.y);
20     }
21     Point operator*(DB k) const {
22         return Point(x * k, y * k);
23     }
24     Point operator/(DB k) const {
25         assert(sign(k));
26         return Point(x / k, y / k);
27     }
28     Point rotate(DB ang) const { // 逆时针旋转 ang 弧度
29         return Point(cos(ang) * x - sin(ang) * y,
30                     cos(ang) * y + sin(ang) * x);
31     }
32     Point turn90() const { // 逆时针旋转 90 度
33         return Point(-y, x);
34     }
35 };
36 DB dot(const Point& a, const Point& b) {
37     return a.x * b.x + a.y * b.y;
38 }
39 DB det(const Point& a, const Point& b) {
40     return a.x * b.y - a.y * b.x;
41 }
42 bool isLL(const Line& l1, const Line& l2, Point& p) { // 直线与直线交点
43     DB s1 = det(l2.b - l2.a, l1.a - l2.a),
44         s2 = -det(l2.b - l2.a, l1.b - l2.a);
45     if (!sign(s1 + s2)) return false;
46     p = (l1.a * s2 + l1.b * s1) / (s1 + s2);
47     return true;
48 }
49 bool onSeg(const Line& l, const Point& p) { // 点在线段上
50     return sign(det(p - l.a, l.b - l.a)) == 0 && sign(dot(p - l.a, p - l.b)) <= 0;
51 }
52 DB disToLine(const Line& l, const Point& p) { // 点到直线距离
53     return fabs(det(p - l.a, l.b - l.a) / (l.b - l.a).len());
54 }
55 DB disToSeg(const Line& l, const Point& p) { // 点到线段距离
56     return sign(dot(p - l.a, l.b - l.a)) * sign(dot(p - l.b, l.a - l.b)) == 1 ?
57         disToLine(l, p) : std::min((p - l.a).len(), (p - l.b).len());
58 }
59 // 圆与直线交点
60 bool isCL(Circle a, Line l, Point& p1, Point& p2) {
61     DB x = dot(l.a - a.o, l.b - l.a),
62         y = (l.b - l.a).len2(),
63         d = x * x - y * ((l.a - a.o).len2() - a.r * a.r);
64     if (sign(d) < 0) return false;
```

```
64     Point p = l.a - ((l.b - l.a) * (x / y)), delta = (l.b - l.a) * (msqrt(d) / y);
65     p1 = p + delta; p2 = p - delta;
66     return true;
67 }
68 // 求凸包
69 std::vector<Point> convexHull(std::vector<Point> ps) {
70     int n = ps.size(); if (n <= 1) return ps;
71     std::sort(ps.begin(), ps.end());
72     std::vector<Point> qs;
73     for (int i = 0; i < n; qs.push_back(ps[i++]))
74         while (qs.size() > 1 && sign(det(qs[qs.size() - 2], qs.back(), ps[i])) <= 0)
75             qs.pop_back();
76     for (int i = n - 2, t = qs.size(); i >= 0; qs.push_back(ps[i--]))
77         while ((int)qs.size() > t && sign(det(qs[qs.size() - 2], qs.back(), ps[i])) <=
78             0)
79             qs.pop_back();
80     return qs;
```

1.1.2 凸包

```
1 // 凸包中的点按逆时针方向
2 struct Convex {
3     int n;
4     std::vector<Point> a, upper, lower;
5     void make_shell(const std::vector<Point>& p,
6                     std::vector<Point>& shell) { // p needs to be sorted.
7         clear(shell); int n = p.size();
8         for (int i = 0, j = 0; i < n; i++, j++) {
9             for (; j >= 2 && sign(det(shell[j-1] - shell[j-2],
10                                     p[i] - shell[j-2])) <= 0; --j) shell.pop_back();
11             shell.push_back(p[i]);
12         }
13     }
14     void make_convex() {
15         std::sort(a.begin(), a.end());
16         make_shell(a, lower);
17         std::reverse(a.begin(), a.end());
18         make_shell(a, upper);
19         a = lower; a.pop_back();
20         a.insert(a.end(), upper.begin(), upper.end());
21         if ((int)a.size() >= 2) a.pop_back();
22         n = a.size();
23     }
24     void init(const std::vector<Point>& _a) {
25         clear(a); a = _a; n = a.size();
26         make_convex();
27     }
28     void read(int _n) { // Won't make convex.
29         clear(a); n = _n; a.resize(n);
30         for (int i = 0; i < n; i++)
31             a[i].read();
32     }
33     std::pair<DB, int> get_tangent(
34         const std::vector<Point>& convex, const Point& vec) {
35         int l = 0, r = (int)convex.size() - 2;
36         assert(r >= 0);
37         for (; l + 1 < r; ) {
38             int mid = (l + r) / 2;
39             if (sign(det(convex[mid + 1] - convex[mid], vec)) > 0)
40                 r = mid;
41             else l = mid;
42         }
43         return std::max(std::make_pair(det(vec, convex[r]), r),
44                         std::make_pair(det(vec, convex[0]), 0));
45     }
46     int binary_search(Point u, Point v, int l, int r) {
47         int s1 = sign(det(v - u, a[l % n] - u));
48         for (; l + 1 < r; ) {
49             int mid = (l + r) / 2;
50             int smid = sign(det(v - u, a[mid % n] - u));
```

```

51     if (smid == s1) l = mid;
52     else r = mid;
53 }
54 return l % n;
55 }
56 // 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
57 int get_tangent(Point vec) {
58     std::pair<DB, int> ret = get_tangent(upper, vec);
59     ret.second = (ret.second + (int)lower.size() - 1) % n;
60     ret = std::max(ret, get_tangent(lower, vec));
61     return ret.second;
62 }
63 // 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
64 bool get_intersection(Point u, Point v, int &i0, int &i1) {
65     int p0 = get_tangent(u - v), p1 = get_tangent(v - u);
66     if (sign(det(v - u, a[p0] - u)) * sign(det(v - u, a[p1] - u)) <= 0) {
67         if (p0 > p1) std::swap(p0, p1);
68         i0 = binary_search(u, v, p0, p1);
69         i1 = binary_search(u, v, p1, p0 + n);
70         return true;
71     }
72     else return false;
73 }
74 };

```

Chapter 2 数论

2.1 求逆元

```

1 void ex_gcd(long long a, long long b, long long &x, long long &y) {
2     if (b == 0) {
3         x = 1;
4         y = 0;
5         return;
6     }
7     long long xx, yy;
8     ex_gcd(b, a % b, xx, yy);
9     y = xx - a / b * yy;
10    x = yy;
11 }
12
13 long long inv(long long x, long long MODN) {
14     long long inv_x, y;
15     ex_gcd(x, MODN, inv_x, y);
16     return (inv_x % MODN + MODN) % MODN;
17 }

```

2.2 中国剩余定理

```

1 // 返回 (ans, M), 其中 ans 是模 M 意义下的解
2 std::pair<long long, long long> CRT(const std::vector<long long>& m, const
    交点, 交在点上不确定返回前后两条边其中之一
3     std::vector<long long, long long>& a) {
4     long long M = 1, ans = 0;
5     int n = m.size();
6     for (int i = 0; i < n; i++) M *= m[i];
7     for (int i = 0; i < n; i++) {
8         ans = (ans + (M / m[i]) * a[i] % M * inv(M / m[i], m[i])) % M; // 可能需要大
        整数相乘取模
9     }
10    return std::make_pair(ans, M);

```

Chapter 3 字符串

3.1 后缀自动机

```

1 struct Sam {
2     static const int MAXL = MAXN * 2; // MAXN is original length

```

```

3     static const int alphabet = 26; // sometimes need changing
4     int l, last, cnt, trans[MAXL][alphabet], par[MAXL], sum[MAXL], seq[MAXL],
    交点, 交在点上不确定返回前后两条边其中之一
5     mxl[MAXL], size[MAXL]; // mxl is maxlength, size is the size of right
6     char str[MAXL];
7     inline void init() {
8         l = strlen(str + 1); cnt = last = 1;
9         for (int i = 0; i <= l * 2; ++i) memset(trans[i], 0, sizeof(trans[i]));
10        memset(par, 0, sizeof(*par) * (l * 2 + 1));
11        memset(mxl, 0, sizeof(*mxl) * (l * 2 + 1));
12        memset(size, 0, sizeof(*size) * (l * 2 + 1));
13    }
14    inline void extend(int pos, int c) {
15        int p = last, np = last = ++cnt;
16        mxl[np] = mxl[p] + 1; size[np] = 1;
17        for (; p && !trans[p][c]; p = par[p]) trans[p][c] = np;
18        if (!p) par[np] = 1;
19        else {
20            int q = trans[p][c];
21            if (mxl[p] + 1 == mxl[q]) par[np] = q;
22            else {
23                int nq = ++cnt;
24                mxl[nq] = mxl[p] + 1;
25                memcpy(trans[nq], trans[q], sizeof(trans[nq]));
26                par[nq] = par[q];
27                par[np] = par[q] = nq;
28                for (; trans[p][c] == q; p = par[p]) trans[p][c] = nq;
29            }
30        }
31    }
32    inline void buildsam() {
33        for (int i = 1; i <= l; ++i) extend(i, str[i] - 'a');
34        memset(sum, 0, sizeof(*sum) * (l * 2 + 1));
35        for (int i = 1; i <= cnt; ++i) sum[mxl[i]]++;
36        for (int i = 1; i <= l; ++i) sum[i] += sum[i - 1];
37        for (int i = cnt; i; --i) seq[sum[mxl[i]] - 1] = i;
38        for (int i = cnt; i; --i) size[par[seq[i]]] += size[seq[i]];
39    }
40 } sam;

```

Chapter 4 图论

4.1 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    __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 };

```

4.2 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,
4     // 使用时先对  $1 \rightarrow n$  调用 hungary(), 再 get_ans() 求值
5     int w[N][N];
6     int lx[N], ly[N], match[N], way[N], slack[N];
7     bool used[N];
8     void init() {
9         for (int i = 1; i <= n; i++) {
10             match[i] = 0;
11             lx[i] = 0;
12             ly[i] = 0;
13             way[i] = 0;
14         }
15     }
16     void hungary(int x) {
17         match[0] = x;
18         int j0 = 0;
19         for (int j = 0; j <= n; j++) {
20             slack[j] = INF;
21             used[j] = false;
22         }
23         do {
24             used[j0] = true;
25             int i0 = match[j0], delta = INF, j1 = 0;
26             for (int j = 1; j <= n; j++) {
27                 if (used[j] == false) {
28                     int cur = -w[i0][j] - lx[i0] - ly[j];
29                     if (cur < slack[j]) {
30                         slack[j] = cur;
31                         way[j] = j0;
32                     }
33                     if (slack[j] < delta) {
34                         delta = slack[j];
35                         j1 = j;
36                     }
37                 }
38             }
39             for (int j = 0; j <= n; j++) {
40                 if (used[j]) {
41                     lx[match[j]] += delta;
42                     ly[j] -= delta;
43                 }
44                 else slack[j] -= delta;
45             }
46             j0 = j1;
47             while (match[j0] != 0);
48             do {
49                 int j1 = way[j0];
50                 match[j0] = match[j1];
51                 j0 = j1;
52             } while (j0);
53         } while (j0);
54     }
55     int get_ans() {
56         int sum = 0;
57         for (int i = 1; i <= n; i++) {
58             if (w[match[i]][i] == -INF) ; // 无解
59             if (match[i] > 0) sum += w[match[i]][i];
60         }
61         return sum;
62     }
63 } km;

```

4.3 点双连通分量

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

```

1 const bool BCC_VERTEX = 0, BCC_EDGE = 1;
2 struct BCC { // 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> BCC_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                    //BCC_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                                //BCC_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 } bcc;
74
75 bcc.init(&raw_graph);
76 bcc.solve();
77 // Do something with bcc.forest ...

```

4.4 边双连通分量

```

1 struct BCC {
2     Graph *g, forest;
3     int dfn[N], low[N], stack[N], tot[N], belong[N], vis[N], top, dfs_clock;
4     // tot[] is the size of each BCC, belong[] is the BCC that each node belongs to
5     pair<int, int> ori[M]; // bridge in raw_graph(raw node)
6     bool is_bridge[M];
7     __inline void init(Graph *raw_graph) {
8         g = raw_graph;
9         memset(is_bridge, false, sizeof(*is_bridge) * g -> e);
10        memset(vis + g -> base, 0, sizeof(*vis) * g -> n);
11    }
12    void tarjan(int u, int from) {
13        dfn[u] = low[u] = ++dfs_clock; vis[u] = 1; stack[++top] = u;
14        for (int p = g -> adj[u]; ~p; p = g -> nxt[p]) {
15            if ((p ^ 1) == from) continue;
16            int v = g -> v[p];
17            if (vis[v]) {
18                if (vis[v] == 1) low[u] = min(low[u], dfn[v]);
19            } else {
20                tarjan(v, p);
21                low[u] = min(low[u], low[v]);
22                if (low[v] > dfn[u]) is_bridge[p / 2] = true;
23            }
24        }
25        if (dfn[u] != low[u]) return;
26        tot[forest.new_node()] = 0;
27        do {
28            belong[stack[top]] = forest.n;
29            vis[stack[top]] = 2;
30            tot[forest.n]++;
31            --top;
32        } while (stack[top + 1] != u);
33    }
34    void solve() {
35        forest.init(g -> base);
36        int n = g -> n;
37        for (int i = 0; i < n; ++i)
38            if (!vis[i + g -> base]) {
39                top = dfs_clock = 0;
40                tarjan(i + g -> base, -1);
41            }
42        for (int i = 0; i < g -> e / 2; ++i)
43            if (is_bridge[i]) {
44                int e = forest.e;
45                forest.bi_ins(belong[g -> v[i * 2]], belong[g -> v[i * 2 + 1]], g ->
46                w[i * 2]);
47                ori[e] = make_pair(g -> v[i * 2 + 1], g -> v[i * 2]);
48                ori[e + 1] = make_pair(g -> v[i * 2], g -> v[i * 2 + 1]);
49            }

```

```

49     }
50 } bcc;

```

Chapter 5 技巧

5.1 真正的释放 STL 容器内存空间

```

1 template <typename T>
2 __inline void clear(T& container) {
3     container.clear(); // 或者删除了一堆元素
4     T(container).swap(container);
5 }

```

5.2 无敌的大整数相乘取模

Time complexity $O(1)$.

```

1 // 需要保证 x 和 y 非负
2 long long mult(long long x, long long y, long long MODN) {
3     long long t = (x * y - (long long)((long double)x / MODN * y + 1e-3) * MODN) %
4     MODN;
5     return t < 0 ? t + MODN : t;
6 }

```

5.3 无敌的读入优化

```

1 // getchar() 读入优化 << 关同步 cin << 此优化
2 // 用 isdigit() 会小幅变慢
3 namespace Reader {
4     const int L = (1 << 15) + 5;
5     char buffer[L], *S, *T;
6     __inline void get_char(char &ch) {
7         if (S == T) {
8             T = (S = buffer) + fread(buffer, 1, L, stdin);
9             if (S == T) {
10                 ch = EOF;
11                 return;
12             }
13         }
14         ch = *S++;
15     }
16     __inline void get_int(int &x) {
17         char ch; bool neg = 0;
18         for (; get_char(ch), ch < '0' || ch > '9'; ) neg ^= ch == '-';
19         x = ch - '0';
20         for (; get_char(ch), ch >= '0' && ch <= '9'; )
21             x = x * 10 + ch - '0';
22         if (neg) x = -x;
23     }
24 }

```

5.4 控制 cout 输出实数精度

```

1 std::cout << std::fixed << std::setprecision(5);

```