Gungnir's Standard Code Library

Shanghai Jiao Tong University

Dated: October 10, 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;

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

struct Point {
    DB x, y;
    Point(): x(0), y(0) {}
    Point(DB x, DB y): x(x), y(y) {}
    Point operator+(const Point &rhs) const {
        return Point(x + rhs.x, y + rhs.y);
    }
    Point operator-(const Point &rhs) const {
        return Point(x - rhs.x, y - rhs.y);
    }
    Point operator*(DB k) const {
        return Point(x * k, y * k);
    }
    Point operator/(DB k) const {
        assert(sign(k));
        return Point(x / k, y / k);
    }
    Point rotate(DB ang) const {  // 逆时针旋转 ang 弧度
        return Point(cos(ang) * x - sin(ang) * y,
                cos(ang) * y + sin(ang) * x);
    }
    Point turn90() const {  // 逆时针旋转 90 度
        return Point(-y, x);
    }
    Point unit() const {
        return *this / len();
    }
};
DB dot(const Point& a, const Point& b) {
    return a.x * b.x + a.y * b.y;
}
DB det(const Point& a, const Point& b) {
    return a.x * b.y - a.y * b.x;
}
#define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))
#define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))
bool isLL(const Line& l1, const Line& l2, Point& p) {  // 直线与直线交点
    DB s1 = det(l2.b - l2.a, l1.a - l2.a),
        s2 = -det(l2.b - l2.a, l1.b - l2.a);
    if (!sign(s1 + s2)) return false;
    p = (l1.a * s2 + l1.b * s1) / (s1 + s2);
    return true;
}
bool onSeg(const Line& l, const Point& p) {  // 点在线段上
    return sign(det(p - l.a, l.b - l.a)) == 0 && sign(dot(p - l.a, p - l.b)) <= 0;
}
Point projection(const Line & l, const Point& p) {
    return l.a + (l.b - l.a) * (dot(p - l.a, l.b - l.a) / (l.b - l.a).len2());
}
DB disToLine(const Line& l, const Point& p) {  // 点到 * 直线 * 距离
    return fabs(det(p - l.a, l.b - l.a) / (l.b - l.a).len());
}
DB disToSeg(const Line& l, const Point& p) {  // 点到线段距离
    return sign(dot(p - l.a, l.b - l.a)) * sign(dot(p - l.b, l.a - l.b)) == 1 ?
     ↪ disToLine(l, p) : std::min((p - l.a).len(), (p - l.b).len());
}
```

```cpp
}
// 圆与直线交点
bool isCL(Circle a, Line l, Point& p1, Point& p2) {
    DB x = dot(l.a - a.o, l.b - l.a),
        y = (l.b - l.a).len2(),
        d = x * x - y * ((l.a - a.o).len2() - a.r * a.r);
    if (sign(d) < 0) return false;
    Point p = l.a - ((l.b - l.a) * (x / y)), delta = (l.b - l.a) * (msqrt(d) / y);
    p1 = p + delta; p2 = p - delta;
    return true;
}
//圆与圆的交面积
DB areaCC(const Circle& c1, const Circle& c2) {
    DB d = (c1.o - c2.o).len();
    if (sign(d - (c1.r + c2.r)) >= 0) return 0;
    if (sign(d - std::abs(c1.r - c2.r)) <= 0) {
        DB r = std::min(c1.r, c2.r);
        return r * r * PI;
    }
    DB x = (d * d + c1.r * c1.r - c2.r * c2.r) / (2 * d),
        t1 = acos(x / c1.r), t2 = acos((d - x) / c2.r);
    return c1.r * c1.r * t1 + c2.r * c2.r * t2 - d * c1.r * sin(t1);
}
// 圆与圆交点
bool isCC(Circle a, Circle b, P& p1, P& p2) {
    DB s1 = (a.o - b.o).len();
    if (sign(s1 - a.r - b.r) > 0 || sign(s1 - std::abs(a.r - b.r)) < 0) return false;
    DB s2 = (a.r * a.r - b.r * b.r) / s1;
    DB aa = (s1 + s2) * 0.5, bb = (s1 - s2) * 0.5;
    P o = (b.o - a.o) * (aa / (aa + bb)) + a.o;
    P delta = (b.o - a.o).unit().turn90() * msqrt(a.r * a.r - aa * aa);
    p1 = o + delta, p2 = o - delta;
    return true;
}
// 求点到圆的切点，按关于点的顺时针方向返回两个点
bool tanCP(const Circle &c, const Point &p0, Point &p1, Point &p2) {
    double x = (p0 - c.o).len2(), d = x - c.r * c.r;
    if (d < eps) return false; // 点在圆上认为没有切点
    Point p = (p0 - c.o) * (c.r * c.r / x);
    Point delta = ((p0 - c.o) * (-c.r * sqrt(d) / x)).turn90();
    p1 = c.o + p + delta;
    p2 = c.o + p - delta;
    return true;
}
// 求圆到圆的内共切线，按关于 c1.o 的顺时针方向返回两条
std::vector<Line> intanCC(const Circle &c1, const Circle &c2) {
    std::vector<Line> ret;
    Point p = (c1.o * c2.r + c2.o * c1.r) / (c1.r + c2.r);
    Point p1, p2, q1, q2;
    if (tanCP(c1, p, p1, p2) && tanCP(c2, p, q1, q2)) { // 两圆相切认为没有切线
        ret.push_back(Line(p1, q1));
        ret.push_back(Line(p2, q2));
    }
    return ret;
}
// 点在多边形内
bool inPolygon(const Point& p, const std::vector<Point>& poly) {
    int n = poly.size();
    int counter = 0;
    for (int i = 0; i < n; ++ i) {
        P a = poly[i], b = poly[(i + 1) % n];
        if (onSeg(Line(a, b), p)) return false; // 边界上不算
        int x = sign(det(p - a, b - a));
        int y = sign(a.y - p.y);
        int z = sign(b.y - p.y);
        if (x > 0 && y <= 0 && z > 0) ++ counter;
        if (x < 0 && z <= 0 && y > 0) -- counter;
    }
    return counter != 0;
}
```

```
135  // 用半平面 (q1,q2) 的逆时针方向去切凸多边形
136  std::vector<Point> convexCut(const std::vector<Point>&ps, Point q1, Point q2) {
137      std::vector<Point> qs; int n = ps.size();
138      for (int i = 0; i < n; ++i) {
139          Point p1 = ps[i], p2 = ps[(i + 1) % n];
140          int d1 = crossOp(q1,q2,p1), d2 = crossOp(q1,q2,p2);
141          if (d1 >= 0) qs.push_back(p1);
142          if (d1 * d2 < 0) qs.push_back(isSS(p1, p2, q1, q2));
143      }
144      return qs;
145  }
146  // 求凸包
147  std::vector<Point> convexHull(std::vector<Point> ps) {
148      int n = ps.size(); if (n <= 1) return ps;
149      std::sort(ps.begin(), ps.end());
150      std::vector<Point> qs;
151      for (int i = 0; i < n; qs.push_back(ps[i ++]))
152          while (qs.size() > 1 && sign(det(qs[qs.size() - 2], qs.back(), ps[i])) <= 0)
153              qs.pop_back();
154      for (int i = n - 2, t = qs.size(); i >= 0; qs.push_back(ps[i --]))
155          while ((int)qs.size() > t && sign(det(qs[qs.size() - 2], qs.back(), ps[i])) <=
     ↪ 0)
156              qs.pop_back();
157      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  };
```

1.2 三维
1.2.1 基础

```
1   // 三维绕轴旋转，大拇指指向 axis 向量方向，四指弯曲方向转 w 弧度
2   Point rotate(const Point& s, const Point& axis, DB w) {
3       DB x = axis.x, y = axis.y, z = axis.z;
4       DB s1 = x * x + y * y + z * z, ss1 = msqrt(s1),
5           cosw = cos(w), sinw = sin(w);
6       DB a[4][4];
7       memset(a, 0, sizeof a);
8       a[3][3] = 1;
9       a[0][0] = ((y * y + z * z) * cosw + x * x) / s1;
10      a[0][1] = x * y * (1 - cosw) / s1 + z * sinw / ss1;
11      a[0][2] = x * z * (1 - cosw) / s1 - y * sinw / ss1;
12      a[1][0] = x * y * (1 - cosw) / s1 - z * sinw / ss1;
13      a[1][1] = ((x * x + z * z) * cosw + y * y) / s1;
14      a[1][2] = y * z * (1 - cosw) / s1 + x * sinw / ss1;
15      a[2][0] = x * z * (1 - cosw) / s1 + y * sinw / ss1;
16      a[2][1] = y * z * (1 - cosw) / s1 - x * sinw / ss1;
17      a[2][2] = ((x * x + y * y) * cos(w) + z * z) / s1;
18      DB ans[4] = {0, 0, 0, 0}, c[4] = {s.x, s.y, s.z, 1};
19      for (int i = 0; i < 4; ++ i)
20          for (int j = 0; j < 4; ++ j)
21              ans[i] += a[j][i] * c[j];
22      return Point(ans[0], ans[1], ans[2]);
23  }
```

Chapter 2 数论
2.1 $O(m^2 \log n)$ 求线性递推数列第 n 项

Given $a_0, a_1, \ldots, a_{m-1}$
$a_n = c_0 \times a_{n-m} + \cdots + c_{m-1} \times a_{n-1}$
Solve for $a_n = v_0 \times a_0 + v_1 \times a_1 + \cdots + v_{m-1} \times a_{m-1}$

```
1   void linear_recurrence(long long n, int m, int a[], int c[], int p) {
2       long long v[M] = {1 % p}, u[M << 1], msk = !!n;
```

```
 3      for(long long i(n); i > 1; i >>= 1) {
 4          msk <<= 1;
 5      }
 6      for(long long x(0); msk; msk >>= 1, x <<= 1) {
 7          fill_n(u, m << 1, 0);
 8          int b(!!(n & msk));
 9          x |= b;
10          if(x < m) {
11              u[x] = 1 % p;
12          }else {
13              for(int i(0); i < m; i++) {
14                  for(int j(0), t(i + b); j < m; j++, t++) {
15                      u[t] = (u[t] + v[i] * v[j]) % p;
16                  }
17              }
18              for(int i((m << 1) - 1); i >= m; i--) {
19                  for(int j(0), t(i - m); j < m; j++, t++) {
20                      u[t] = (u[t] + c[j] * u[i]) % p;
21                  }
22              }
23          }
24          copy(u, u + m, v);
25      }
26      //a[n] = v[0] * a[0] + v[1] * a[1] + ... + v[m - 1] * a[m - 1].
27      for(int i(m); i < 2 * m; i++) {
28          a[i] = 0;
29          for(int j(0); j < m; j++) {
30              a[i] = (a[i] + (long long)c[j] * a[i + j - m]) % p;
31          }
32      }
33      for(int j(0); j < m; j++) {
34          b[j] = 0;
35          for(int i(0); i < m; i++) {
36              b[j] = (b[j] + v[i] * a[i + j]) % p;
37          }
38      }
39      for(int j(0); j < m; j++) {
40          a[j] = b[j];
41      }
42  }
```

2.2 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  }
```

2.3 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
 ↪  std::vector<long long>& a) {
 3      long long M = 1, ans = 0;
 4      int n = m.size();
 5      for (int i = 0; i < n; i++) M *= m[i];
 6      for (int i = 0; i < n; i++) {
```

```
 7          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
 8      }
 9      return std::make_pair(ans, M);
10  }
```

Chapter 3 代数

3.1 快速傅里叶变换

```
 1  // n 必须是 2 的次幂
 2  void fft(Complex a[], int n, int f) {
 3      for (int i = 0; i < n; ++i)
 4          if (R[i] < i) swap(a[i], a[R[i]]);
 5      for (int i = 1, h = 0; i < n; i <<= 1, h++) {
 6          Complex wn = Complex(cos(pi / i), f * sin(pi / i));
 7          Complex w = Complex(1, 0);
 8          for (int k = 0; k < i; ++k, w = w * wn) tmp[k] = w;
 9          for (int p = i << 1, j = 0; j < n; j += p) {
10              for (int k = 0; k < i; ++k) {
11                  Complex x = a[j + k], y = a[j + k + i] * tmp[k];
12                  a[j + k] = x + y; a[j + k + i] = x - y;
13              }
14          }
15      }
16  }
```

Chapter 4 字符串

4.1 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],
 ↪  mxl[MAXL], size[MAXL];   // mxl is maxlength, size is the size of right
 5      char str[MAXL];
 6      inline void init() {
 7          l = strlen(str + 1); cnt = last = 1;
 8          for (int i = 0; i <= l * 2; ++i) memset(trans[i], 0, sizeof(trans[i]));
 9          memset(par, 0, sizeof(*par) * (l * 2 + 1));
10          memset(mxl, 0, sizeof(*mxl) * (l * 2 + 1));
11          memset(size, 0, sizeof(*size) * (l * 2 + 1));
12      }
13      inline void extend(int pos, int c) {
14          int p = last, np = last = ++cnt;
15          mxl[np] = mxl[p] + 1; size[np] = 1;
16          for (; p && !trans[p][c]; p = par[p]) trans[p][c] = np;
17          if (!p) par[np] = 1;
18          else {
19              int q = trans[p][c];
20              if (mxl[p] + 1 == mxl[q]) par[np] = q;
21              else {
22                  int nq = ++cnt;
23                  mxl[nq] = mxl[p] + 1;
24                  memcpy(trans[nq], trans[q], sizeof(trans[nq]));
25                  par[nq] = par[q];
26                  par[np] = par[q] = nq;
27                  for (; trans[p][c] == q; p = par[p]) trans[p][c] = nq;
28              }
29          }
30      }
31      inline void buildsam() {
32          for (int i = 1; i <= l; ++i) extend(i, str[i] - 'a');
33          memset(sum, 0, sizeof(*sum) * (l * 2 + 1));
34          for (int i = 1; i <= cnt; ++i) sum[mxl[i]]++;
35          for (int i = 1; i <= l; ++i) sum[i] += sum[i - 1];
36          for (int i = cnt; i; --i) seq[sum[mxl[i]]--] = i;
37          for (int i = cnt; i; --i) size[par[seq[i]]] += size[seq[i]];
```

```
38          }
39  } sam;
```

Chapter 5 图论

5.1 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  };
```

5.2 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，
        ↪ 使用时先对 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 init() {
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) {
16          match[0] = x;
17          int j0 = 0;
18          for (int j = 0; j <= n; j++) {
19              slack[j] = INF;
20              used[j] = false;
21          }
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
49          do {
50              int j1 = way[j0];
51              match[j0] = match[j1];
52              j0 = j1;
53          } while (j0);
54      }
55
56      int get_ans() {
57          int sum = 0;
58          for(int i = 1; i <= n; i++) {
59              if (w[match[i]][i] == -INF) ; // 无解
60              if (match[i] > 0) sum += w[match[i]][i];
61          }
62          return sum;
63      }
64  } km;
```

5.3 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        }
74    } bcc;
75
76    bcc.init(&raw_graph);
77    bcc.solve();
78    // Do something with bcc.forest ...
```

5.4 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 ->
   w[i * 2]);
46                 ori[e] = make_pair(g -> v[i * 2 + 1], g -> v[i * 2]);
47                 ori[e + 1] = make_pair(g -> v[i * 2], g -> v[i * 2 + 1]);
48             }
49     }
50 } bcc;
```

Chapter 6 技巧

6.1 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  }
```

6.2 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) %
   MODN;
4      return t < 0 ? t + MODN : t;
5  }
```

6.3 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  // 返回 false 表示读到文件尾
4  namespace Reader {
5      const int L = (1 << 15) + 5;
6      char buffer[L], *S, *T;
7      __inline bool getchar(char &ch) {
8          if (S == T) {
9              T = (S = buffer) + fread(buffer, 1, L, stdin);
10             if (S == T) {
11                 ch = EOF;
12                 return false;
13             }
14         }
15         ch = *S++;
16         return true;
17     }
18     __inline bool getint(int &x) {
19         char ch; bool neg = 0;
20         for (; getchar(ch) && (ch < '0' || ch > '9'); ) neg ^= ch == '-';
21         if (ch == EOF) return false;
22         x = ch - '0';
23         for (; getchar(ch), ch >= '0' && ch <= '9'; )
```

```
24              x = x * 10 + ch - '0';
25          if (neg) x = -x;
26          return true;
27      }
28  }
```

6.4 控制 cout 输出实数精度

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

Chapter 7 提示
7.1 线性规划转对偶

$$
\begin{array}{c}
\text{maximize } \mathbf{c}^T\mathbf{x} \\
\text{subject to } \mathbf{A}\mathbf{x} \le \mathbf{b}, \mathbf{x} \ge 0
\end{array}
\iff
\begin{array}{c}
\text{minimize } \mathbf{y}^T\mathbf{b} \\
\text{subject to } \mathbf{y}^T\mathbf{A} \ge \mathbf{c}^T, \mathbf{y} \ge 0
\end{array}
$$

7.2 32-bit/64-bit 随机素数

32-bit	64-bit
73550053	12492928468556885773
148898719	17017504344419805569
189560747	3605499878424114901
459874703	5648316673387803781
1202316001	6125342570814357977
1431183547	6215155308775851301
1438011109	6294606778040623451
1538762023	6347330550446020547
1557944263	7429632924303725207
1981315913	8524720079480389849

7.3 NTT 素数及其原根

Prime	Primitive root
1053818881	7
1051721729	6
1045430273	3
1012924417	5
1007681537	3