Gungnir'l Standard Code Library[*]

Shanghai Jiao Tong University

Dated: November 20, 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);
    }
```

```
32      Point turn90() const {  // 逆时针旋转 90 度
33          return Point(-y, x);
34      }
35      Point unit() const {
36          return *this / len();
37      }
38  };
39  DB dot(const Point& a, const Point& b) {
40      return a.x * b.x + a.y * b.y;
41  }
42  DB det(const Point& a, const Point& b) {
43      return a.x * b.y - a.y * b.x;
44  }
45  #define cross(p1,p2,p3) ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))
46  #define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))
47  bool isLL(const Line& l1, const Line& l2, Point& p) {  // 直线与直线交点
48      DB s1 = det(l2.b - l2.a, l1.a - l2.a),
49         s2 = -det(l2.b - l2.a, l1.b - l2.a);
50      if (!sign(s1 + s2)) return false;
51      p = (l1.a * s2 + l1.b * s1) / (s1 + s2);
52      return true;
53  }
54  bool onSeg(const Line& l, const Point& p) {  // 点在线段上
55      return sign(det(p - l.a, l.b - l.a)) == 0 && sign(dot(p - l.a, p - l.b)) <= 0;
56  }
57  Point projection(const Line & l, const Point& p) {
58      return l.a + (l.b - l.a) * (dot(p - l.a, l.b - l.a) / (l.b - l.a).len2());
59  }
60  DB disToLine(const Line& l, const Point& p) {  // 点到 * 直线 * 距离
61      return fabs(det(p - l.a, l.b - l.a) / (l.b - l.a).len());
62  }
63  DB disToSeg(const Line& l, const Point& p) {  // 点到线段距离
64      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());
65  }
66  // 圆与直线交点
67  bool isCL(Circle a, Line l, Point& p1, Point& p2) {
68      DB x = dot(l.a - a.o, l.b - l.a),
69         y = (l.b - l.a).len2(),
70         d = x * x - y * ((l.a - a.o).len2() - a.r * a.r);
71      if (sign(d) < 0) return false;
72      Point p = l.a - ((l.b - l.a) * (x / y)), delta = (l.b - l.a) * (msqrt(d) / y);
73      p1 = p + delta; p2 = p - delta;
74      return true;
75  }
76  //圆与圆的交面积
77  DB areaCC(const Circle& c1, const Circle& c2) {
78      DB d = (c1.o - c2.o).len();
79      if (sign(d - (c1.r + c2.r)) >= 0) return 0;
```

```cpp
80        if (sign(d - std::abs(c1.r - c2.r)) <= 0) {
81            DB r = std::min(c1.r, c2.r);
82            return r * r * PI;
83        }
84        DB x = (d * d + c1.r * c1.r - c2.r * c2.r) / (2 * d),
85            t1 = acos(x / c1.r), t2 = acos((d - x) / c2.r);
86        return c1.r * c1.r * t1 + c2.r * c2.r * t2 - d * c1.r * sin(t1);
87   }
88   // 圆与圆交点
89   bool isCC(Circle a, Circle b, P& p1, P& p2) {
90       DB s1 = (a.o - b.o).len();
91       if (sign(s1 - a.r - b.r) > 0 || sign(s1 - std::abs(a.r - b.r)) < 0) return
     ↪ false;
92       DB s2 = (a.r * a.r - b.r * b.r) / s1;
93       DB aa = (s1 + s2) * 0.5, bb = (s1 - s2) * 0.5;
94       P o = (b.o - a.o) * (aa / (aa + bb)) + a.o;
95       P delta = (b.o - a.o).unit().turn90() * msqrt(a.r * a.r - aa * aa);
96       p1 = o + delta, p2 = o - delta;
97       return true;
98   }
99   // 求点到圆的切点，按关于点的顺时针方向返回两个点
100  bool tanCP(const Circle &c, const Point &p0, Point &p1, Point &p2) {
101      double x = (p0 - c.o).len2(), d = x - c.r * c.r;
102      if (d < eps) return false; // 点在圆上认为没有切点
103      Point p = (p0 - c.o) * (c.r * c.r / x);
104      Point delta = ((p0 - c.o) * (-c.r * sqrt(d) / x)).turn90();
105      p1 = c.o + p + delta;
106      p2 = c.o + p - delta;
107      return true;
108  }
109  // 求圆到圆的内共切线，按关于 c1.o 的顺时针方向返回两条线
110  std::vector<Line> intanCC(const Circle &c1, const Circle &c2) {
111      std::vector<Line> ret;
112      Point p = (c1.o * c2.r + c2.o * c1.r) / (c1.r + c2.r);
113      Point p1, p2, q1, q2;
114      if (tanCP(c1, p, p1, p2) && tanCP(c2, p, q1, q2)) { // 两圆相切认为没有切线
115          ret.push_back(Line(p1, q1));
116          ret.push_back(Line(p2, q2));
117      }
118      return ret;
119  }
120  // 点在多边形内
121  bool inPolygon(const Point& p, const std::vector<Point>& poly) {
122      int n = poly.size();
123      int counter = 0;
124      for (int i = 0; i < n; ++ i) {
125          P a = poly[i], b = poly[(i + 1) % n];
126          if (onSeg(Line(a, b), p)) return false; // 边界上不算
127          int x = sign(det(p - a, b - a));
```

```
128        int y = sign(a.y - p.y);
129        int z = sign(b.y - p.y);
130        if (x > 0 && y <= 0 && z > 0) ++ counter;
131        if (x < 0 && z <= 0 && y > 0) -- counter;
132    }
133    return counter != 0;
134 }
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 }
```

1.2.2 凸包

```
1  __inline P cross(const P& a, const P& b) {
2      return P(
3          a.y * b.z - a.z * b.y,
4          a.z * b.x - a.x * b.z,
5          a.x * b.y - a.y * b.x
6      );
```

```
7   }
8
9   __inline DB mix(const P& a, const P& b, const P& c) {
10      return dot(cross(a, b), c);
11  }
12
13  __inline DB volume(const P& a, const P& b, const P& c, const P& d) {
14      return mix(b - a, c - a, d - a);
15  }
16
17  struct Face {
18      int a, b, c;
19      __inline Face() {}
20      __inline Face(int _a, int _b, int _c):
21          a(_a), b(_b), c(_c) {}
22      __inline DB area() const {
23          return 0.5 * cross(p[b] - p[a], p[c] - p[a]).len();
24      }
25      __inline P normal() const {
26          return cross(p[b] - p[a], p[c] - p[a]).unit();
27      }
28      __inline DB dis(const P& p0) const {
29          return dot(normal(), p0 - p[a]);
30      }
31  };
32
33  std::vector<Face> face, tmp;  // Should be O(n).
34  int mark[N][N], Time, n;
35
36  __inline void add(int v) {
37      ++ Time;
38      clear(tmp);
39      for (int i = 0; i < (int)face.size(); ++ i) {
40          int a = face[i].a, b = face[i].b, c = face[i].c;
41          if (sign(volume(p[v], p[a], p[b], p[c])) > 0) {
42              mark[a][b] = mark[b][a] = mark[a][c] =
43                  mark[c][a] = mark[b][c] = mark[c][b] = Time;
44          }
45          else {
46              tmp.push_back(face[i]);
47          }
48      }
49      clear(face); face = tmp;
50      for (int i = 0; i < (int)tmp.size(); ++ i) {
51          int a = face[i].a, b = face[i].b, c = face[i].c;
52          if (mark[a][b] == Time) face.emplace_back(v, b, a);
53          if (mark[b][c] == Time) face.emplace_back(v, c, b);
54          if (mark[c][a] == Time) face.emplace_back(v, a, c);
55          assert(face.size() < 500u);
```

```
56        }
57  }
58
59  void reorder() {
60      for (int i = 2; i < n; ++ i) {
61          P tmp = cross(p[i] - p[0], p[i] - p[1]);
62          if (sign(tmp.len())) {
63              std::swap(p[i], p[2]);
64              for (int j = 3; j < n; ++ j)
65                  if (sign(volume(p[0], p[1], p[2], p[j]))) {
66                      std::swap(p[j], p[3]);
67                      return;
68                  }
69          }
70      }
71  }
72
73  void build_convex() {
74      reorder();
75      clear(face);
76      face.emplace_back(0, 1, 2);
77      face.emplace_back(0, 2, 1);
78      for (int i = 3; i < n; ++ i)
79          add(i);
80  }
```

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 $a_n = v_0 \times a_0 + v_1 \times a_1 + \cdots + v_{m-1} \times a_{m-1}$

```cpp
void linear_recurrence(long long n, int m, int a[], int c[], int p) {
    long long v[M] = {1 % p}, u[M << 1], msk = !!n;
    for(long long i(n); i > 1; i >>= 1) {
        msk <<= 1;
    }
    for(long long x(0); msk; msk >>= 1, x <<= 1) {
        fill_n(u, m << 1, 0);
        int b(!!(n & msk));
        x |= b;
        if(x < m) {
            u[x] = 1 % p;
        }else {
            for(int i(0); i < m; i++) {
                for(int j(0), t(i + b); j < m; j++, t++) {
                    u[t] = (u[t] + v[i] * v[j]) % p;
                }
            }
            for(int i((m << 1) - 1); i >= m; i--) {
                for(int j(0), t(i - m); j < m; j++, t++) {
                    u[t] = (u[t] + c[j] * u[i]) % p;
                }
            }
        }
        copy(u, u + m, v);
    }
    //a[n] = v[0] * a[0] + v[1] * a[1] + ... + v[m - 1] * a[m - 1].
    for(int i(m); i < 2 * m; i++) {
        a[i] = 0;
        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 }
```

2.4   素性测试

```
1  int strong_pseudo_primetest(long long n,int base) {
2      long long n2=n-1,res;
3      int s=0;
4      while(n2%2==0) n2>>=1,s++;
5      res=powmod(base,n2,n);
6      if((res==1)||(res==n-1)) return 1;
7      s--;
8      while(s>=0) {
9          res=mulmod(res,res,n);
10         if(res==n-1) return 1;
11         s--;
12     }
13     return 0; // n is not a strong pseudo prime
14 }
15 int isprime(long long n) {
16     static LL testNum[]={2,3,5,7,11,13,17,19,23,29,31,37};
17     static LL lim[]={4,0,1373653LL,25326001LL,25000000000LL,2152302898747LL,
   ↪3474749660383LL,341550071728321LL,0,0,0,0};
18     if(n<2||n==3215031751LL) return 0;
19     for(int i=0;i<12;++i){
20         if(n<lim[i]) return 1;
21         if(strong_pseudo_primetest(n,testNum[i])==0) return 0;
22     }
23     return 1;
24 }
```

2.5   质因数分解

```
1  int ansn; LL ans[1000];
2  LL func(LL x,LL n){ return(mod_mul(x,x,n)+1)%n; }
3  LL Pollard(LL n){
4      LL i,x,y,p;
5      if(Rabin_Miller(n)) return n;
6      if(!(n&1)) return 2;
7      for(i=1;i<20;i++){
8          x=i; y=func(x,n); p=gcd(y-x,n);
9          while(p==1) {x=func(x,n); y=func(func(y,n),n); p=gcd((y-x+n)%n,n)%n;}
10         if(p==0||p==n) continue;
11         return p;
12     }
13 }
14 void factor(LL n){
```

```
15    LL x;
16    x=Pollard(n);
17    if(x==n){ ans[ansn++]=x; return; }
18    factor(x), factor(n/x);
19 }
```

2.6   线下整点

```
1 // ∑_{i=0}^{n-1} ⌊(a+bi)/m⌋,  n,m,a,b > 0
2 LL solve(LL n,LL a,LL b,LL m){
3    if(b==0) return n*(a/m);
4    if(a>=m) return n*(a/m)+solve(n,a%m,b,m);
5    if(b>=m) return (n-1)*n/2*(b/m)+solve(n,a,b%m,m);
6    return solve((a+b*n)/m,(a+b*n)%m,m,b);
7 }
```

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 后缀数组

```cpp
const int MAXN = MAXL * 2 + 1;
int a[MAXN], x[MAXN], y[MAXN], c[MAXN], sa[MAXN], rank[MAXN], height[MAXN];
void calc_sa(int n) {
    int m = alphabet, k = 1;
    memset(c, 0, sizeof(*c) * (m + 1));
    for (int i = 1; i <= n; ++i) c[x[i] = a[i]]++;
    for (int i = 1; i <= m; ++i) c[i] += c[i - 1];
    for (int i = n; i; --i) sa[c[x[i]]--] = i;
    for (; k <= n; k <<= 1) {
        int tot = k;
        for (int i = n - k + 1; i <= n; ++i) y[i - n + k] = i;
        for (int i = 1; i <= n; ++i)
            if (sa[i] > k) y[++tot] = sa[i] - k;
        memset(c, 0, sizeof(*c) * (m + 1));
        for (int i = 1; i <= n; ++i) c[x[i]]++;
        for (int i = 1; i <= m; ++i) c[i] += c[i - 1];
        for (int i = n; i; --i) sa[c[x[y[i]]]--] = y[i];
        for (int i = 1; i <= n; ++i) y[i] = x[i];
        tot = 1; x[sa[1]] = 1;
        for (int i = 2; i <= n; ++i) {
            if (max(sa[i], sa[i - 1]) + k > n || y[sa[i]] != y[sa[i - 1]] || y[sa[i]
  ↪ + k] != y[sa[i - 1] + k]) ++tot;
            x[sa[i]] = tot;
        }
        if (tot == n) break; else m = tot;
    }
}
void calc_height(int n) {
    for (int i = 1; i <= n; ++i) rank[sa[i]] = i;
    for (int i = 1; i <= n; ++i) {
        height[rank[i]] = max(0, height[rank[i - 1]] - 1);
        if (rank[i] == 1) continue;
        int j = sa[rank[i] - 1];
```

```
33        while (max(i, j) + height[rank[i]] <= n && a[i + height[rank[i]]] == a[j +
   ↪height[rank[i]]]) ++height[rank[i]];
34    }
35 }
```

4.2 后缀自动机

```
1  static const int MAXL = MAXN * 2;  // MAXN is original length
2  static const int alphabet = 26;  // sometimes need changing
3  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
4  char str[MAXL];
5  inline void init() {
6      l = strlen(str + 1); cnt = last = 1;
7      for (int i = 0; i <= l * 2; ++i) memset(trans[i], 0, sizeof(trans[i]));
8      memset(par, 0, sizeof(*par) * (l * 2 + 1));
9      memset(mxl, 0, sizeof(*mxl) * (l * 2 + 1));
10     memset(size, 0, sizeof(*size) * (l * 2 + 1));
11 }
12 inline void extend(int pos, int c) {
13     int p = last, np = last = ++cnt;
14     mxl[np] = mxl[p] + 1; size[np] = 1;
15     for (; p && !trans[p][c]; p = par[p]) trans[p][c] = np;
16     if (!p) par[np] = 1;
17     else {
18         int q = trans[p][c];
19         if (mxl[p] + 1 == mxl[q]) par[np] = q;
20         else {
21             int nq = ++cnt;
22             mxl[nq] = mxl[p] + 1;
23             memcpy(trans[nq], trans[q], sizeof(trans[nq]));
24             par[nq] = par[q];
25             par[np] = par[q] = nq;
26             for (; trans[p][c] == q; p = par[p]) trans[p][c] = nq;
27         }
28     }
29 }
30 inline void buildsam() {
31     for (int i = 1; i <= l; ++i) extend(i, str[i] - 'a');
32     memset(sum, 0, sizeof(*sum) * (l * 2 + 1));
33     for (int i = 1; i <= cnt; ++i) sum[mxl[i]]++;
34     for (int i = 1; i <= l; ++i) sum[i] += sum[i - 1];
35     for (int i = cnt; i; --i) seq[sum[mxl[i]]--] = i;
36     for (int i = cnt; i; --i) size[par[seq[i]]] += size[seq[i]];
37 }
```

4.3 EX 后缀自动机

```
1  inline void add_node(int x, int &last) {
2      int lastnode = last;
3      if (c[lastnode][x]) {
4          int nownode = c[lastnode][x];
5          if (l[nownode] == l[lastnode] + 1) last = nownode;
6          else {
7              int auxnode = ++cnt; l[auxnode] = l[lastnode] + 1;
8              for (int i = 0; i < alphabet; ++i) c[auxnode][i] = c[nownode][i];
9              par[auxnode] = par[nownode]; par[nownode] = auxnode;
10             for (; lastnode && c[lastnode][x] == nownode; lastnode = par[lastnode])
   ↪ {
11                 c[lastnode][x] = auxnode;
12             }
13             last = auxnode;
14         }
15     } else {
16         int newnode = ++cnt; l[newnode] = l[lastnode] + 1;
17         for (; lastnode && !c[lastnode][x]; lastnode = par[lastnode]) c[lastnode][x]
   ↪ = newnode;
18         if (!lastnode) par[newnode] = 1;
19         else {
20             int nownode = c[lastnode][x];
21             if (l[lastnode] + 1 == l[nownode]) par[newnode] = nownode;
22             else {
23                 int auxnode = ++cnt; l[auxnode] = l[lastnode] + 1;
24                 for (int i = 0; i < alphabet; ++i) c[auxnode][i] = c[nownode][i];
25                 par[auxnode] = par[nownode]; par[nownode] = par[newnode] = auxnode;
26                 for (; lastnode && c[lastnode][x] == nownode; lastnode =
   ↪ par[lastnode]) {
27                     c[lastnode][x] = auxnode;
28                 }
29             }
30         }
31         last = newnode;
32     }
33 }
```

4.4 回文自动机

```
1  int nT, nStr, last, c[MAXT][26], fail[MAXT], r[MAXN], l[MAXN], s[MAXN];
2  int allocate(int len) {
3      l[nT] = len;
4      r[nT] = 0;
5      fail[nT] = 0;
6      memset(c[nT], 0, sizeof(c[nT]));
7      return nT++;
```

```
 8 }
 9 void init() {
10     nT = nStr = 0;
11     int newE = allocate(0);
12     int newO = allocate(-1);
13     last = newE;
14     fail[newE] = newO;
15     fail[newO] = newE;
16     s[0] = -1;
17 }
18 void add(int x) {
19     s[++nStr] = x;
20     int now = last;
21     while (s[nStr - l[now] - 1] != s[nStr]) now = fail[now];
22     if (!c[now][x]) {
23         int newnode = allocate(l[now] + 2), &newfail = fail[newnode];
24         newfail = fail[now];
25         while (s[nStr - l[newfail] - 1] != s[nStr]) newfail = fail[newfail];
26         newfail = c[newfail][x];
27         c[now][x] = newnode;
28     }
29     last = c[now][x];
30     r[last]++;
31 }
32 void count() {
33     for (int i = nT - 1; i >= 0; i--) {
34         r[fail[i]] += r[i];
35     }
36 }
```

Chapter 5

数据结构

5.1 KD-Tree

```cpp
long long norm(const long long &x) {
    //    For manhattan distance
    return std::abs(x);
    //    For euclid distance
    return x * x;
}

struct Point {
    int x, y, id;

    const int& operator [] (int index) const {
        if (index == 0) {
            return x;
        } else {
            return y;
        }
    }

    friend long long dist(const Point &a, const Point &b) {
        long long result = 0;
        for (int i = 0; i < 2; ++i) {
            result += norm(a[i] - b[i]);
        }
        return result;
    }
} point[N];

struct Rectangle {
    int min[2], max[2];

    Rectangle() {
        min[0] = min[1] = INT_MAX;  // sometimes int is not enough
        max[0] = max[1] = INT_MIN;
```

```
34         }
35
36     void add(const Point &p) {
37         for (int i = 0; i < 2; ++i) {
38             min[i] = std::min(min[i], p[i]);
39             max[i] = std::max(max[i], p[i]);
40         }
41     }
42
43     long long dist(const Point &p) {
44         long long result = 0;
45         for (int i = 0; i < 2; ++i) {
46             //    For minimum distance
47             result += norm(std::min(std::max(p[i], min[i]), max[i]) - p[i]);
48             //    For maximum distance
49             result += std::max(norm(max[i] - p[i]), norm(min[i] - p[i]));
50         }
51         return result;
52     }
53 };
54
55 struct Node {
56     Point seperator;
57     Rectangle rectangle;
58     int child[2];
59
60     void reset(const Point &p) {
61         seperator = p;
62         rectangle = Rectangle();
63         rectangle.add(p);
64         child[0] = child[1] = 0;
65     }
66 } tree[N << 1];
67
68 int size, pivot;
69
70 bool compare(const Point &a, const Point &b) {
71     if (a[pivot] != b[pivot]) {
72         return a[pivot] < b[pivot];
73     }
74     return a.id < b.id;
75 }
76
77 // 左閉右開: build(1, n + 1)
78 int build(int l, int r, int type = 1) {
79     pivot = type;
80     if (l >= r) {
81         return 0;
82     }
```

```
83      int x = ++size;
84      int mid = l + r >> 1;
85      std::nth_element(point + l, point + mid, point + r, compare);
86      tree[x].reset(point[mid]);
87      for (int i = l; i < r; ++i) {
88          tree[x].rectangle.add(point[i]);
89      }
90      tree[x].child[0] = build(l, mid, type ^ 1);
91      tree[x].child[1] = build(mid + 1, r, type ^ 1);
92      return x;
93  }
94
95  int insert(int x, const Point &p, int type = 1) {
96      pivot = type;
97      if (x == 0) {
98          tree[++size].reset(p);
99          return size;
100     }
101     tree[x].rectangle.add(p);
102     if (compare(p, tree[x].seperator)) {
103         tree[x].child[0] = insert(tree[x].child[0], p, type ^ 1);
104     } else {
105         tree[x].child[1] = insert(tree[x].child[1], p, type ^ 1);
106     }
107     return x;
108 }
109
110 // For minimum distance
111 // For maximum: 下面递归 query 时 0, 1 换顺序;< and >;min and max
112 void query(int x, const Point &p, std::pair<long long, int> &answer, int type = 1) {
113     pivot = type;
114     if (x == 0 || tree[x].rectangle.dist(p) > answer.first) {
115         return;
116     }
117     answer = std::min(answer,
118             std::make_pair(dist(tree[x].seperator, p), tree[x].seperator.id));
119     if (compare(p, tree[x].seperator)) {
120         query(tree[x].child[0], p, answer, type ^ 1);
121         query(tree[x].child[1], p, answer, type ^ 1);
122     } else {
123         query(tree[x].child[1], p, answer, type ^ 1);
124         query(tree[x].child[0], p, answer, type ^ 1);
125     }
126 }
127
128 std::priority_queue<std::pair<long long, int> > answer;
129
130 void query(int x, const Point &p, int k, int type = 1) {
131     pivot = type;
```

```
132    if (x == 0 || (int)answer.size() == k && tree[x].rectangle.dist(p) >
   ↪answer.top().first) {
133        return;
134    }
135    answer.push(std::make_pair(dist(tree[x].seperator, p), tree[x].seperator.id));
136    if ((int)answer.size() > k) {
137        answer.pop();
138    }
139    if (compare(p, tree[x].seperator)) {
140        query(tree[x].child[0], p, k, type ^ 1);
141        query(tree[x].child[1], p, k, type ^ 1);
142    } else {
143        query(tree[x].child[1], p, k, type ^ 1);
144        query(tree[x].child[0], p, k, type ^ 1);
145    }
146 }
```

5.2   Treap

```
1  struct Node{
2      int mn, key, size, tag;
3      bool rev;
4      Node* ch[2];
5      Node(int mn, int key, int size): mn(mn), key(key), size(size), rev(0), tag(0){}
6      void downtag();
7      Node* update(){
8          mn = min(ch[0] -> mn, min(key, ch[1] -> mn));
9          size = ch[0] -> size + 1 + ch[1] -> size;
10         return this;
11     }
12 };
13 typedef pair<Node*, Node*> Pair;
14 Node *null, *root;
15 void Node::downtag(){
16     if(rev){
17         for(int i = 0; i < 2; i++)
18             if(ch[i] != null){
19                 ch[i] -> rev ^= 1;
20                 swap(ch[i] -> ch[0], ch[i] -> ch[1]);
21             }
22         rev = 0;
23     }
24     if(tag){
25         for(int i = 0; i < 2; i++)
26             if(ch[i] != null){
27                 ch[i] -> key += tag;
28                 ch[i] -> mn += tag;
29                 ch[i] -> tag += tag;
```

```
30              }
31          tag = 0;
32      }
33 }
34 int r(){
35      static int s = 3023192386;
36      return (s += (s << 3) + 1) & (~0u >> 1);
37 }
38 bool random(int x, int y){
39      return r() % (x + y) < x;
40 }
41 Node* merge(Node *p, Node *q){
42      if(p == null) return q;
43      if(q == null) return p;
44      p -> downtag();
45      q -> downtag();
46      if(random(p -> size, q -> size)){
47          p -> ch[1] = merge(p -> ch[1], q);
48          return p -> update();
49      }else{
50          q -> ch[0] = merge(p, q -> ch[0]);
51          return q -> update();
52      }
53 }
54 Pair split(Node *x, int n){
55      if(x == null) return make_pair(null, null);
56      x -> downtag();
57      if(n <= x -> ch[0] -> size){
58          Pair ret = split(x -> ch[0], n);
59          x -> ch[0] = ret.second;
60          return make_pair(ret.first, x -> update());
61      }
62      Pair ret = split(x -> ch[1], n - x -> ch[0] -> size - 1);
63      x -> ch[1] = ret.first;
64      return make_pair(x -> update(), ret.second);
65 }
66 pair<Node*, Pair> get_segment(int l, int r){
67      Pair ret = split(root, l - 1);
68      return make_pair(ret.first, split(ret.second, r - l + 1));
69 }
70 int main(){
71      null = new Node(INF, INF, 0);
72      null -> ch[0] = null -> ch[1] = null;
73      root = null;
74 }
```

5.3 Link/cut Tree

```
inline void reverse(int x) {
    tr[x].rev ^= 1; swap(tr[x].c[0], tr[x].c[1]);
}

inline void rotate(int x, int k) {
    int y = tr[x].fa, z = tr[y].fa;
    tr[x].fa = z; tr[z].c[tr[z].c[1] == y] = x;
    tr[tr[x].c[k ^ 1]].fa = y; tr[y].c[k] = tr[x].c[k ^ 1];
    tr[x].c[k ^ 1] = y; tr[y].fa = x;
}

inline void splay(int x, int w) {
    int z = x; pushdown(x);
    while (tr[x].fa != w) {
        int y = tr[x].fa; z = tr[y].fa;
        if (z == w) {
            pushdown(z = y); pushdown(x);
            rotate(x, tr[y].c[1] == x);
            update(y); update(x);
        } else {
            pushdown(z); pushdown(y); pushdown(x);
            int t1 = tr[y].c[1] == x, t2 = tr[z].c[1] == y;
            if (t1 == t2) rotate(y, t2), rotate(x, t1);
            else rotate(x, t1), rotate(x, t2);
            update(z); update(y); update(x);
        }
    }
    update(x);
    if (x != z) par[x] = par[z], par[z] = 0;
}

inline void access(int x) {
    for (int y = 0; x; y = x, x = par[x]) {
        splay(x, 0);
        if (tr[x].c[1]) par[tr[x].c[1]] = x, tr[tr[x].c[1]].fa = 0;
        tr[x].c[1] = y; par[y] = 0; tr[y].fa = x; update(x);
    }
}

inline void makeroot(int x) {
    access(x); splay(x, 0); reverse(x);
}

inline void link(int x, int y) {
    makeroot(x); par[x] = y;
}

```

```
48  inline void cut(int x, int y) {
49      access(x); splay(y, 0);
50      if (par[y] != x) swap(x, y), access(x), splay(y, 0);
51      par[y] = 0;
52  }
53
54  inline void split(int x, int y) {  // x will be the root of the tree
55      makeroot(y); access(x); splay(x, 0);
56  }
```

Chapter 6

图论

6.1 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);
    }
};
```

6.2 KM

```cpp
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];
```

```
 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;
```

6.3   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 ...
```

6.4   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;
```

6.5 最小树形图

```
const int MAXN,INF;// INF >= sum( W_ij )
int from[MAXN + 10][MAXN * 2 + 10],n,m,edge[MAXN + 10][MAXN * 2 + 10];
int sel[MAXN * 2 + 10],fa[MAXN * 2 + 10],vis[MAXN * 2 + 10];
int getfa(int x){if(x == fa[x]) return x; return fa[x] = getfa(fa[x]);}
void liuzhu(){ // 1-base: root is 1, answer = (sel[i], i) for i in [2..n]
    fa[1] = 1;
    for(int i = 2; i <= n; ++i){
        sel[i] = 1; fa[i] = i;
        for(int j = 1; j <= n; ++j) if(fa[j] != i)
            if(from[j][i] = i, edge[sel[i]][i] > edge[j][i]) sel[i] = j;
    }
    int limit = n;
    while(1){
        int prelimit = limit; memset(vis, 0, sizeof(vis)); vis[1] = 1;
        for(int i = 2; i <= prelimit; ++i) if(fa[i] == i && !vis[i]){
            int j = i; while(!vis[j]) vis[j] = i, j = getfa(sel[j]);
            if(j == 1 || vis[j] != i) continue; vector<int> C; int k = j;
            do C.push_back(k), k = getfa(sel[k]); while(k != j);
            ++limit;
            for(int i = 1; i <= n; ++i){
                edge[i][limit] = INF, from[i][limit] = limit;
            }
            fa[limit] = vis[limit] = limit;
            for(int i = 0; i < int(C.size()); ++i){
                int x = C[i], fa[x] = limit;
                for(int j = 1; j <= n; ++j)
                    if(edge[j][x] != INF && edge[j][limit] > edge[j][x] -
 edge[sel[x]][x]){
                        edge[j][limit] = edge[j][x] - edge[sel[x]][x];
                        from[j][limit] = x;
                    }
            }
            for(int j=1;j<=n;++j) if(getfa(j)==limit) edge[j][limit] = INF;
            sel[limit] = 1;
            for(int j = 1; j <= n; ++j)
                if(edge[sel[limit]][limit] > edge[j][limit]) sel[limit] = j;
        }
        if(prelimit == limit) break;
    }
    for(int i = limit; i > 1; --i) sel[from[sel[i]][i]] = sel[i];
}
```

6.6 带花树

```
vector<int> link[maxn];
int n,match[maxn],Queue[maxn],head,tail;
```

```
3  int pred[maxn],base[maxn],start,finish,newbase;
4  bool InQueue[maxn],InBlossom[maxn];
5  void push(int u){ Queue[tail++]=u;InQueue[u]=true; }
6  int pop(){ return Queue[head++]; }
7  int FindCommonAncestor(int u,int v){
8      bool InPath[maxn];
9      for(int i=0;i<n;i++) InPath[i]=0;
10     while(true){ u=base[u];InPath[u]=true;if(u==start) break;u=pred[match[u]]; }
11     while(true){ v=base[v];if(InPath[v]) break;v=pred[match[v]]; }
12     return v;
13 }
14 void ResetTrace(int u){
15     int v;
16     while(base[u]!=newbase){
17         v=match[u];
18         InBlossom[base[u]]=InBlossom[base[v]]=true;
19         u=pred[v];
20         if(base[u]!=newbase) pred[u]=v;
21     }
22 }
23 void BlossomContract(int u,int v){
24     newbase=FindCommonAncestor(u,v);
25     for (int i=0;i<n;i++)
26     InBlossom[i]=0;
27     ResetTrace(u);ResetTrace(v);
28     if(base[u]!=newbase) pred[u]=v;
29     if(base[v]!=newbase) pred[v]=u;
30     for(int i=0;i<n;++i)
31     if(InBlossom[base[i]]){
32         base[i]=newbase;
33         if(!InQueue[i]) push(i);
34     }
35 }
36 bool FindAugmentingPath(int u){
37     bool found=false;
38     for(int i=0;i<n;++i) pred[i]=-1,base[i]=i;
39     for (int i=0;i<n;i++) InQueue[i]=0;
40     start=u;finish=-1; head=tail=0; push(start);
41     while(head<tail){
42         int u=pop();
43         for(int i=link[u].size()-1;i>=0;i--){
44             int v=link[u][i];
45             if(base[u]!=base[v]&&match[u]!=v)
46                 if(v==start||(match[v]>=0&&pred[match[v]]>=0))
47                     BlossomContract(u,v);
48                 else if(pred[v]==-1){
49                     pred[v]=u;
50                     if(match[v]>=0) push(match[v]);
51                     else{ finish=v; return true; }
```

```
52                }
53            }
54        }
55        return found;
56 }
57 void AugmentPath(){
58     int u=finish,v,w;
59     while(u>=0){ v=pred[u];w=match[v];match[v]=u;match[u]=v;u=w; }
60 }
61 void FindMaxMatching(){
62     for(int i=0;i<n;++i) match[i]=-1;
63     for(int i=0;i<n;++i) if(match[i]==-1) if(FindAugmentingPath(i)) AugmentPath();
64 }
```

6.7 Dominator Tree

```
1 vector<int> prec[N], succ[N];
2 vector<int> ord;
3 int stamp, vis[N];
4 int num[N];
5 int fa[N];
6 void dfs(int u) {
7     vis[u] = stamp;
8     num[u] = ord.size();
9     ord.push_back(u);
10    for (int i = 0; i < (int)succ[u].size(); ++i) {
11        int v = succ[u][i];
12        if (vis[v] != stamp) {
13            fa[v] = u;
14            dfs(v);
15        }
16    }
17 }
18 int fs[N], mins[N], dom[N], sem[N];
19 int find(int u) {
20     if (u != fs[u]) {
21         int v = fs[u];
22         fs[u] = find(fs[u]);
23         if (mins[v] != -1 && num[sem[mins[v]]] < num[sem[mins[u]]]) {
24             mins[u] = mins[v];
25         }
26     }
27     return fs[u];
28 }
29 void merge(int u, int v) { fs[u] = v; }
30 vector<int> buf[N];
31 int buf2[N];
32 void mark(int source) {
```

```
33        ord.clear();
34        ++stamp;
35        dfs(source);
36        for (int i = 0; i < (int)ord.size(); ++i) {
37            int u = ord[i];
38            fs[u] = u, mins[u] = -1, buf2[u] = -1;
39        }
40        for (int i = (int)ord.size() - 1; i > 0; --i) {
41            int u = ord[i], p = fa[u];
42            sem[u] = p;
43            for (int j = 0; j < (int)prec[u].size(); ++j) {
44                int v = prec[u][j];
45                if (use[v] != stamp) continue;
46                if (num[v] > num[u]) {
47                    find(v); v = sem[mins[v]];
48                }
49                if (num[v] < num[sem[u]]) {
50                    sem[u] = v;
51                }
52            }
53            buf[sem[u]].push_back(u);
54            mins[u] = u;
55            merge(u, p);
56            while (buf[p].size()) {
57                int v = buf[p].back();
58                buf[p].pop_back();
59                find(v);
60                if (sem[v] == sem[mins[v]]) {
61                    dom[v] = sem[v];
62                } else {
63                    buf2[v] = mins[v];
64                }
65            }
66        }
67        dom[ord[0]] = ord[0];
68        for (int i = 0; i < (int)ord.size(); ++i) {
69            int u = ord[i];
70            if (~buf2[u]) {
71                dom[u] = dom[buf2[u]];
72            }
73        }
74  }
```

6.8   无向图最小割

```
1  int cost[maxn][maxn],seq[maxn],len[maxn],n,m,pop,ans;
2  bool used[maxn];
3  void Init(){
```

```c
    int i,j,a,b,c;
    for(i=0;i<n;i++) for(j=0;j<n;j++) cost[i][j]=0;
    for(i=0;i<m;i++){
        scanf("%d %d %d",&a,&b,&c); cost[a][b]+=c; cost[b][a]+=c;
    }
    pop=n; for(i=0;i<n;i++) seq[i]=i;
}
void Work(){
    ans=inf; int i,j,k,l,mm,sum,pk;
    while(pop > 1){
        for(i=1;i<pop;i++) used[seq[i]]=0; used[seq[0]]=1;
        for(i=1;i<pop;i++) len[seq[i]]=cost[seq[0]][seq[i]];
        pk=0; mm=-inf; k=-1;
        for(i=1;i<pop;i++) if(len[seq[i]] > mm){ mm=len[seq[i]]; k=i; }
        for(i=1;i<pop;i++){
            used[seq[l=k]]=1;
            if(i==pop-2) pk=k;
            if(i==pop-1) break;
            mm=-inf;
            for(j=1;j<pop;j++) if(!used[seq[j]])
                if((len[seq[j]]+=cost[seq[l]][seq[j]]) > mm)
                    mm=len[seq[j]], k=j;
        }
        sum=0;
        for(i=0;i<pop;i++) if(i != k) sum+=cost[seq[k]][seq[i]];
        ans=min(ans,sum);
        for(i=0;i<pop;i++)
            cost[seq[k]][seq[i]]=cost[seq[i]][seq[k]]+=cost[seq[pk]][seq[i]];
        seq[pk]=seq[--pop];
    }
    printf("%d\n",ans);
}
```

Chapter 7

其他

7.1 Dancing Links

```cpp
struct Node {
    Node *l, *r, *u, *d, *col;
    int size, line_no;
    Node() {
        size = 0; line_no = -1;
        l = r = u = d = col = NULL;
    }
} *root;

void cover(Node *c) {
    c->l->r = c->r; c->r->l = c->l;
    for (Node *u = c->d; u != c; u = u->d)
        for (Node *v = u->r; v != u; v = v->r) {
            v->d->u = v->u;
            v->u->d = v->d;
            -- v->col->size;
        }
}

void uncover(Node *c) {
    for (Node *u = c->u; u != c; u = u->u) {
        for (Node *v = u->l; v != u; v = v->l) {
            ++ v->col->size;
            v->u->d = v;
            v->d->u = v;
        }
    }
    c->l->r = c; c->r->l = c;
}

std::vector<int> answer;
bool search(int k) {
    if (root->r == root) return true;
```

```
34      Node *r = NULL;
35      for (Node *u = root->r; u != root; u = u->r)
36          if (r == NULL || u->size < r->size)
37              r = u;
38      if (r == NULL || r->size == 0) return false;
39      else {
40          cover(r);
41          bool succ = false;
42          for (Node *u = r->d; u != r && !succ; u = u->d) {
43              answer.push_back(u->line_no);
44              for (Node *v = u->r; v != u; v = v->r)  // Cover row
45                  cover(v->col);
46              succ |= search(k + 1);
47              for (Node *v = u->l; v != u; v = v->l)
48                  uncover(v->col);
49              if (!succ) answer.pop_back();
50          }
51          uncover(r);
52          return succ;
53      }
54  }
55
56  bool entry[CR][CC];
57  Node *who[CR][CC];
58  int cr, cc;
59
60  void construct() {
61      root = new Node();
62      Node *last = root;
63      for (int i = 0; i < cc; ++ i) {
64          Node *u = new Node();
65          last->r = u; u->l = last;
66          Node *v = u; u->line_no = i;
67          last = u;
68          for (int j = 0; j < cr; ++ j)
69              if (entry[j][i]) {
70                  ++ u->size;
71                  Node *cur = new Node();
72                  who[j][i] = cur;
73                  cur->line_no = j;
74                  cur->col = u;
75                  cur->u = v; v->d = cur;
76                  v = cur;
77              }
78          v->d = u; u->u = v;
79      }
80      last->r = root; root->l = last;
81      for (int j = 0; j < cr; ++ j) {
82          Node *last = NULL;
```

```
83          for (int i = cc - 1; i >= 0; -- i)
84              if (entry[j][i]) {
85                  last = who[j][i];
86                  break;
87              }
88          for (int i = 0; i < cc; ++ i)
89              if (entry[j][i]) {
90                  last->r = who[j][i];
91                  who[j][i]->l = last;
92                  last = who[j][i];
93              }
94      }
95  }
96
97  void destruct() {
98      for (Node *u = root->r; u != root; ) {
99          for (Node *v = u->d; v != u; ) {
100             Node *nxt = v->d;
101             delete(v);
102             v = nxt;
103         }
104         Node *nxt = u->r;
105         delete(u); u = nxt;
106     }
107     delete root;
108 }
```

Chapter 8

技巧

8.1　真正的释放 STL 容器内存空间

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

8.2　无敌的大整数相乘取模

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;
}
```

8.3　无敌的读入优化

```
// getchar() 读入优化 << 关同步 cin << 此优化
// 用 isdigit() 会小幅变慢
// 返回 false 表示读到文件尾
namespace Reader {
    const int L = (1 << 15) + 5;
    char buffer[L], *S, *T;
    __inline bool getchar(char &ch) {
        if (S == T) {
            T = (S = buffer) + fread(buffer, 1, L, stdin);
            if (S == T) {
                ch = EOF;
                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  }
```

Chapter 9

提示

9.1 控制 cout 输出实数精度

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

9.2 vimrc

```
set nu
set sw=4
set sts=4
set ts=4
syntax on
set cindent
```

9.3 让 make 支持 c ++ 11

In .bashrc or whatever:

```
export CXXFLAGS='-std=c++11 -Wall'
```

9.4 线性规划转对偶

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

9.5   32-bit/64-bit 随机素数

32-bit	64-bit
73550053	12492928468556885773
148898719	17017504344198005569
189560747	36054998784424114901
459874703	56483166733878037811
1202316001	61253425708143357977
1431183547	62151553087758851301
1438011109	62946067780406234511
1538762023	63473305504460205477
1557944263	74296329243037252077
1981315913	85247200794803898499

9.6   NTT 素数及其原根

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