

Wizards

Standard Code Library

2017 年 11 月 19 日

目录

1 数论	3	5.3 Treap	23
1.1 $O(m^2 \log n)$ 线性递推	3	5.4 Splay	23
1.2 求逆元	3	5.5 Link cut Tree	25
1.3 中国剩余定理	3	5.6 树上莫队	25
1.4 素性测试	3	5.7 CDQ 分治	26
1.5 质因数分解	3	5.8 整体二分	26
1.6 佩尔方程	4	6 图论	27
1.7 二次剩余	4	6.1 2-SAT	27
1.8 一元三次方程	4	6.2 2-SAT (tarjan)	27
1.9 线下整点	4	6.3 KM	27
1.10 线性同余不等式	4	6.4 点双连通分量	28
1.11 组合数取模	5	6.5 边双连通分量	28
1.12 Schreier-Sims	5	6.6 最小树形图	29
2 代数	6	6.7 带花树	29
2.1 快速傅里叶变换	6	6.8 支配树	30
2.2 分治卷积	6	6.9 无向图最小割	30
2.3 快速数论变换	6	6.10 最大团搜索	30
2.4 快速沃尔什变换	6	6.11 弦图判定	31
2.5 自适应辛普森积分	7	6.12 斯坦纳树	31
2.6 单纯形	7	6.13 虚树	32
3 计算几何	7	6.14 点分治	32
3.1 二维	7	6.15 最小割最大流	33
3.1.1 点类	7	6.16 最小费用流	33
3.1.2 凸包	9	6.17 zkw 费用流	34
3.1.3 凸包最近点对	9	6.18 最小割树	34
3.1.4 三角形的心	12	6.19 上下界网络流建图	35
3.1.5 半平面交	12	6.19.1 无源汇的上下界可行流	35
3.1.6 最大空凸包	12	6.19.2 有源汇的上下界可行流	35
3.1.7 平面最近点对	13	6.19.3 有源汇的上下界最大流	35
3.1.8 最小覆盖圆	14	6.19.4 有源汇的上下界最小流	35
3.1.9 多边形内部可视	14	7 其他	35
3.1.10 V 图	16	7.1 Dancing Links	35
3.2 三维	18	7.1.1 精确覆盖	35
3.2.1 三维点类	18	7.1.2 重复覆盖	36
3.2.2 凸包	18	7.1.3 斜率优化	37
3.2.3 最小覆盖球	18	7.1.4 决策单调性	38
4 字符串	19	7.2 蔡勒公式	38
4.1 AC 自动机	19	7.3 五边形数定理	38
4.2 后缀数组	20	7.4 凸包闵可夫斯基和	39
4.3 后缀自动机	20	7.5 树上路径求交	39
4.4 广义后缀自动机	20	8 技巧	39
4.5 manacher	21	8.1 STL 归还空间	39
4.6 回文自动机	21	8.2 大整数取模	39
4.7 循环串的最小表示	21	8.3 读入优化	39
5 数据结构	22	8.4 二次随机法	39
5.1 可并堆	22	8.5 vimrc	39
5.2 KD-Tree	22	8.6 控制 cout 输出实数精度	40
		8.7 让 make 支持 c++11	40
		8.8 tuple 相关	40

8.9	汇编技巧	40
9	提示	40
9.1	线性规划转对偶	40
9.2	NTT 素数及其原根	40
9.3	积分表	40
9.3.1	$ax^2 + bx + c(a > 0)$	40
9.3.2	$\sqrt{\pm ax^2 + bx + c}(a > 0)$	40
9.3.3	$\sqrt{\pm \frac{x-a}{x-b}}$ 或 $\sqrt{(x-a)(x-b)}$	40
9.3.4	三角函数的积分	40
9.3.5	反三角函数的积分 (其中 $a > 0$)	41
9.3.6	指数函数的积分	41
9.3.7	对数函数的积分	41

1. 数论

1.1 $O(m^2 \log n)$ 线性递推

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

```

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 }

```

1.2 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 }

1.3 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 }

```

1.4 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,21523028987
  ↳ 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 }

```

1.5 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 }

```

1.6 佩尔方程

```

1 import java.math.BigInteger;
2 import java.util.Scanner;
3 //a[n]=(g[n]+a[0])/h[n]
4 //g[n]=a[n-1]*h[n-1]-g[n-1]
5 //h[n]=(N-g[n]*g[n])/h[n-1]
6 //p[n]=a[n-1]*p[n-1]+p[n-2]
7 //q[n]=a[n-1]*q[n-1]+q[n-2]
8 //so:
9 //p[n]*q[n-1]-p[n-1]*q[n]=(-1)^(n+1);
10 //p[n]^2-N*q[n]^2=(-1)^(n+1)*h[n+1];
11 public class Main {
12     public static BigInteger p, q;
13     public static void solve(int n) {
14         BigInteger N, p1, p2, q1, q2, a0, a1, a2, g1, g2,
15             h1, h2;
16         g1 = q2 = p1 = BigInteger.ZERO;
17         h1 = q1 = p2 = BigInteger.ONE;
18         a0 = a1 =
19             BigInteger.valueOf((long)Math.sqrt(1.0*n));
20         N = BigInteger.valueOf(n);
21         while (true) {
22             g2 = a1.multiply(h1).subtract(g1);
23             h2 = N.subtract(g2.pow(2)).divide(h1);
24             a2 = g2.add(a0).divide(h2);
25             p = a1.multiply(p2).add(p1);
26             q = a1.multiply(q2).add(q1);
27             if
28                 (p.pow(2).subtract(N.multiply(q.pow(2))).compareTo(BigInteger.ONE)
29                 == 0) return;
30             g1 = g2; h1 = h2; a1 = a2;
31             p1 = p2; p2 = p;
32             q1 = q2; q2 = q;
33         }
34     }
35     public static void main(String[] args) {
36         Scanner cin = new Scanner(System.in);
37         int t=cin.nextInt();
38         while (t--!=0) {
39             solve(cin.nextInt());
40             System.out.println(p + " " + q);
41         }
42     }
43 }

```

1.7 二次剩余

```

1 // x^2 = a (mod p), 0 <= a < p, 返回 true or false 代表
2 // 是否存在解
3 // p 必须是质数, 若是多个单质数的乘积, 可以分别
4 // 求解再用 CRT 合并
5 // 复杂度为 O(log n)
6 void multiply(ll &c, ll &d, ll a, ll b, ll w) {
7     int cc = (a * c + b * d % MOD * w) % MOD;
8     int dd = (a * d + b * c) % MOD;
9     c = cc, d = dd;
10 }
11 bool solve(int n, int &x) {
12     if (MOD == 2) return x = 1, true;
13     if (power(n, MOD / 2, MOD) == MOD - 1) return false;
14     ll c = 1, d = 0, b = 1, a, w;
15     // finding a such that a^2 - n is not a square
16     do { a = rand() % MOD;
17         w = (a * a - n + MOD) % MOD;

```

```

17         if (w == 0) return x = a, true;
18     } while (power(w, MOD / 2, MOD) != MOD - 1);
19     for (int times = (MOD + 1) / 2; times; times >>= 1) {
20         if (times & 1) multiply(c, d, a, b, w);
21         multiply(a, b, a, b, w);
22     }
23     // x = (a + sqrt(w)) ^ ((p + 1) / 2)
24     return x = c, true;
25 }

```

1.8 一元三次方程

```

1 double a(p[3]), b(p[2]), c(p[1]), d(p[0]);
2 double k(b / a), m(c / a), n(d / a);
3 double p(-k * k / 3. + m);
4 double q(2. * k * k * k / 27 - k * m / 3. + n);
5 Complex omega[3] = {Complex(1, 0), Complex(-0.5, 0.5 *
6     sqrt(3)), Complex(-0.5, -0.5 * sqrt(3))};
7 Complex r1, r2;
8 double delta(q * q / 4 + p * p * p / 27);
9 if (delta > 0) {
10     r1 = cubrt(-q / 2. + sqrt(delta));
11     r2 = cubrt(-q / 2. - sqrt(delta));
12 } else {
13     r1 = pow(-q / 2. + pow(Complex(delta), 0.5), 1. / 3);
14     r2 = pow(-q / 2. - pow(Complex(delta), 0.5), 1. / 3);
15 }
16 for(int _ (0); _ < 3; _++) {
17     Complex x = -k / 3. + r1 * omega[_ * 1] + r2 * omega[_
18         * 2 % 3];
19 }

```

1.9 线下整点

```

1 // sum_{i=0}^{n-1} floor((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, b);
7 }

```

1.10 线性同余不等式

```

1 // Find the minimal non-negative solutions for
2 // l ≤ d · x mod m ≤ r
3 // 0 ≤ d, l, r < m; l ≤ r, O(log n)
4 ll cal(ll m, ll d, ll l, ll r) {
5     if (l == 0) return 0;
6     if (d == 0) return MXL; // 无解
7     if (d * 2 > m) return cal(m, m - d, m - r, m - l);
8     if ((l - 1) / d < r / d) return (l - 1) / d + 1;
9     ll k = cal(d, (-m % d + d) % d, l % d, r % d);
10    return k == MXL ? MXL : (k * m + l - 1) / d + 1; // 无
11    解 2
12 }
13 // return all x satisfying l1<=x<=r1 and
14 // l2<=(x*mul+add)%LIM<=r2
15 // here LIM = 2^32 so we use UI instead of "%".
16 // O(log p + #solutions)
17 struct Jump {
18     UI val, step;
19     Jump(UI val, UI step) : val(val), step(step) {}
20     Jump operator + (const Jump & b) const {
21         return Jump(val + b.val, step + b.step);
22     }
23     Jump operator - (const Jump & b) const {
24         return Jump(val - b.val, step + b.step);
25     }
26 };
27 inline Jump operator * (UI x, const Jump & a) {

```

```

24     return Jump(x * a.val, x * a.step);
25 }
26 vector<UI> solve(UI l1, UI r1, UI l2, UI r2, pair<UI, UI>
    ↪ muladd) {
27     UI mul = muladd.first, add = muladd.second, w = r2 -
    ↪ l2;
28     Jump up(mul, 1), dn(-mul, 1);
29     UI s(l1 * mul + add);
30     Jump lo(r2 - s, 0), hi(s - l2, 0);
31     function<void(Jump &, Jump &)> sub = [&](Jump & a,
    ↪ Jump & b) {
32         if (a.val > w) {
33             UI t(((long long)a.val - max(0ll, w + 1ll -
    ↪ b.val)) / b.val);
34             a = a - t * b;
35         }
36     };
37     sub(lo, up), sub(hi, dn);
38     while (up.val > w || dn.val > w) {
39         sub(up, dn); sub(lo, up);
40         sub(dn, up); sub(hi, dn); }
41     assert(up.val + dn.val > w);
42     vector<UI> res;
43     Jump bg(s + mul * min(lo.step, hi.step), min(lo.step,
    ↪ hi.step));
44     while (bg.step <= r1 - l1) {
45         if (l2 <= bg.val && bg.val <= r2)
46             res.push_back(bg.step + l1);
47         if (l2 <= bg.val - dn.val && bg.val - dn.val <=
    ↪ r2) {
48             bg = bg - dn;
49         } else bg = bg + up;
50     } return res;
51 }

```

1.11 组合数取模

```

1 LL prod=1,P;
2 pair<LL,LL> comput(LL n,LL p,LL k){
3     if(n<=1)return make_pair(0,1);
4     LL ans=1,cnt=0;
5     ans=pow(prod,n/P,P);
6     cnt=n/p;
7     pair<LL,LL>res=comput(n/p,p,k);
8     cnt+=res.first;
9     ans=ans*res.second%P;
10    for(int i=n-n/P+1;i<=n;i++)if(i%p){
11
12        ans=ans*i%P;
13    }
14    return make_pair(cnt,ans);
15 }
16 pair<LL,LL> calc(LL n,LL p,LL k){
17     prod=1,P=pow(p,k,1e18);
18     for(int i=1;i<P;i++)if(i%p)prod=prod*i%P;
19     pair<LL,LL> res=comput(n,p,k);
20     // res.second=res.second*pow(p,res.first%k,P)%P;
21     // res.first-=res.first%k;
22     return res;
23 }
24 LL calc(LL n,LL m,LL p,LL k){
25     pair<LL,LL>A,B,C;
26     LL P=pow(p,k,1e18);
27     A=calc(n,p,k);
28     B=calc(m,p,k);
29     C=calc(n-m,p,k);
30     LL ans=1;
31     ans=pow(p,A.first-B.first-C.first,P);
32
    ↪ ans=ans*A.second%P*inv(B.second,P)%P*inv(C.second,P)%P;
33     return ans;
34 }

```

1.12 Schreier-Sims

```

1 struct Perm{
2     vector<int> P; Perm() {} Perm(int n) { P.resize(n); }
3     Perm inv()const{
4         Perm ret(P.size());
5         for(int i = 0; i < int(P.size()); ++i) ret.P[P[i]] =
    ↪ i;
6         return ret;
7     }
8     int &operator [] (const int &dn){ return P[dn]; }
9     void resize(const size_t &sz){ P.resize(sz); }
10    size_t size()const{ return P.size(); }
11    const int &operator [] (const int &dn)const{ return
    ↪ P[dn]; }
12 };
13 Perm operator *(const Perm &a, const Perm &b){
14     Perm ret(a.size());
15     for(int i = 0; i < (int)a.size(); ++i) ret[i] = b[a[i]];
16     return ret;
17 }
18 typedef vector<Perm> Bucket;
19 typedef vector<int> Table;
20 typedef pair<int,int> PII;
21 int n, m;
22 vector<Bucket> buckets, bucketsInv; vector<Table>
    ↪ lookupTable;
23 int fastFilter(const Perm &g, bool addToGroup = true) {
24     int n = buckets.size();
25     Perm p(g);
26     for(int i = 0; i < n; ++i){
27         int res = lookupTable[i][p[i]];
28         if(res == -1){
29             if(addToGroup){
30                 buckets[i].push_back(p);
    ↪ bucketsInv[i].push_back(p.inv());
31                 lookupTable[i][p[i]] = (int)buckets[i].size() - 1;
32             }
33             return i;
34         }
35         p = p * bucketsInv[i][res];
36     }
37     return -1;
38 }
39 long long calcTotalSize(){
40     long long ret = 1;
41     for(int i = 0; i < n; ++i) ret *= buckets[i].size();
42     return ret;
43 }
44 bool inGroup(const Perm &g){ return fastFilter(g, false)
    ↪ == -1; }
45 void solve(const Bucket &gen,int _n){// m perm[0..n - 1]s
46     n = _n, m = gen.size();
47     //clear all
48     vector<Bucket> _buckets(n); swap(buckets, _buckets);
49     vector<Bucket> _bucketsInv(n); swap(bucketsInv,
    ↪ _bucketsInv);
50     vector<Table> _lookupTable(n); swap(lookupTable,
    ↪ _lookupTable);
51 }
52 for(int i = 0; i < n; ++i){
53     lookupTable[i].resize(n);
54     fill(lookupTable[i].begin(), lookupTable[i].end(),
    ↪ -1);
55 }
56 Perm id(n);
57 for(int i = 0; i < n; ++i) id[i] = i;
58 for(int i = 0; i < n; ++i){
59     buckets[i].push_back(id); bucketsInv[i].push_back(id);
60     lookupTable[i][i] = 0;
61 }
62 for(int i = 0; i < m; ++i) fastFilter(gen[i]);

```

```

63 queue<pair<PII,PII> > toUpdate;
64 for(int i = 0; i < n; ++i)
65     for(int j = i; j < n; ++j)
66         for(int k = 0; k < (int)buckets[i].size(); ++k)
67             for(int l = 0; l < (int)buckets[j].size(); ++l)
68                 toUpdate.push(make_pair(PII(i,k), PII(j,l)));
69 while(!toUpdate.empty()){
70     PII a = toUpdate.front().first, b =
71         toUpdate.front().second;
72     toUpdate.pop();
73     int res = fastFilter(buckets[a.first][a.second] *
74         buckets[b.first][b.second]);
75     if(res==-1) continue;
76     PII newPair(res, (int)buckets[res].size() - 1);
77     for(int i = 0; i < n; ++i)
78         for(int j = 0; j < (int)buckets[i].size(); ++j){
79             if(i <= res) toUpdate.push(make_pair(PII(i, j),
80                 newPair));
81             if(res <= i) toUpdate.push(make_pair(newPair,
82                 PII(i, j)));
83         }
84 }

```

2. 代数

2.1 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 }

```

2.2 分治卷积

```

1 int nT, nStr, last, c[MAXT][26], fail[MAXT], r[MAXN],
2     l[MAXN], s[MAXN];
3 int allocate(int len) {
4     l[nT] = len;
5     r[nT] = 0;
6     fail[nT] = 0;
7     memset(c[nT], 0, sizeof(c[nT]));
8     return nT++;
9 }
10 void init() {
11     nT = nStr = 0;
12     int newE = allocate(0);
13     int newO = allocate(-1);
14     last = newE;
15     fail[newE] = newO;
16     fail[newO] = newE;
17     s[0] = -1;
18 }
19 void add(int x) {
20     s[++nStr] = x;
21     int now = last;
22     while (s[nStr - 1[now] - 1] != s[nStr]) now = fail[now];
23     if (!c[now][x]) {
24         int newnode = allocate(l[now] + 2), &newfail =
25             fail[newnode];
26     }

```

```

24     newfail = fail[now];
25     while (s[nStr - 1[newfail] - 1] != s[nStr]) newfail =
26         fail[newfail];
27     newfail = c[newfail][x];
28     c[now][x] = newnode;
29 }
30 last = c[now][x];
31 r[last]++;
32 }
33 void count() {
34     for (int i = nT - 1; i >= 0; i--) {
35         r[fail[i]] += r[i];
36     }

```

2.3 快速数论变换

```

1 int nT, nStr, last, c[MAXT][26], fail[MAXT], r[MAXN],
2     l[MAXN], s[MAXN];
3 int allocate(int len) {
4     l[nT] = len;
5     r[nT] = 0;
6     fail[nT] = 0;
7     memset(c[nT], 0, sizeof(c[nT]));
8     return nT++;
9 }
10 void init() {
11     nT = nStr = 0;
12     int newE = allocate(0);
13     int newO = allocate(-1);
14     last = newE;
15     fail[newE] = newO;
16     fail[newO] = newE;
17     s[0] = -1;
18 }
19 void add(int x) {
20     s[++nStr] = x;
21     int now = last;
22     while (s[nStr - 1[now] - 1] != s[nStr]) now = fail[now];
23     if (!c[now][x]) {
24         int newnode = allocate(l[now] + 2), &newfail =
25             fail[newnode];
26         newfail = fail[now];
27         while (s[nStr - 1[newfail] - 1] != s[nStr]) newfail =
28             fail[newfail];
29         newfail = c[newfail][x];
30         c[now][x] = newnode;
31     }
32     last = c[now][x];
33     r[last]++;
34 }
35 void count() {
36     for (int i = nT - 1; i >= 0; i--) {
37         r[fail[i]] += r[i];
38     }

```

2.4 快速沃尔什变换

```

1 void FWT(LL a[], int n, int ty) { //the length is 2^n
2     for(int d=1; d<n; d<=1){
3         for(int m=(d<1), i=0; i<n; i+=m){
4             if(ty==1){
5                 for(int j=0; j<d; j++){
6                     LL x=a[i+j], y=a[i+j+d];
7                     a[i+j]=x+y;
8                     a[i+j+d]=x-y;
9                     //and: a[i+j]=x+y; or: a[i+j+d]=x+y;
10                }
11            }else{
12                for(int j=0; j<d; j++){

```

```

13     LL x=a[i+j],y=a[i+j+d];
14     a[i+j]=(x+y)/2;
15     a[i+j+d]=(x-y)/2;
16         //and:a[i+j]=x-y; or:a[i+j+d]=y-x;
17     }
18 }
19 }
20 }
21 }

```

2.5 自适应辛普森积分

```

1 namespace adaptive_simpson {
2     template<typename function>
3     inline double area(function f, const double &left, const
4         ↪ double &right) {
5         double mid = (left + right) / 2;
6         return (right - left) * (f(left) + 4 * f(mid) +
7             ↪ f(right)) / 6;
8     }
9
10    template<typename function>
11    inline double simpson(function f, const double &left,
12        ↪ const double &right, const double &eps, const
13        ↪ double &area_sum) {
14        double mid = (left + right) / 2;
15        double area_left = area(f, left, mid);
16        double area_right = area(f, mid, right);
17        double area_total = area_left + area_right;
18        if (fabs(area_total - area_sum) <= 15 * eps) {
19            return area_total + (area_total - area_sum) / 15;
20        }
21        return simpson(f, left, right, eps / 2, area_left) +
22            ↪ simpson(f, mid, right, eps / 2, area_right);
23    }
24
25    template<typename function>
26    inline double simpson(function f, const double &left,
27        ↪ const double &right, const double &eps) {
28        return simpson(f, left, right, eps, area(f, left,
29            ↪ right));
30    }
31 }

```

2.6 单纯形

```

1 const double eps = 1e-8;
2 // max{c * x | Ax <= b, x >= 0} 的解, 无解返回空的
3   ↪ vector, 否则就是解.
4 vector<double> simplex(vector<vector<double>> &A,
5     ↪ vector<double> b, vector<double> c) {
6     int n = A.size(), m = A[0].size() + 1, r = n, s = m - 1;
7     vector<vector<double>> D(n + 2, vector<double>(m + 1));
8     vector<int> ix(n + m);
9     for(int i = 0; i < n + m; i++) {
10         ix[i] = i;
11     }
12     for(int i = 0; i < n; i++) {
13         for(int j = 0; j < m - 1; j++) {
14             D[i][j] = -A[i][j];
15         }
16         D[i][m - 1] = 1;
17         D[i][m] = b[i];
18         if (D[r][m] > D[i][m]) {
19             r = i;
20         }
21     }
22     for(int j = 0; j < m - 1; j++) {
23         D[n][j] = c[j];
24     }
25     D[n + 1][m - 1] = -1;

```

```

25 for(double d; ;) {
26     if (r < n) {
27         swap(ix[s], ix[r + m]);
28         D[r][s] = 1. / D[r][s];
29         for(int j = 0; j <= m; j++) {
30             if (j != s) {
31                 D[r][j] *= -D[r][s];
32             }
33         }
34         for(int i = 0; i <= n + 1; i++) {
35             if (i != r) {
36                 for(int j = 0; j <= m; j++) {
37                     if (j != s) {
38                         D[i][j] += D[r][j] * D[i][s];
39                     }
40                 }
41                 D[i][s] *= D[r][s];
42             }
43         }
44     }
45     r = -1, s = -1;
46     for(int j = 0; j < m; j++) {
47         if (s < 0 || ix[s] > ix[j]) {
48             if (D[n + 1][j] > eps || D[n + 1][j] > -eps &&
49                 ↪ D[n][j] > eps) {
50                 s = j;
51             }
52         }
53     }
54     if (s < 0) {
55         break;
56     }
57     for(int i = 0; i < n; i++) {
58         if (D[i][s] < -eps) {
59             if (r < 0 || (d = D[r][m] / D[r][s] - D[i][m] /
60                 ↪ D[i][s]) < -eps
61                 || d < eps && ix[r + m] > ix[i + m]) {
62                 r = i;
63             }
64         }
65     }
66     if (r < 0) {
67         return vector<double> ();
68     }
69 }
70 if (D[n + 1][m] < -eps) {
71     return vector<double> ();
72 }
73
74 vector<double> x(m - 1);
75 for(int i = m; i < n + m; i++) {
76     if (ix[i] < m - 1) {
77         x[ix[i]] = D[i - m][m];
78     }
79 }
80 return x;
81 }

```

3. 计算几何

3.1 二维

3.1.1 点类

```

1 int sign(DB x) {
2     return (x > eps) - (x < -eps);
3 }
4 DB msqrt(DB x) {
5     return sign(x) > 0 ? sqrt(x) : 0;
6 }

```



```

7
8 struct Point {
9     DB x, y;
10     Point rotate(DB ang) const { // 逆时针旋转 ang 弧度
11         return Point(cos(ang) * x - sin(ang) * y,
12             cos(ang) * y + sin(ang) * x);
13     }
14     Point turn90() const { // 逆时针旋转 90 度
15         return Point(-y, x);
16     }
17     Point unit() const {
18         return *this / len();
19     }
20 };
21 DB dot(const Point& a, const Point& b) {
22     return a.x * b.x + a.y * b.y;
23 }
24 DB det(const Point& a, const Point& b) {
25     return a.x * b.y - a.y * b.x;
26 }
27 #define cross(p1,p2,p3)
28     ↪ ((p2.x-p1.x)*(p3.y-p1.y)-(p3.x-p1.x)*(p2.y-p1.y))
29 #define crossOp(p1,p2,p3) sign(cross(p1,p2,p3))
30 bool isLL(const Line& l1, const Line& l2, Point& p) { //
31     ↪ 直线与直线交点
32     DB s1 = det(l2.b - l2.a, l1.a - l2.a),
33         s2 = -det(l2.b - l2.a, l1.b - l2.a);
34     if (!sign(s1 + s2)) return false;
35     p = (l1.a * s2 + l1.b * s1) / (s1 + s2);
36     return true;
37 }
38 bool onSeg(const Line& l, const Point& p) { // 点在线段
39     ↪ 上
40     return sign(det(p - l.a, l.b - l.a)) == 0 && sign(dot(p
41         ↪ - l.a, p - l.b)) <= 0;
42 }
43 Point projection(const Line & l, const Point& p) {
44     return l.a + (l.b - l.a) * (dot(p - l.a, l.b - l.a) /
45         ↪ (l.b - l.a).len2());
46 }
47 DB disToLine(const Line& l, const Point& p) { // 点到 *
48     ↪ 直线 * 距离
49     return fabs(det(p - l.a, l.b - l.a) / (l.b -
50         ↪ l.a).len());
51 }
52 DB disToSeg(const Line& l, const Point& p) { // 点到线段
53     ↪ 距离
54     return sign(dot(p - l.a, l.b - l.a)) * sign(dot(p - l.b,
55         ↪ l.a - l.b)) == 1 ? disToLine(l, p) : std::min((p -
56         ↪ l.a).len(), (p - l.b).len());
57 }
58 // 圆与直线交点
59 bool isCL(Circle a, Line l, Point& p1, Point& p2) {
60     DB x = dot(l.a - a.o, l.b - l.a),
61         y = (l.b - l.a).len2(),
62         d = x * x - y * ((l.a - a.o).len2() - a.r * a.r);
63     if (sign(d) < 0) return false;
64     Point p = l.a - ((l.b - l.a) * (x / y)), delta = (l.b -
65         ↪ l.a) * (msqrt(d) / y);
66     p1 = p + delta; p2 = p - delta;
67     return true;
68 }
69 // 圆与圆的交面积
70 DB areaCC(const Circle& c1, const Circle& c2) {
71     DB d = (c1.o - c2.o).len();
72     if (sign(d - (c1.r + c2.r)) >= 0) return 0;
73     if (sign(d - std::abs(c1.r - c2.r)) <= 0) {
74         DB r = std::min(c1.r, c2.r);
75         return r * r * PI;
76     }
77     DB x = (d * d + c1.r * c1.r - c2.r * c2.r) / (2 * d),
78         t1 = acos(x / c1.r), t2 = acos((d - x) / c2.r);
79     return c1.r * c1.r * t1 + c2.r * c2.r * t2 - d * c1.r *
80         ↪ sin(t1);
81 }
82 // 圆与圆交点
83 bool isCC(Circle a, Circle b, P& p1, P& p2) {
84     DB s1 = (a.o - b.o).len();
85     if (sign(s1 - a.r - b.r) > 0 || sign(s1 - std::abs(a.r -
86         ↪ b.r)) < 0) return false;
87     DB s2 = (a.r * a.r - b.r * b.r) / s1;
88     DB aa = (s1 + s2) * 0.5, bb = (s1 - s2) * 0.5;
89     P o = (b.o - a.o) * (aa / (aa + bb)) + a.o;
90     P delta = (b.o - a.o).unit().turn90() * msqrt(a.r * a.r
91         ↪ - aa * aa);
92     p1 = o + delta, p2 = o - delta;
93     return true;
94 }
95 // 求点到圆的切点, 按关于点的顺时针方向返回两个点
96 bool tanCP(const Circle &c, const Point &p0, Point &p1,
97     ↪ Point &p2) {
98     double x = (p0 - c.o).len2(), d = x - c.r * c.r;
99     if (d < eps) return false; // 点在圆上认为没有切点
100     Point p = (p0 - c.o) * (c.r * c.r / x);
101     Point delta = ((p0 - c.o) * (-c.r * sqrt(d) /
102         ↪ x)).turn90();
103     p1 = c.o + p + delta;
104     p2 = c.o + p - delta;
105     return true;
106 }
107 // 求圆到圆的外共切线, 按关于 c1.o 的顺时针方向返
108     ↪ 回两条线
109 vector<Line> extanCC(const Circle &c1, const Circle &c2) {
110     vector<Line> ret;
111     if (sign(c1.r - c2.r) == 0) {
112         Point dir = c2.o - c1.o;
113         dir = (dir * (c1.r / dir.len())).turn90();
114         ret.push_back(Line(c1.o + dir, c2.o + dir));
115         ret.push_back(Line(c1.o - dir, c2.o - dir));
116     } else {
117         Point p = (c1.o * -c2.r + c2.o * c1.r) / (c1.r -
118         ↪ c2.r);
119         Point p1, p2, q1, q2;
120         if (tanCP(c1, p, p1, p2) && tanCP(c2, p, q1, q2)) {
121             if (c1.r < c2.r) swap(p1, p2), swap(q1, q2);
122             ret.push_back(Line(p1, q1));
123             ret.push_back(Line(p2, q2));
124         }
125     }
126     return ret;
127 }
128 // 求圆到圆的内共切线, 按关于 c1.o 的顺时针方向返
129     ↪ 回两条线
130 std::vector<Line> intanCC(const Circle &c1, const Circle
131     ↪ &c2) {
132     std::vector<Line> ret;
133     Point p = (c1.o * c2.r + c2.o * c1.r) / (c1.r + c2.r);
134     Point p1, p2, q1, q2;
135     if (tanCP(c1, p, p1, p2) && tanCP(c2, p, q1, q2)) { //
136         ↪ 两圆相切认为没有切线
137         ret.push_back(Line(p1, q1));
138         ret.push_back(Line(p2, q2));
139     }
140     return ret;
141 }
142 bool contain(vector<Point> polygon, Point p) { // 判断点
143     ↪ p 是否被多边形包含, 包括落在边界上
144     int ret = 0, n = polygon.size();
145     for(int i = 0; i < n; ++i) {
146         Point u = polygon[i], v = polygon[(i + 1) % n];
147         if (onSeg(Line(u, v), p)) return true; // Here I
148         ↪ guess.
149         if (sign(u.y - v.y) <= 0) swap(u, v);
150     }
151 }

```

```

127     if (sign(p.y - u.y) > 0 || sign(p.y - v.y) <= 0)
128         ↪ continue;
129     ret += sign(det(p, v, u)) > 0;
130 }
131 return ret & 1;
132 }
133 // 用半平面 (q1,q2) 的逆时针方向去切凸多边形
134 std::vector<Point> convexCut(const std::vector<Point>&ps,
135     ↪ Point q1, Point q2) {
136     std::vector<Point> qs; int n = ps.size();
137     for (int i = 0; i < n; ++i) {
138         Point p1 = ps[i], p2 = ps[(i + 1) % n];
139         int d1 = crossOp(q1,q2,p1), d2 = crossOp(q1,q2,p2);
140         if (d1 >= 0) qs.push_back(p1);
141         if (d1 * d2 < 0) qs.push_back(isSS(p1, p2, q1, q2));
142     }
143     return qs;
144 }
145 // 求凸包
146 std::vector<Point> convexHull(std::vector<Point> ps) {
147     int n = ps.size(); if (n <= 1) return ps;
148     std::sort(ps.begin(), ps.end());
149     std::vector<Point> qs;
150     for (int i = 0; i < n; qs.push_back(ps[i ++]))
151         while (qs.size() > 1 && sign(det(qs[qs.size() - 2],
152             ↪ qs.back(), ps[i])) <= 0)
153             qs.pop_back();
154     for (int i = n - 2, t = qs.size(); i >= 0;
155         ↪ qs.push_back(ps[i --]))
156         while ((int)qs.size() > t && sign(det(qs[qs.size() -
157             ↪ 2], qs.back(), ps[i])) <= 0)
158             qs.pop_back();
159     return qs;

```

3.1.2 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
7         ↪ sorted.
8         clear(shell); int n = p.size();
9         for (int i = 0, j = 0; i < n; i++, j++) {
10             for (; j >= 2 && sign(det(shell[j-1] - shell[j-2],
11                 ↪ p[i] - shell[j-2])) <= 0; --j)
12                 shell.pop_back();
13             shell.push_back(p[i]);
14         }
15     }
16 }
17 void make_convex() {
18     std::sort(a.begin(), a.end());
19     make_shell(a, lower);
20     std::reverse(a.begin(), a.end());
21     make_shell(a, upper);
22     a = lower; a.pop_back();
23     a.insert(a.end(), upper.begin(), upper.end());
24     if ((int)a.size() >= 2) a.pop_back();
25     n = a.size();
26 }
27 void init(const std::vector<Point>& _a) {
28     clear(a); a = _a; n = a.size();
29     make_convex();
30 }
31 void read(int _n) { // Won't make convex.
32     clear(a); n = _n; a.resize(n);
33     for (int i = 0; i < n; i++)
34         a[i].read();
35 }
36 std::pair<DB, int> get_tangent(
37     const std::vector<Point>& convex, const Point& vec)
38     ↪ {

```

```

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)) >
40             ↪ 0)
41             r = mid;
42         else l = mid;
43     }
44     return std::max(std::make_pair(det(vec, convex[r]),
45         ↪ r),
46         std::make_pair(det(vec, convex[0]), 0));
47 }
48 int binary_search(Point u, Point v, int l, int r) {
49     int s1 = sign(det(v - u, a[l % n] - u));
50     for (; l + 1 < r; ) {
51         int mid = (l + r) / 2;
52         int smid = sign(det(v - u, a[mid % n] - u));
53         if (smid == s1) l = mid;
54         else r = mid;
55     }
56     return l % n;
57 }
58 // 求凸包上和向量 vec 叉积最大的点, 返回编号, 共
59 ↪ 线的多个切点返回任意一个
60 int get_tangent(Point vec) {
61     std::pair<DB, int> ret = get_tangent(upper, vec);
62     ret.second = (ret.second + (int)lower.size() - 1) % n;
63     ret = std::max(ret, get_tangent(lower, vec));
64     return ret.second;
65 }
66 // 求凸包和直线 u, v 的交点, 如果不相交返回 false,
67 ↪ 如果有则是和 (i, next(i)) 的交点, 交在点上不
68 ↪ 确定返回前后两条边其中之一
69 bool get_intersection(Point u, Point v, int &i0, int
70     ↪ &i1) {
71     int p0 = get_tangent(u - v), p1 = get_tangent(v - u);
72     if (sign(det(v - u, a[p0] - u)) * sign(det(v - u,
73         ↪ a[p1] - u)) <= 0) {
74         if (p0 > p1) std::swap(p0, p1);
75         i0 = binary_search(u, v, p0, p1);
76         i1 = binary_search(u, v, p1, p0 + n);
77         return true;
78     }
79     else return false;
80 }
81 }
82 };

```

3.1.3 凸包最近点对

```

1 #include<cstdio>
2 #include<cmath>
3 #include<cstring>
4 #include<iostream>
5 #include<algorithm>
6 #include<stdlib.h>
7 #include<queue>
8 #include<map>
9 #include<stack>
10 #include<set>
11 #define e exp(1.0); //2.718281828
12 #define mod 1000000007
13 #define INF 0x7fffffff
14 #define inf 0x3f3f3f3f
15 typedef long long LL;
16 using namespace std;
17
18 #define zero(x) (((x)>0?(x):(-x))<eps)
19 const double eps=1e-8;
20
21 //判断数 k 的符号 -1 负数 1 正数 0 零

```

```

22 int dcmp(double k) {
23     return k<-eps?-1:k>eps?1:0;
24 }
25
26 inline double sqr(double x) {
27     return x*x;
28 }
29 struct point {
30     double x,y;
31     point() {};
32     point(double a,double b):x(a),y(b) {};
33     void input() {
34         scanf("%lf %lf",&x,&y);
35     }
36     friend point operator + (const point &a,const point
        ↪ &b) {
37         return point(a.x+b.x,a.y+b.y);
38     }
39     friend point operator - (const point &a,const point
        ↪ &b) {
40         return point(a.x-b.x,a.y-b.y);
41     }
42     friend bool operator == (const point &a,const point
        ↪ &b) {
43         return dcmp(a.x-b.x)==0&&dcmp(a.y-b.y)==0;
44     }
45     friend point operator * (const point &a,const double
        ↪ &b) {
46         return point(a.x*b,a.y*b);
47     }
48     friend point operator * (const double &a,const point
        ↪ &b) {
49         return point(a*b.x,a*b.y);
50     }
51     friend point operator / (const point &a,const double
        ↪ &b) {
52         return point(a.x/b,a.y/b);
53     }
54     friend bool operator < (const point &a, const point
        ↪ &b) {
55         return a.x < b.x || (a.x == b.x && a.y < b.y);
56     }
57     double norm() {
58         return sqrt(sqr(x)+sqr(y));
59     }
60 };
61 //计算两个向量的叉积
62 double cross(const point &a,const point &b) {
63     return a.x*b.y-a.y*b.x;
64 }
65 double cross3(point A,point B,point C) { //叉乘
66     return (B.x-A.x)*(C.y-A.y)-(B.y-A.y)*(C.x-A.x);
67 }
68 //计算两个点的点积
69 double dot(const point &a,const point &b) {
70     return a.x*b.x+a.y*b.y;
71 }
72 double dot3(point A,point B,point C) { //点乘
73     return (C.x-A.x)*(B.x-A.x)+(C.y-A.y)*(B.y-A.y);
74 }
75
76 //向量长度
77 double length(const point &a) {
78     return sqrt(dot(a,a));
79 }
80 //两个向量的角度
81 double angle(const point &a,const point &b) {
82     return acos(dot(a,b)/length(a)/length(b));
83 }
84 //计算两个点的距离
85 double dist(const point &a,const point &b) {
86     return (a-b).norm();
87 }
88 //op 沿远点逆时针旋转角度 A
89 point rotate_point(const point &p,double A) {
90     double tx=p.x,ty=p.y;
91     return point(tx*cos(A)-ty*sin(A),tx*sin(A)+ty*cos(A));
92 }
93 double TriArea(const point &a, const point &b, const point
        ↪ &c) {
94     return fabs( cross( b - a, c - a ) ) / 2;
95 }
96 point Normal(const point &a) {
97     double L = length(a);
98     return point(-a.y/L, a.x/L);
99 }
100 //求两条直线的交点, p 和 q 分别为两条直线上的点, v
        ↪ 和 w 分别为直线的方向向量
101 point GetLineIntersection(point p, point v, point q, point
        ↪ w) {
102     point u = p - q;
103     double t = cross(w, u) / cross(v, w);
104     return p + v * t;
105 }
106 //求点 p 到直线 ab 的距离
107 double DistanceToLine(point p, point a, point b) {
108     point v1 = b - a, v2 = p - a;
109     return fabs(cross(v1,v2)) / length(v1);
110 }
111 //求点 p 到线段 ab 的距离
112 double DistanceToSegment(point p, point a, point b) {
113     if(a==b) return length(p - a);
114     point v1 = b - a, v2 = p - a, v3 = p - b;
115     if(dcmp(dot(v1,v2)) < 0) return length(v2);
116     else if(dcmp(dot(v1,v3)) > 0) return length(v3);
117     else return fabs(cross(v1,v2)) / length(v1);
118 }
119 //判断直线 a1a2 和直线 b1b2 是否规范相交
120 bool SegmentProperIntersection(point a1, point a2, point
        ↪ b1, point b2) {
121     double c1 = cross(a2-a1,b1-a1), c2 = cross(a2-a1,
        ↪ b2-a1);
122     double c3 = cross(b2-b1, a1-b1), c4 = cross(b2-b1,
        ↪ a2-b1);
123     return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) <
        ↪ 0;
124 }
125
126 //判断点 p 是否在直线 a1a2 上
127 bool OnSegment(point p, point a1, point a2) {
128     return dcmp(cross(a1-p,a2-p)) ==0 &&
        ↪ dcmp(dot(a1-p,a2-p))>0;
129 }
130 //判断线段 a1a2 和线段 b1b2 是否相交, 可以在端点处
        ↪ 相交
131 bool SegmentIntersection(point a1, point a2, point b1,
        ↪ point b2) {
132     return SegmentProperIntersection(a1, a2, b1, b2) ||
        ↪ OnSegment(a1, b1, b2) || OnSegment(a2, b1, b2);
133 }
134
135 double SegmentToSegment(point a1, point a2, point b1,
        ↪ point b2) {
136     //线段间的最短距离分为四种情况
137     double t1 = DistanceToSegment(b1, a1, a2);
138     double t2 = DistanceToSegment(b2, a1, a2);
139     double t3 = DistanceToSegment(a1, b1, b2);
140     double t4 = DistanceToSegment(a2, b1, b2);
141     return min(t1,min(t2,min(t3,t4)));
142 }
143 //使点集逆时针转
144 void antiClockSort(point *ch, int n) {
145     double res = cross(ch[1] - ch[0], ch[2] - ch[0]);
146     if(dcmp(res) >= 0) return;
147     reverse(ch, ch+n);

```

```

148 }
149
150 int ConvexHull(point* P, int cnt, point* res) {
151     sort(P, P + cnt);
152     cnt = (int) (unique(P, P + cnt) - P);
153     int m = 0;
154     for (int i = 0; i < cnt; i++) {
155         while (m > 1 && cross(res[m - 1] - res[m - 2],
156             ↪ P[i] - res[m - 2]) <= 0)
157             m--;
158         res[m++] = P[i];
159     }
160     int k = m;
161     for (int i = cnt - 2; i >= 0; i--) {
162         while (m > k && cross(res[m - 1] - res[m - 2],
163             ↪ P[i] - res[m - 2]) <= 0)
164             m--;
165         res[m++] = P[i];
166     }
167     if (cnt > 1) m--;
168     return m;
169 }
170 //判断点是否在多边形内
171 int isPointInPolygon(point p, point *a, int n) {
172     int cnt = 0;
173     for(int i=0; i<n; ++i) {
174         if(OnSegment(p, a[i], a[(i+1)%n])) return -1;
175         double k = cross(a[(i+1)%n]-a[i], p-a[i]);
176         double d1 = a[i].y - p.y;
177         double d2 = a[(i+1)].y - p.y;
178         if(k>0 &&d1<=0 &&d2>0)//点在线段的左侧
179             cnt++;
180         if(k<0 &&d2<=0 &&d1>0)//点在线段的右侧
181             cnt++;
182         //k==0, 点和线段共线的情况不考虑
183     }
184     if(cnt&1)return 1;
185     return 0;
186 }
187 //判断凸包是否相离
188 bool two_getaway_ConvexHull(point *cha, int n1, point
189     ↪ *chb, int m1) {
190     if(n1==1 && m1==1) {
191         if(cha[0]==chb[0])
192             return false;
193     } else if(n1==1 && m1==2) {
194         if(OnSegment(cha[0], chb[0], chb[1]))
195             return false;
196     } else if(n1==2 && m1==1) {
197         if(OnSegment(chb[0], cha[0], cha[1]))
198             return false;
199     } else if(n1==2 && m1==2) {
200         if(SegmentIntersection(cha[0], cha[1], chb[0],
201             ↪ chb[1]))
202             return false;
203     } else if(n1==2) {
204         for(int i=0; i<n1; ++i)
205             if(isPointInPolygon(cha[i], chb, m1))
206                 return false;
207     } else if(m1==2) {
208         for(int i=0; i<m1; ++i)
209             if(isPointInPolygon(chb[i], cha, n1))
210                 return false;
211     } else {
212         for(int i=0; i<n1; ++i) {
213             for(int j=0; j<m1; ++j) {
214                 if(SegmentIntersection(cha[i],
215                     ↪ cha[(i+1)%n1], chb[j],
216                     ↪ chb[(j+1)%m1]))
217                     return false;
218             }
219         }
220     }
221 }

```

```

215     for(int i=0; i<n1; ++i)
216         if(isPointInPolygon(cha[i], chb, m1))
217             return false;
218     for(int i=0; i<m1; ++i)
219         if(isPointInPolygon(chb[i], cha, n1))
220             return false;
221 }
222 return true;
223 }
224 //旋转卡壳求两个凸包最近距离
225 double solve(point *P, point *Q, int n, int m) {
226     if(n==1 && m==1) {
227         return length(P[0] - Q[0]);
228     } else if(n==1 && m==2) {
229         return DistanceToSegment(P[0], Q[0], Q[1]);
230     } else if(n==2 && m==1) {
231         return DistanceToSegment(Q[0], P[0], P[1]);
232     } else if(n==2 && m==2) {
233         return SegmentToSegment(P[0], P[1], Q[0], Q[1]);
234     }
235
236     int yminP = 0, ymaxQ = 0;
237     for(int i=0; i<n; ++i) if(P[i].y < P[yminP].y) yminP =
238         ↪ i;
239     for(int i=0; i<m; ++i) if(Q[i].y > Q[ymaxQ].y) ymaxQ =
240         ↪ i;
241     P[n] = P[0];
242     Q[m] = Q[0];
243     double INF2 = 1e100;
244     double arg, ans = INF2;
245
246     for(int i=0; i<n; ++i) {
247         //当叉积负转正时, 说明点 ymaxQ 就是对踵点
248         while((arg=cross(P[yminP] - P[yminP+1], Q[ymaxQ+1]
249             ↪ - Q[ymaxQ])) < -eps)
250             ymaxQ = (ymaxQ+1)%m;
251         double ret;
252
253         if(arg > eps) { //卡住第二个凸包上的点。
254             ret = DistanceToSegment(Q[ymaxQ], P[yminP],
255                 ↪ P[yminP+1]);
256             ans = min(ans, ret);
257         } else { //arg==0, 卡住第二个凸包的边
258             ret =
259                 ↪ SegmentToSegment(P[yminP], P[yminP+1], Q[ymaxQ], Q[
260                 ↪ ymaxQ+1]);
261             ans = min(ans, ret);
262         }
263         yminP = (yminP+1)%n;
264     }
265     return ans;
266 }
267
268 double mindis_twotubao(point *P, point *Q, int n, int m){
269     //尼玛, hdu2823 要判是否分离, poj3608 不判
270     //return min(solve(P, Q, n, m), solve(Q, P, m, n));
271     //判断凸包是不是相离, 如果不是, 输出 0
272     if(two_getaway_ConvexHull(P, n, Q, m)==true) return
273         ↪ min(solve(P, Q, n, m), solve(Q, P, m, n));
274     else return 0.0;
275 }
276
277 const int N=10005;
278 point a[N], b[N];
279 point cha[N], chb[N];
280 int main() {
281     int n, m;
282     while(scanf("%d%d", &n, &m) != EOF) {
283         for(int i=0; i<n; ++i)
284             ↪ scanf("%lf%lf", &a[i].x, &a[i].y);
285         for(int i=0; i<m; ++i)
286             ↪ scanf("%lf%lf", &b[i].x, &b[i].y);
287         //先求凸包
288         int n1 = ConvexHull(a, n, cha);

```

```

279     int m1 = ConvexHull(b, m, chb);
280     printf("%.4f\n", mindis_twtotubao(cha, chb, n1, m1));
281 }
282 return 0;
283 }

```

3.1.4 三角形的心

```

1 Point inCenter(const Point &A, const Point &B, const Point
  ↳ &C) { // 内心
2     double a = (B - C).len(), b = (C - A).len(), c = (A -
  ↳ B).len(),
3     s = fabs(det(B - A, C - A)),
4     r = s / p;
5     return (A * a + B * b + C * c) / (a + b + c);
6 }
7 Point circumCenter(const Point &a, const Point &b, const
  ↳ Point &c) { // 外心
8     Point bb = b - a, cc = c - a;
9     double db = bb.len2(), dc = cc.len2(), d = 2 * det(bb,
  ↳ cc);
10    return a - Point(bb.y * dc - cc.y * db, cc.x * db - bb.x
  ↳ * dc) / d;
11 }
12 Point othroCenter(const Point &a, const Point &b, const
  ↳ Point &c) { // 垂心
13     Point ba = b - a, ca = c - a, bc = b - c;
14     double Y = ba.y * ca.y * bc.y,
15     A = ca.x * ba.y - ba.x * ca.y,
16     x0 = (Y + ca.x * ba.y * b.x - ba.x * ca.y * c.x) /
  ↳ A,
17     y0 = -ba.x * (x0 - c.x) / ba.y + ca.y;
18     return Point(x0, y0);
19 }

```

3.1.5 半平面交

```

1 struct Point {
2     int quad() const { return sign(y) == 1 || (sign(y) == 0
  ↳ && sign(x) >= 0); }
3 };
4 struct Line {
5     bool include(const Point &p) const { return sign(det(b -
  ↳ a, p - a)) > 0; }
6     Line push() const { // 将半平面向外推 eps
7         const double eps = 1e-6;
8         Point delta = (b - a).turn90().norm() * eps;
9         return Line(a - delta, b - delta);
10    }
11 };
12 bool sameDir(const Line &l0, const Line &l1) { return
  ↳ parallel(l0, l1) && sign(dot(l0.b - l0.a, l1.b -
  ↳ l1.a)) == 1; }
13 bool operator < (const Point &a, const Point &b) {
14     if (a.quad() != b.quad()) {
15         return a.quad() < b.quad();
16     } else {
17         return sign(det(a, b)) > 0;
18     }
19 }
20 bool operator < (const Line &l0, const Line &l1) {
21     if (sameDir(l0, l1)) {
22         return l1.include(l0.a);
23     } else {
24         return (l0.b - l0.a) < (l1.b - l1.a);
25     }
26 }
27 bool check(const Line &u, const Line &v, const Line &w) {
28     ↳ return w.include(intersect(u, v)); }
29 vector<Point> intersection(vector<Line> &l) {
30     sort(l.begin(), l.end());
31     deque<Line> q;

```

```

31     for (int i = 0; i < (int)l.size(); ++i) {
32         if (i && sameDir(l[i], l[i - 1])) {
33             continue;
34         }
35         while (q.size() > 1 && !check(q[q.size() - 2],
  ↳ q[q.size() - 1], l[i])) q.pop_back();
36         while (q.size() > 1 && !check(q[1], q[0], l[i]))
  ↳ q.pop_front();
37         q.push_back(l[i]);
38     }
39     while (q.size() > 2 && !check(q[q.size() - 2],
  ↳ q[q.size() - 1], q[0])) q.pop_back();
40     while (q.size() > 2 && !check(q[1], q[0], q[q.size() -
  ↳ 1])) q.pop_front();
41     vector<Point> ret;
42     for (int i = 0; i < (int)q.size(); ++i)
  ↳ ret.push_back(intersect(q[i], q[(i + 1) %
  ↳ q.size()]));
43     return ret;
44 }

```

3.1.6 最大空凸包

```

1 #include <iostream>
2 #include <cmath>
3 #include <cstdio>
4 #include <algorithm>
5 using namespace std;
6 typedef double type_p;
7 const double eps = 1e-6;
8 const int maxn = 510;
9 double dp[maxn][maxn];
10 inline double eq(double x, double y)
11 {
12     return fabs(x-y)<eps;
13 }
14 inline int eq(int x, int y)
15 {
16     return x==y;
17 }
18 struct point
19 {
20     type_p x,y;
21 };
22 type_p xmult(point a, point b, point o)
23 {
24     return (a.x-o.x)*(o.y-b.y)-(a.y-o.y)*(o.x-b.x); //b at
  ↳ ao left if negative, at right if positive
25 }
26 type_p dist(point a, point b)
27 {
28     return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y);
29 }
30 point o;
31 bool cmp_angle(point a, point b)
32 {
33     if(eq(xmult(a,b,o),0.0))
34     {
35         return dist(a,o)<dist(b,o);
36     }
37     return xmult(a,o,b)>0;
38 }
39 /*
40 Input:  p: Point set
41         pn: size of the point set
42
43 Output: the area of the largest empty convex
44 */
45 double empty_convex(point *p, int pn)
46 {
47     double ans=0;
48     for(int i=0; i<pn; i++)

```



```

49     {
50         for(int j=0; j<pn; j++)
51         {
52             dp[i][j]=0;
53         }
54     }
55
56     for(int i=0; i<pn; i++)
57     {
58         int j = i-1;
59         while(j>=0 && eq(xmult(p[i], p[j],
60             ↪ o),0.0))j--;//coline
61
62         bool flag= j==i-1;
63
64         while(j>=0)
65         {
66             int k = j-1;
67             while(k >= 0 && xmult(p[i],p[k],p[j])>0)k--;
68             double area = fabs(xmult(p[i],p[j],o))/2;
69             if(k >= 0)area+=dp[j][k];
70             if(flag) dp[i][j]=area;
71             ans=max(ans,area);
72             j=k;
73         }
74         if(flag)
75         {
76             for(int j=1; j<i; j++)
77             {
78                 dp[i][j] = max(dp[i][j],dp[i][j-1]);
79             }
80         }
81         return ans;
82     }
83 double largest_empty_convex(point *p, int pn)
84 {
85     point data[maxn];
86     double ans=0;
87     for(int i=0; i<pn; i++)
88     {
89         o=p[i];
90         int dn=0;
91         for(int j=0; j<pn; j++)
92         {
93             if(p[j].y>o.y || (p[j].y==o.y&&p[j].x>o.x))
94             {
95                 data[dn++]=p[j];
96             }
97         }
98         sort(data, data+dn, cmp_angle);
99         ans=max(ans, empty_convex(data, dn));
100     }
101     return ans;
102 }
103 int main()
104 {
105     point p[110];
106     int t;
107     scanf("%d",&t);
108     while(t--)
109     {
110         int pn;
111         scanf("%d",&pn);
112         for(int i=0; i<pn; i++)
113         {
114             scanf("%lf%lf",&p[i].x,&p[i].y);
115         }
116         printf("%.1f\n",largest_empty_convex(p,pn));
117     }
118     return 0;
119 }

```

3.1.7 平面最近点对

```

1  #include <iostream>
2  #include <cstdio>
3  #include <cstdlib>
4  #include <cstring>
5  #include <algorithm>
6  #include <cmath>
7
8  using namespace std;
9
10 const double eps = 1e-8;
11 const int INF = 0x7fffffff;
12 int n;
13
14 struct Point
15 {
16     double x,y;
17     Point(double x=0, double y=0):x(x),y(y) {}
18     bool operator < (const Point& p) const
19     {
20         if(x != p.x) return x < p.x;
21         else return y < p.y;
22     }
23 }p[200000+5],temp[200000+5];
24
25 bool cmpy(Point a, Point b)
26 {
27     return a.y < b.y;
28 }
29
30 double Dis(Point a, Point b)
31 {
32     return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
33 }
34
35 double Closest_Pair(int left, int right)
36 {
37     double d = INF;
38     if(left == right)
39         return d;
40     if(left +1 == right)
41         return Dis(p[left],p[right]);
42     int mid = (left+right)>>1;
43     double d1 = Closest_Pair(left,mid);
44     double d2 = Closest_Pair(mid,right);
45     d = min(d1,d2);
46     int k = 0;
47     for(int i = left; i <= right; i++)
48     {
49         if(fabs(p[mid].x - p[i].x) <= d)
50             temp[k++] = p[i];
51     }
52     sort(temp,temp+k,cmpy);
53     for(int i = 0; i < k; i++)
54     {
55         for(int j = i+1; j < k && temp[j].y - temp[i].y < d;
56             ↪ j++)
57         {
58             double d3 = Dis(temp[i],temp[j]);
59             d = min(d,d3);
60         }
61     }
62     return d;
63 }
64
65 int main()
66 {
67     cin>>n;
68     for(int i=0; i<n; i++)
69     {
70         double a,b;
71         scanf("%lf%lf",&a,&b);

```

```

71     p[i] = Point(a,b);
72 }
73 sort(p,p+n);
74 printf("%.3f",Closest_Pair(0,n-1));
75 }

```

3.1.8 最小覆盖圆

```

1  #include<cmath>
2  #include<cstdio>
3  #include<algorithm>
4  using namespace std;
5  const double eps=1e-6;
6  struct couple
7  {
8      double x, y;
9      couple(){}
10     couple(const double &xx, const double &yy)
11     {
12         x = xx; y = yy;
13     }
14 } a[100001];
15 int n;
16 bool operator < (const couple & a, const couple & b)
17 {
18     return a.x < b.x - eps or (abs(a.x - b.x) < eps and a.y
19         ↪ < b.y - eps);
20 }
21 bool operator == (const couple & a, const couple & b)
22 {
23     return !(a < b) and !(b < a);
24 }
25 inline couple operator - (const couple &a, const couple
26     ↪ &b)
27 {
28     return couple(a.x-b.x, a.y-b.y);
29 }
30 inline couple operator + (const couple &a, const couple
31     ↪ &b)
32 {
33     return couple(a.x+b.x, a.y+b.y);
34 }
35 inline couple operator * (const couple &a, const double
36     ↪ &b)
37 {
38     return couple(a.x*b, a.y*b);
39 }
40 inline couple operator / (const couple &a, const double
41     ↪ &b)
42 {
43     return a*(1/b);
44 }
45 inline double operator * (const couple &a, const couple
46     ↪ &b)
47 {
48     return a.x*b.y-a.y*b.x;
49 }
50 inline double len(const couple &a)
51 {
52     return a.x*a.x+a.y*a.y;
53 }
54 inline double di2(const couple &a, const couple &b)
55 {
56     return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y);
57 }
58 inline double dis(const couple &a, const couple &b)
59 {
60     return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
61 }
62 struct circle
63 {
64     double r; couple c;
65 } cir;

```

```

60 inline bool inside(const couple & x)
61 {
62     return di2(x, cir.c) < cir.r*cir.r+eps;
63 }
64 inline void p2c(int x, int y)
65 {
66     cir.c.x = (a[x].x+a[y].x)/2;
67     cir.c.y = (a[x].y+a[y].y)/2;
68     cir.r = dis(cir.c, a[x]);
69 }
70 inline void p3c(int i, int j, int k)
71 {
72     couple x = a[i], y = a[j], z = a[k];
73     cir.r =
74         ↪ sqrt(di2(x,y)*di2(y,z)*di2(z,x))/fabs(x*y+y*z+z*x)/2;
75     couple t1((x-y).x, (y-z).x), t2((x-y).y, (y-z).y),
76         ↪ t3((len(x)-len(y))/2, (len(y)-len(z))/2);
77     cir.c = couple(t3*t2, t1*t3)/(t1*t2);
78 }
79 inline circle mi()
80 {
81     sort(a + 1, a + 1 + n);
82     n = unique(a + 1, a + 1 + n) - a - 1;
83     if(n == 1)
84     {
85         cir.c = a[1];
86         cir.r = 0;
87         return cir;
88     }
89     random_shuffle(a + 1, a + 1 + n);
90     p2c(1, 2);
91     for(int i = 3; i <= n; i++)
92     {
93         if(!inside(a[i]))
94         {
95             p2c(1, i);
96             for(int j = 2; j < i; j++)
97             {
98                 if(!inside(a[j]))
99                 {
100                     p2c(i, j);
101                     for(int k = 1; k < j; k++)
102                     {
103                         if(!inside(a[k]))
104                         {
105                             p3c(i,j, k);
106                         }
107                     }
108                 }
109             }
110         }
111     }
112     return cir;
113 }

```

3.1.9 多边形内部可视

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  const int N = 510;
6  const double eps = 1e-3;
7
8  struct Point {
9      double x, y;
10     Point() {}
11     Point(double x, double y): x(x), y(y) {}
12     void read() {
13         scanf("%lf %lf", &x, &y);
14     }
15     void print() const {
16         printf("%.10f %.10f\n", x, y);
17     }
18 };
19
20 Point p[N];
21 Point A, B;
22 int n, dfn;
23 int g[N][N], vis[N][N], f[N][N], v[N][N];

```

```

24
25 Point operator + (const Point & a, const Point & b) {
26     return Point(a.x + b.x, a.y + b.y);
27 }
28
29 Point operator - (const Point & a, const Point & b) {
30     return Point(a.x - b.x, a.y - b.y);
31 }
32
33 Point operator * (const Point & a, double p) {
34     return Point(a.x * p, a.y * p);
35 }
36
37 Point operator / (const Point & a, double p) {
38     return Point(a.x / p, a.y / p);
39 }
40
41 double Cross(const Point & a, const Point & b) {
42     return a.x * b.y - a.y * b.x;
43 }
44
45 double Dot(const Point & a, const Point & b) {
46     return a.x * b.x + a.y * b.y;
47 }
48
49 int dcmp(double x) {
50     if (fabs(x) < eps) return 0;
51     return x < 0 ? -1 : 1;
52 }
53
54 Point Get(const Point & P, const Point & v, const Point &
    ↪ Q, const Point & w) {
55     Point u = P - Q;
56     double t = Cross(w, u) / Cross(v, w);
57     return P + v * t;
58 }
59
60 int OnLine(const Point & a, const Point & b, const Point &
    ↪ c) {
61     return dcmp(Cross(b - a, b - c)) == 0 && dcmp(Dot(b - a,
    ↪ b - c)) < 0;
62 }
63
64 int C(const Point & P, const Point & A, const Point & Q,
    ↪ const Point & B) {
65     Point C = Get(P, A - P, Q, Q - B);
66     return OnLine(Q, C, B);
67 }
68
69 int Onleft(const Point & a, const Point & b, const Point &
    ↪ c) {
70     return dcmp(Cross(b - c, a - c)) > 0;
71 }
72
73 int visible(int x, int y) {
74     int P = (x + n - 1) % n, Q = (x + 1) % n;
75     Point u = p[y] - p[x], v = p[x] - p[P], w = p[x] - p[Q];
76     if (Onleft(p[Q], p[x], p[P])) {
77         return dcmp(Cross(v, u)) > 0 && dcmp(Cross(w, u)) < 0;
78     } else {
79         return !(dcmp(Cross(v, u)) < 0 && dcmp(Cross(w, u)) >
    ↪ 0);
80     }
81 }
82
83 int solve(int x, int y) {
84     if (vis[x][y] == dfn) return g[x][y];
85     vis[x][y] = dfn;
86     if (x == y || y == x + 1) return g[x][y] = 1;
87     for (int i = x; i + 1 <= y; i++) {
88         if (C(p[x], p[y], p[i], p[i + 1])) return g[x][y] = 0;
89     }
90     for (int i = x + 1; i < y; i++) {
91         if (OnLine(p[x], p[i], p[y])) {
92             return g[x][y] = solve(x, i) && solve(i, y);
93         }
94     }
95     if (!visible(x, y) || !visible(y, x)) return g[x][y] =
    ↪ 0;
96     return g[x][y] = 1;
97 }
98
99 void DP(int x, int y) {
100     if (v[x][y] == dfn || x > y) return;
101     v[x][y] = dfn;
102     if (x == y) {
103         f[x][y] = 1;
104         return;
105     }
106     DP(x + 1, y);
107     DP(x, y - 1);
108     f[x][y] = max(f[x][y - 1], f[x + 1][y]);
109     if (g[x][y] == 0) {
110         int z = x;
111         while (!g[z][y] && z < y) ++z;
112         DP(x, z - 1);
113         DP(z + 1, y);
114         f[x][y] = max(f[x][y], f[x][z - 1] + f[z + 1][y]);
115     }
116 }
117
118 vector<int> ans;
119
120 void DFS(int x, int y) {
121     if (x > y) return;
122     if (x == y) {
123         ans.push_back(x);
124         return;
125     }
126     if (f[x][y] == f[x][y - 1]) {
127         DFS(x, y - 1);
128     } else if (f[x][y] == f[x + 1][y]) {
129         DFS(x + 1, y);
130     } else {
131         int z = x;
132         while (!g[z][y] && z < y) ++z;
133         DFS(x, z - 1);
134         DFS(z + 1, y);
135     }
136 }
137
138 int main() {
139     freopen("hide.in", "r", stdin);
140     freopen("hide.out", "w", stdout);
141     while (scanf("%d", &n) && n) {
142         ++dfn;
143         for (int i = 0; i < n; i++) {
144             p[i].read();
145         }
146         for (int i = 1; i < n; i++) {
147             for (int j = i; j < n; j++) {
148                 g[i][j] = solve(i, j);
149             }
150         }
151         DP(1, n - 1);
152         cout << f[1][n - 1] << endl;
153         ans.clear();
154         DFS(1, n - 1);
155         for (int i = 0; i < ans.size(); i++) {
156             printf("%d%c", ans[i] + 1, i + 1 < ans.size() ? ' '
    ↪ : '\n');
157         }
158     }
159     return 0;
160 }

```


3.1.10 V 图

```

1  const int AIX = 5;
2  const int MAXM = AIX * MAXN;
3
4  struct point {
5      double x, y;
6      int index;
7      struct Edge *in;
8      point(double _x = 0, double _y = 0) : x(_x), y(_y) {}
9  };
10 inline bool operator< (const point &a, const point &b) {
11     return a.x < b.x || (sgn(a.x - b.x) == 0 && a.y < b.y);
12 }
13 inline double cross(const point &a, const point &b, const
    ↪ point &c) { return det
14     (b - a, c - a); }
15 struct Edge {
16     point *Org, *Dest;
17     Edge *Onext, *Oprev, *Dnext, *Dprev;
18 };
19 inline point* Other(const Edge *e, const point *p) {
20     ↪ return e->Org == p ?
21         e->Dest : e->Org; }
22 inline Edge* Next(const Edge *e, const point *p) { return
23     ↪ e->Org == p ? e->Onext
24         : e->Dnext; }
25 inline Edge* Prev(const Edge *e, const point *p) { return
26     ↪ e->Org == p ? e->Oprev
27         : e->Dprev; }
28 struct gEdge {
29     int u, v;
30     double w;
31     gEdge() {}
32     gEdge(int _u, int _v, double _w) : u(_u), v(_v), w(_w)
33         ↪ {}
34 };
35 inline bool operator< (const gEdge &a, const gEdge &b) {
36     ↪ return a.w < b.w; }
37 point p[MAXN], *Q[MAXN];
38 Edge mem[AIX * MAXN], *elist[AIX * MAXN];
39 static int nfree;
40 //Alloc memory
41 inline void Alloc_Memory(const int &n) {
42     nfree = AIX * n;
43     Edge *e = mem;
44     for (int i = 0; i < nfree; ++i)
45         elist[i] = e++;
46 }
47 //Add an edge to a ring of edges
48 inline void Splice(Edge *a, Edge *b, point *v) {
49     Edge *next;
50     if (a->Org == v)
51         next = a->Onext, a->Onext = b;
52     else
53         next = a->Dnext, a->Dnext = b;
54     if (next->Org == v)
55         next->Oprev = b;
56     else
57         next->Dprev = b;
58     if (b->Org == v)
59         b->Onext = next, b->Oprev = a;
60     else
61         b->Dnext = next, b->Dprev = a;
62 }
63 //Initialise a new edge
64 inline Edge *MakeEdge(point *u, point *v) {
65     Edge *e = elist[--nfree];
66     e->Onext = e->Oprev = e->Dnext = e->Dprev = e;
67     e->Org = u, e->Dest = v;
68     if (!u->in)
69         u->in = e;
70     if (!v->in)

```

```

66     v->in = e;
67     return e;
68 }
69 //Creates a new edge and adds it to two rings of edges.
70 inline Edge *Join(Edge *a, point *u, Edge *b, point *v,
    ↪ int side) {
71     Edge *e = MakeEdge(u, v);
72     if (side == 1) {
73         if (a->Org == u)
74             Splice(a->Oprev, e, u);
75         else
76             Splice(a->Dprev, e, u);
77     }
78     Splice(b, e, v);
79 }
80 else {
81     Splice(a, e, u);
82     if (b->Org == v)
83         Splice(b->Oprev, e, v);
84     else
85         Splice(b->Dprev, e, v);
86 }
87 return e;
88 }
89 //Remove an edge
90 inline void Remove(Edge *e) {
91     point *u = e->Org, *v = e->Dest;
92     if (u->in == e)
93         u->in = e->Onext;
94     if (v->in == e)
95         v->in = e->Dnext;
96     if (e->Onext->Org == u)
97         e->Onext->Oprev = e->Oprev;
98     else
99         e->Onext->Dprev = e->Oprev;
100     if (e->Oprev->Org == u)
101         e->Oprev->Onext = e->Onext;
102     else
103         e->Oprev->Dnext = e->Onext;
104     if (e->Dnext->Org == v)
105         e->Dnext->Oprev = e->Dprev;
106     else
107         e->Dnext->Dprev = e->Dprev;
108     if (e->Dprev->Org == v)
109         e->Dprev->Onext = e->Dnext;
110     else
111         e->Dprev->Dnext = e->Dnext;
112     elist[nfree++] = e;
113 }
114 //Determines the lower tangent of two triangulations
115 inline void Low_tangent(Edge *e_l, point *o_l, Edge *e_r,
    ↪ point *o_r, Edge
116     **l_low, point **OL, Edge **r_low, point **OR) {
117     point *d_l = Other(e_l, o_l), *d_r = Other(e_r, o_r);
118     while (true) {
119         if (cross(*o_l, *o_r, *d_l) < -EPS) {
120             e_l = Prev(e_l, d_l);
121             o_l = d_l;
122             d_l = Other(e_l, o_l);
123         }
124         else if (cross(*o_l, *o_r, *d_r) < -EPS) {
125             e_r = Next(e_r, d_r);
126             o_r = d_r;
127             d_r = Other(e_r, o_r);
128         }
129         else
130             break;
131     }
132     *OL = o_l, *OR = o_r;
133     *l_low = e_l, *r_low = e_r;
134 }
135 inline void Merge(Edge *lr, point *s, Edge *rl, point *u,
    ↪ Edge **tangent) {
136     double cot_L, cot_R, N1, cot_N, P1, cot_P;

```

```

136 point l1, l2, r1, r2, uu, vv;
137 point *O, *D, *OR, *OL;
138 Edge *B, *L, *R;
139 Low_tangent(lr, s, r1, u, &L, &OL, &R, &OR);
140 *tangent = B = Join(L, OL, R, OR, 0);
141 O = OL, D = OR;
142 do {
143     Edge *El = Next(B, O), *Er = Prev(B, D), *next, *prev;
144     point *l = Other(El, O), *r = Other(Er, D);
145     l1 = *O - *l, l2 = *D - *l, r1 = *O - *r, r2 = *D -
        ↪ *r;
146     double c1 = det(l1, l2), cr = det(r1, r2);
147     bool BL = c1 > EPS, BR = cr > EPS;
148     if (!BL && !BR)
149         break;
150     if (BL) {
151         double dl = dot(l1, l2);
152         cot_L = dl / c1;
153         do {
154             next = Next(El, O);
155             uu = *O - *Other(next, O);
156             vv = *D - *Other(next, O);
157             N1 = det(uu, vv);
158             if (!(N1 > EPS))
159                 break;
160             cot_N = dot(uu, vv) / N1;
161             if (cot_N > cot_L)
162                 break;
163             Remove(El);
164             El = next;
165             cot_L = cot_N;
166         }
167         while (true);
168     }
169     if (BR) {
170         double dr = dot(r1, r2);
171         cot_R = dr / cr;
172         do {
173             prev = Prev(Er, D);
174             uu = *O - *Other(prev, D);
175             vv = *D - *Other(prev, D);
176             P1 = det(uu, vv);
177             if (!(P1 > EPS))
178                 break;
179             cot_P = dot(uu, vv) / P1;
180             if (cot_P > cot_R)
181                 break;
182             Remove(Er);
183             Er = prev;
184             cot_R = cot_P;
185         }
186         while (true);
187     }
188     l = Other(El, O); r = Other(Er, D);
189     if (!BL || (BL && BR && cot_R < cot_L)) {
190         B = Join(B, O, Er, r, 0);
191         D = r;
192     }
193     else {
194         B = Join(El, l, B, D, 0);
195         O = l;
196     }
197 }
198 while (true);
199 }
200 inline void Divide(int s, int t, Edge **L, Edge **R) {
201     Edge *a, *b, *c, *ll, *lr, *rr, *tangent;
202     int n = t - s + 1;
203     if (n == 2)
204         *L = *R = MakeEdge(Q[s], Q[t]);
205     else if (n == 3) {
206         a = MakeEdge(Q[s], Q[s + 1]);
207         b = MakeEdge(Q[s + 1], Q[t]);
208         Ssplice(a, b, Q[s + 1]);
209         double v = cross(*Q[s], *Q[s + 1], *Q[t]);
210         if (v > EPS) {
211             c = Join(a, Q[s], b, Q[t], 0);
212             *L = a, *R = b;
213         }
214         else if (v < -EPS) {
215             c = Join(a, Q[s], b, Q[t], 1);
216             *L = c, *R = c;
217         }
218         else
219             *L = a, *R = b;
220     }
221     else if (n > 3) {
222         int split = (s + t) / 2;
223         Divide(s, split, &ll, &lr);
224         Divide(split + 1, t, &rl, &rr);
225         Merge(lr, Q[split], rl, Q[split + 1], &tangent);
226         if (tangent->Org == Q[s])
227             ll = tangent;
228         if (tangent->Dest == Q[t])
229             rr = tangent;
230         *L = ll; *R = rr;
231     }
232 }
233 int task, n, m, k, root[MAXN];
234 gEdge E[MAXM], MST[MAXN];
235 inline int Make_Graph() {
236     Edge *start, *e;
237     int M = 0;
238     point *u, *v;
239     for(int i = 0; i < n; ++i) {
240         u = p + i;
241         start = e = u->in;
242         do {
243             v = Other(e, u);
244             if (u < v)
245                 E[M++] = gEdge(u - p + 1, v - p + 1, dis(*u, *v));
246             e = Next(e, u);
247         }
248         while(e != start);
249     }
250     return M;
251 }
252 int find_root(const int &x) { return root[x] ? root[x] =
    ↪ find_root(root[x]) : x;
253 }
254 inline bool merge(const int &x, const int &y) {
255     int p = find_root(x), q = find_root(y);
256     if (p != q) {
257         root[p] = q;
258         return true;
259     }
260     else
261         return false;
262 }
263 inline void kruskal(gEdge *E, int m, int n, gEdge* MST) {
264     for (int i = 1; i <= n; ++i)
265         root[i] = 0;
266     sort(E, E + m);
267     int tot = 0;
268     for (int i = 0; i < m; ++i)
269         if (merge(E[i].u, E[i].v))
270             MST[tot++] = E[i];
271 }
272 inline void MinimumEuclideanSpanningTree(point* p, int n,
    ↪ gEdge* MST) {
273     Alloc_Memory(n);
274     sort(p, p + n);
275     for (int i = 0; i < n; ++i)
276         Q[i] = p + i;
277     Edge *L, *R;
278     Divide(0, n - 1, &L, &R);

```

```

279 m = Make_Graph();
280 kruskal(E, m, n, MST);
281 }
282 int main() {
283     for (scanf("%d", &task); task--; ) {
284         scanf("%d", &k);
285         for (n = 0; scanf("%lf", &p[n].x) == 1 && p[n].x !=
                ↪ -1; ++n) {
286             scanf("%lf", &p[n].y);
287             p[n].in = NULL;
288             p[n].index = n;
289         }
290         if (n == 1) {
291             printf("0\n");
292             continue;
293         }
294         MinimumEuclideanSpaningTree(p, n, MST);
295         printf("%d\n", int(ceil(k > n ? 0 : MST[n - k - 1].w)
                ↪ + EPS));
296     }
297 }

```

3.2 三维

3.2.1 三维点类

```

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) * cosw + 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 }

```

3.2.2 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 0(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 }

```

3.2.3 最小覆盖球

```

1 #include<iostream>
2 #include<cstring>
3 #include<algorithm>
4 #include<cstdio>
5 #include<cmath>
6
7 using namespace std;

```

```

8
9  const int eps = 1e-8;
10
11 struct Tpoint
12 {
13     double x, y, z;
14 };
15
16 int npoint, nouter;
17
18 Tpoint pt[200000], outer[4], res;
19 double radius, tmp;
20 inline double dist(Tpoint p1, Tpoint p2) {
21     double dx=p1.x-p2.x, dy=p1.y-p2.y, dz=p1.z-p2.z;
22     return ( dx*dx + dy*dy + dz*dz );
23 }
24 inline double dot(Tpoint p1, Tpoint p2) {
25     return p1.x*p2.x + p1.y*p2.y + p1.z*p2.z;
26 }
27 void ball() {
28     Tpoint q[3]; double m[3][3], sol[3], L[3], det;
29     int i, j;
30     res.x = res.y = res.z = radius = 0;
31     switch ( nouter ) {
32         case 1: res=outer[0]; break;
33         case 2:
34             res.x=(outer[0].x+outer[1].x)/2;
35             res.y=(outer[0].y+outer[1].y)/2;
36             res.z=(outer[0].z+outer[1].z)/2;
37             radius=dist(res, outer[0]);
38             break;
39         case 3:
40             for (i=0; i<2; ++i) {
41                 q[i].x=outer[i+1].x-outer[0].x;
42                 q[i].y=outer[i+1].y-outer[0].y;
43                 q[i].z=outer[i+1].z-outer[0].z;
44             }
45             for (i=0; i<2; ++i) for (j=0; j<2; ++j)
46                 m[i][j]=dot(q[i], q[j])*2;
47             for (i=0; i<2; ++i) sol[i]=dot(q[i], q[i]);
48             if (fabs(det=m[0][0]*m[1][1]-m[0][1]*m[1][0])<eps)
49                 return;
50             L[0]=(sol[0]*m[1][1]-sol[1]*m[0][1])/det;
51             L[1]=(sol[1]*m[0][0]-sol[0]*m[1][0])/det;
52             res.x=outer[0].x+q[0].x*L[0]+q[1].x*L[1];
53             res.y=outer[0].y+q[0].y*L[0]+q[1].y*L[1];
54             res.z=outer[0].z+q[0].z*L[0]+q[1].z*L[1];
55             radius=dist(res, outer[0]);
56             break;
57         case 4:
58             for (i=0; i<3; ++i) {
59                 q[i].x=outer[i+1].x-outer[0].x;
60                 q[i].y=outer[i+1].y-outer[0].y;
61                 q[i].z=outer[i+1].z-outer[0].z;
62                 sol[i]=dot(q[i], q[i]);
63             }
64             for (i=0; i<3; ++i)
65                 for (j=0; j<3; ++j) m[i][j]=dot(q[i], q[j])*2;
66             det= m[0][0]*m[1][1]*m[2][2]
67                 + m[0][1]*m[1][2]*m[2][0]
68                 + m[0][2]*m[1][0]*m[2][1]
69                 - m[0][2]*m[1][1]*m[2][0]
70                 - m[0][1]*m[1][0]*m[2][2]
71                 - m[0][0]*m[1][2]*m[2][1];
72             if ( fabs(det)<eps ) return;
73             for (j=0; j<3; ++j) {
74                 for (i=0; i<3; ++i) m[i][j]=sol[i];
75                 L[j]=( m[0][0]*m[1][1]*m[2][2]
76                     + m[0][1]*m[1][2]*m[2][0]
77                     + m[0][2]*m[1][0]*m[2][1]
78                     - m[0][2]*m[1][1]*m[2][0]
79                     - m[0][1]*m[1][0]*m[2][2]
80                     - m[0][0]*m[1][2]*m[2][1])

```

```

81         ) / det;
82         for (i=0; i<3; ++i)
83             m[i][j]=dot(q[i], q[j])*2;
84     }
85     res=outer[0];
86     for (i=0; i<3; ++i) {
87         res.x += q[i].x * L[i];
88         res.y += q[i].y * L[i];
89         res.z += q[i].z * L[i];
90     }
91     radius=dist(res, outer[0]);
92 }
93 }
94 void minball(int n) {
95     ball();
96     //printf("(%.3lf,%.3lf,%.3lf) %.3lf\n",
97         ↪ res.x, res.y, res.z, radius);
98     if ( nouter<4 )
99         for (int i=0; i<n; ++i)
100             if (dist(res, pt[i])-radius>eps) {
101                 outer[nouter]=pt[i];
102                 ++nouter;
103                 minball(i);
104                 --nouter;
105                 if (i>0) {
106                     Tpoint Tt = pt[i];
107                     memmove(&pt[1], &pt[0], sizeof(Tpoint)*i);
108                     pt[0]=Tt;
109                 }
110             }
111 void solve()
112 {
113     for (int i=0; i<npoint; i++)
114         ↪ scanf("%lf%lf%lf", &pt[i].x, &pt[i].y, &pt[i].z);
115     random_shuffle(pt, pt + npoint);
116     radius=-1;
117     for (int i=0; i<npoint; i++){
118         if (dist(res, pt[i])-radius>eps){
119             nouter=1;
120             outer[0]=pt[i];
121             minball(i);
122         }
123     }
124     printf("%.5f\n", sqrt(radius));
125 }
126 int main(){
127     for( ; cin >> npoint && npoint; )
128         solve();
129     return 0;
130 }

```

4. 字符串

4.1 AC 自动机

```

1 int newnode()
2 {
3     ++tot;
4     memset(ch[tot], 0, sizeof(ch[tot]));
5     fail[tot] = 0;
6     dep[tot] = 0;
7     par[tot] = 0;
8
9     return tot;
10 }
11 void insert(char *s, int x)
12 {
13     if(*s == '\0') return;
14     else
15     {
16         int &y = ch[x][*s - 'a'];
17

```

```

18     if(y == 0)
19     {
20         y = newnode();
21         par[y] = x;
22         dep[y] = dep[x] + 1;
23     }
24
25     insert(s + 1, y);
26 }
27
28 void build()
29 {
30     int line[maxn];
31     int f = 0, r = 0;
32
33     fail[root] = root;
34
35     for(int i = 0; i < alpha; i++)
36     {
37         if(ch[root][i])
38         {
39             fail[ch[root][i]] = root;
40             line[r++] = ch[root][i];
41         }
42         else
43         {
44             ch[root][i] = root;
45         }
46     }
47
48     while(f != r)
49     {
50         int x = line[f++];
51
52         for(int i = 0; i < alpha; i++)
53         {
54             if(ch[x][i])
55             {
56                 fail[ch[x][i]] = ch[fail[x]][i];
57                 line[r++] = ch[x][i];
58             }
59             else
60             {
61                 ch[x][i] = ch[fail[x]][i];
62             }
63         }
64     }
65 }

```

4.2 后缀数组

```

1  const int MAXN = MAXL * 2 + 1;
2  int a[MAXN], x[MAXN], y[MAXN], c[MAXN], sa[MAXN],
    ↪ rank[MAXN], height[MAXN];
3  void calc_sa(int n) {
4      int m = alphabet, k = 1;
5      memset(c, 0, sizeof(*c) * (m + 1));
6      for (int i = 1; i <= n; ++i) c[x[i]] = a[i]++;
7      for (int i = 1; i <= m; ++i) c[i] += c[i - 1];
8      for (int i = n; i; --i) sa[c[x[i]]--] = i;
9      for (; k <= n; k <= 1) {
10         int tot = k;
11         for (int i = n - k + 1; i <= n; ++i) y[i - n + k] = i;
12         for (int i = 1; i <= n; ++i)
13             if (sa[i] > k) y[++tot] = sa[i] - k;
14         memset(c, 0, sizeof(*c) * (m + 1));
15         for (int i = 1; i <= n; ++i) c[x[i]]++;
16         for (int i = 1; i <= m; ++i) c[i] += c[i - 1];
17         for (int i = n; i; --i) sa[c[x[y[i]]]--] = y[i];
18         for (int i = 1; i <= n; ++i) y[i] = x[i];
19         tot = 1; x[sa[1]] = 1;
20         for (int i = 2; i <= n; ++i) {

```

```

21         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;
22         x[sa[i]] = tot;
23     }
24     if (tot == n) break; else m = tot;
25 }
26
27 void calc_height(int n) {
28     for (int i = 1; i <= n; ++i) rank[sa[i]] = i;
29     for (int i = 1; i <= n; ++i) {
30         height[rank[i]] = max(0, height[rank[i - 1]] - 1);
31         if (rank[i] == 1) continue;
32         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 }

```

4.3 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 <= 1 * 2; ++i) memset(trans[i], 0,
            ↪ sizeof(trans[i]));
8      memset(par, 0, sizeof(*par) * (1 * 2 + 1));
9      memset(mxl, 0, sizeof(*mxl) * (1 * 2 + 1));
10     memset(size, 0, sizeof(*size) * (1 * 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) * (1 * 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 }

```

4.4 广义后缀自动机

```

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 }

```

```

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 }

```

4.5 manacher

```

1 void Manacher(std::string s, int p[])
2 {
3     string t = "$#";
4
5     for (int i = 0; i < s.size(); i++)
6     {
7         t += s[i];
8         t += "#";
9     }
10
11     std::vector<int> p(t.size(), 0);
12
13     int mx = 0, id = 0;
14
15     for (int i = 1; i < t.size(); i++)
16     {
17         p[i] = mx > i ? min(p[2 * id - i], mx - i) : 1;
18         while (t[i + p[i]] == t[i - p[i]]) ++p[i];
19
20         if (mx < i + p[i])
21         {
22             mx = i + p[i];
23             id = i;
24         }
25     }
26 }

```

4.6 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) {

```

4.7 循环串的最小表示

```

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 }

```

5. 数据结构

5.1 可并堆

```

1  int merge(int x, int y)
2  {
3      //p[i] 结点 i 的权值, 这里是维护大根堆
4      //d[i] 在 i 的子树中, i 到右叶子结点的最远距离.
5
6      if(!x) return y;
7      if(!y) return x;
8
9      if(p[x] < p[y]) std::swap(x, y);
10
11     r[x] = merge(r[x], y);
12     if(r[x]) fa[r[x]] = x;
13
14     if(d[l[x]] < d[r[x]]) std::swap(l[x], r[x]); //调整树
        ↳ 的结构, 使其满足左偏性质
15
16     d[x] = d[r[x]] + 1;
17     return x;
18 }

```

5.2 KD-Tree

```

1  long long norm(const long long &x) {
2      // For manhattan distance
3      return std::abs(x);
4      // For euclid distance
5      return x * x;
6  }
7
8  struct Point {
9      int x, y, id;
10
11     const int& operator [] (int index) const {
12         if (index == 0) {
13             return x;
14         } else {
15             return y;
16         }
17     }
18
19     friend long long dist(const Point &a, const Point &b)
        ↳ {
20         long long result = 0;
21         for (int i = 0; i < 2; ++i) {
22             result += norm(a[i] - b[i]);
23         }
24         return result;
25     }
26 } point[N];
27
28 struct Rectangle {
29     int min[2], max[2];
30
31     Rectangle() {
32         min[0] = min[1] = INT_MAX; // sometimes int is
        ↳ not enough
33         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 separator;
57     Rectangle rectangle;
58     int child[2];
59
60     void reset(const Point &p) {
61         separator = 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].separator)) {
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
112 //               ↪ >;min and max
113 void query(int x, const Point &p, std::pair<long long,
114           ↪ int> &answer, int type = 1) {
115     pivot = type;
116     if (x == 0 || tree[x].rectangle.dist(p) >
117         ↪ answer.first) {
118         return;
119     }
120     answer = std::min(answer,
121         std::make_pair(dist(tree[x].separator, p),
122             ↪ tree[x].separator.id));
123     if (compare(p, tree[x].separator)) {
124         query(tree[x].child[0], p, answer, type ^ 1);
125         query(tree[x].child[1], p, answer, type ^ 1);
126     } else {
127         query(tree[x].child[1], p, answer, type ^ 1);
128         query(tree[x].child[0], p, answer, type ^ 1);
129     }
130 }
131
132 std::priority_queue<std::pair<long long, int> > answer;
133
134 void query(int x, const Point &p, int k, int type = 1) {
135     pivot = type;
136     if (x == 0 || (int)answer.size() == k &&
137         ↪ tree[x].rectangle.dist(p) > answer.top().first) {
138         return;
139     }
140     answer.push(std::make_pair(dist(tree[x].separator, p),
141         ↪ tree[x].separator.id));
142     if ((int)answer.size() > k) {
143         answer.pop();
144     }
145     if (compare(p, tree[x].separator)) {
146         query(tree[x].child[0], p, k, type ^ 1);
147         query(tree[x].child[1], p, k, type ^ 1);
148     } else {
149         query(tree[x].child[1], p, k, type ^ 1);
150         query(tree[x].child[0], p, k, type ^ 1);
151     }
152 }

```

5.3 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),
6         ↪ size(size), rev(0), tag(0){}
7     void dntag();
8     Node* update(){
9         mn = min(ch[0] -> mn, min(key, ch[1] -> mn));
10        size = ch[0] -> size + 1 + ch[1] -> size;
11        return this;
12    }
13 };
14 typedef pair<Node*, Node*> Pair;
15 Node *null, *root;
16 void Node::dntag(){
17     if(rev){
18         for(int i = 0; i < 2; i++){
19             if(ch[i] != null){
20                 ch[i] -> rev ^= 1;
21                 swap(ch[i] -> ch[0], ch[i] -> ch[1]);
22             }
23         }
24         rev = 0;
25     }
26 }

```

```

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         }
32         tag = 0;
33     }
34 }
35 int r(){
36     static int s = 3023192386;
37     return (s += (s << 3) + 1) & (~0u >> 1);
38 }
39 bool random(int x, int y){
40     return r() % (x + y) < x;
41 }
42 Node* merge(Node *p, Node *q){
43     if(p == null) return q;
44     if(q == null) return p;
45     p -> dntag();
46     q -> dntag();
47     if(random(p -> size, q -> size)){
48         p -> ch[1] = merge(p -> ch[1], q);
49         return p -> update();
50     }else{
51         q -> ch[0] = merge(p, q -> ch[0]);
52         return q -> update();
53     }
54 }
55 Pair split(Node *x, int n){
56     if(x == null) return make_pair(null, null);
57     x -> dntag();
58     if(n <= x -> ch[0] -> size){
59         Pair ret = split(x -> ch[0], n);
60         x -> ch[0] = ret.second;
61         return make_pair(ret.first, x -> update());
62     }
63     Pair ret = split(x -> ch[1], n - x -> ch[0] -> size -
64         ↪ 1);
65     x -> ch[1] = ret.first;
66     return make_pair(x -> update(), ret.second);
67 }
68 pair<Node*, Pair> get_segment(int l, int r){
69     Pair ret = split(root, l - 1);
70     return make_pair(ret.first, split(ret.second, r - l +
71         ↪ 1));
72 }
73
74 int main(){
75     null = new Node(INF, INF, 0);
76     null -> ch[0] = null -> ch[1] = null;
77     root = null;
78 }

```

5.4 Splay

```

1 template<class T>void checkmin(T &x, T y)
2 {
3     if(y < x) x = y;
4 }
5 struct Node
6 {
7     Node *c[2], *fa;
8     int size, rev;
9
10    LL val, add, min;
11
12    Node *init(LL v)
13    {
14        val = min = v;
15        add = rev = 0;
16    }
17 }

```



```

16     c[0] = c[1] = fa = NULL;
17     size = 1;
18
19     return this;
20 }
21 void rvs()
22 {
23     std::swap(c[0], c[1]);
24     rev ^= 1;
25 }
26 void inc(LL x)
27 {
28     val += x;
29     add += x;
30     min += x;
31 }
32 void pushdown()
33 {
34     if(rev)
35     {
36         if(c[0]) c[0]->rvs();
37         if(c[1]) c[1]->rvs();
38         rev = 0;
39     }
40     if(add)
41     {
42         if(c[0]) c[0]->inc(add);
43         if(c[1]) c[1]->inc(add);
44         add = 0;
45     }
46 }
47 void update()
48 {
49     min = val;
50     if(c[0]) checkmin(min, c[0]->min);
51     if(c[1]) checkmin(min, c[1]->min);
52
53     size = 1;
54     if(c[0]) size += c[0]->size;
55     if(c[1]) size += c[1]->size;
56 }
57 } *root;
58
59 Node* newnode(LL x)
60 {
61     static Node pool[maxs], *p = pool;
62
63     return (++p)->init(x);
64 }
65
66 void setc(Node *x, int t, Node *y)
67 {
68     x->c[t] = y;
69     if(y) y->fa = x;
70 }
71 Node *find(int k)
72 {
73     Node *now = root;
74
75     while(true)
76     {
77         now->pushdown();
78
79         int t = (now->c[0] ? now->c[0]->size : 0) + 1;
80
81         if(t == k) break;
82
83         if(t > k) now = now->c[0];
84         else now = now->c[1], k -= t;
85     }
86
87     return now;
88 }
89
90 void rotate(Node *x, Node* &k)
91 {
92     Node *y = x->fa, *z = y->fa;
93
94     if(y != k) z->c[z->c[1] == y] = x;
95     else k = x;
96
97     x->fa = z;
98
99     int i = (y->c[1] == x);
100
101     setc(y, i, x->c[i ^ 1]);
102     setc(x, i ^ 1, y);
103
104     y->update(), x->update();
105 }
106 void spaly(Node *x, Node* &k)
107 {
108     static Node *st[maxs];
109     int top = 0;
110     Node *y, *z;
111
112     y = x;
113     while(y != k) st[++top] = y, y = y->fa;
114     st[++top] = y;
115
116     while(top) st[top]->pushdown(), top--;
117
118     while(x != k)
119     {
120         y = x->fa, z = y->fa;
121
122         if(y != k)
123         {
124             if((y == z->c[1]) ^ (x == y->c[1])) rotate(x, k);
125             else rotate(y, k);
126         }
127
128         rotate(x, k);
129     }
130 }
131 Node *subtree(int l, int r)
132 {
133     assert(++l <= ++r);
134     spaly(find(l - 1), root);
135     spaly(find(r + 1), root->c[1]);
136
137     return root->c[1]->c[0];
138 }
139 void ins(int pos, int v)
140 {
141     pos++;
142     spaly(find(pos), root);
143     spaly(find(pos + 1), root->c[1]);
144     setc(root->c[1], 0, newnode(v));
145     root->c[1]->update();
146     root->update();
147 }
148 void del(int pos)
149 {
150     pos++;
151     spaly(find(pos - 1), root);
152     spaly(find(pos + 1), root->c[1]);
153     root->c[1]->c[0] = NULL;
154     root->c[1]->update();
155     root->update();
156 }
157 void init()
158 {
159     root = newnode(0);
160     setc(root, 1, newnode(0));
161     root->update();
162 }

```

5.5 Link cut Tree

```

1 inline void reverse(int x) {
2     tr[x].rev ^= 1; swap(tr[x].c[0], tr[x].c[1]);
3 }
4
5 inline void rotate(int x, int k) {
6     int y = tr[x].fa, z = tr[y].fa;
7     tr[x].fa = z; tr[z].c[tr[z].c[1] == y] = x;
8     tr[tr[x].c[k ^ 1]].fa = y; tr[y].c[k] = tr[x].c[k ^
9         ↪ 1];
10    tr[x].c[k ^ 1] = y; tr[y].fa = x;
11 }
12
13 inline void splay(int x, int w) {
14     int z = x; pushdown(x);
15     while (tr[x].fa != w) {
16         int y = tr[x].fa; z = tr[y].fa;
17         if (z == w) {
18             pushdown(z = y); pushdown(x);
19             rotate(x, tr[y].c[1] == x);
20             update(y); update(x);
21         } else {
22             pushdown(z); pushdown(y); pushdown(x);
23             int t1 = tr[y].c[1] == x, t2 = tr[z].c[1] == y;
24             if (t1 == t2) rotate(y, t2), rotate(x, t1);
25             else rotate(x, t1), rotate(x, t2);
26             update(z); update(y); update(x);
27         }
28     }
29     update(x);
30     if (x != z) par[x] = par[z], par[z] = 0;
31 }
32
33 inline void access(int x) {
34     for (int y = 0; x; y = x, x = par[x]) {
35         splay(x, 0);
36         if (tr[x].c[1]) par[tr[x].c[1]] = x, tr[tr[x].c[1]].fa
37             ↪ = 0;
38         tr[x].c[1] = y; par[y] = 0; tr[y].fa = x; update(x);
39     }
40 }
41
42 inline void makeroot(int x) {
43     access(x); splay(x, 0); reverse(x);
44 }
45
46 inline void link(int x, int y) {
47     makeroot(x); par[x] = y;
48 }
49
50 inline void cut(int x, int y) {
51     access(x); splay(y, 0);
52     if (par[y] != x) swap(x, y), access(x), splay(y, 0);
53     par[y] = 0;
54 }
55
56 inline void split(int x, int y) { // x will be the root
57     ↪ of the tree
58     makeroot(y); access(x); splay(x, 0);
59 }

```

5.6 树上莫队

```

1 void dfs(int u)
2 {
3     dep[u] = dep[fa[u][0]] + 1;
4     for(int i = 1; i < logn; i++)
5         fa[u][i] = fa[fa[u][i - 1]][i - 1];
6
7     stk.push(u);
8     for(int i = 0; i < vec[u].size(); i++)
9     {

```

```

10         int v = vec[u][i];
11
12         if(v == fa[u][0]) continue;
13
14         fa[v][0] = u, dfs(v);
15
16         size[u] += size[v];
17
18         if(size[u] >= bufsize)
19         {
20             ++bcnt;
21
22             while(stk.top() != u)
23             {
24                 block[stk.top()] = bcnt;
25                 stk.pop();
26             }
27
28             size[u] = 0;
29         }
30     }
31
32     size[u]++;
33 }
34
35 void prework()
36 {
37     dfs(1);
38
39     ++bcnt;
40     while(!stk.empty())
41     {
42         block[stk.top()] = bcnt;
43         stk.pop();
44     }
45 }
46
47 void rev(int u)
48 {
49     now -= (cnt[val[u]] > 0);
50
51     if(used[u])
52     {
53         cnt[val[u]]--;
54         used[u] = false;
55     }
56     else
57     {
58         cnt[val[u]]++;
59         used[u] = true;
60     }
61
62     now += (cnt[val[u]] > 0);
63 }
64
65 void move(int &x, int y, int z)
66 {
67     int fwd = y;
68
69     rev(getlca(x, z));
70     rev(getlca(y, z));
71
72     while(x != y)
73     {
74         if(dep[x] < dep[y]) std::swap(x, y);
75
76         rev(x), x = fa[x][0];
77     }
78
79     x = fwd;
80 }
81
82 void solve()
83 {
84     std::sort(query + 1, query + m + 1);
85
86     int L = 1, R = 1;

```

```

83 rev(1);
84
85 for(int i = 1; i <= m; i++)
86 {
87     int l = query[i].u;
88     int r = query[i].v;
89
90     move(L, l, R);
91     move(R, r, L);
92
93     ans[query[i].t] = now;
94 }
95 }

```

5.7 CDQ 分治

```

1 struct Node
2 {
3     int x, y, z, idx;
4
5     friend bool operator == (const Node &a, const Node &b)
6     {
7         return a.x == b.x && a.y == b.y && a.z == b.z;
8     }
9     friend bool operator < (const Node &a, const Node &b)
10    {
11        return a.y < b.y;
12    }
13 } triple[maxn];
14
15 bool cmpx(const Node &a, const Node &b)
16 {
17     if(a.x != b.x) return a.x < b.x;
18     if(a.y != b.y) return a.y < b.y;
19     return a.z < b.z;
20 }
21
22 void solve(int l, int r)
23 {
24     if(l == r) return;
25
26     int mid = (l + r) >> 1;
27
28     solve(l, mid);
29
30     static std::pair<Node, int> Lt[maxn], Rt[maxn];
31     int Ls = 0, Rs = 0;
32
33     for(int i = l; i <= mid; i++)
34         Lt[++Ls] = std::make_pair(triple[i], i);
35     for(int i = mid + 1; i <= r; i++)
36         Rt[++Rs] = std::make_pair(triple[i], i);
37
38     int pos = 1;
39
40     std::sort(Lt + 1, Lt + Ls + 1);
41     std::sort(Rt + 1, Rt + Rs + 1);
42
43     backup.clear();
44     for(int i = 1; i <= Rs; i++)
45     {
46         while(pos <= Ls && !Rt[i].first < Lt[pos].first)
47         {
48             insert(Lt[pos].first.z, 1);
49
50             pos++;
51         }
52
53         f[Rt[i].second] += query(Rt[i].first.z);
54     }
55 }
56

```

```

57     for(int i = 0; i < backup.size(); i++) pre[backup[i]] =
58         ↳ 0;
59     solve(mid + 1, r);
60 }

```

5.8 整体二分

```

1 void solve(int l, int r, std::vector<int> q)
2 {
3     if(l == r || q.empty())
4     {
5         for(int i = 0; i < q.size(); i++)
6         {
7             ans[q[i]] = 1;
8         }
9     }
10    else
11    {
12        int mid = (l + r) >> 1;
13
14        backup.clear();
15
16        for(int i = l; i <= mid; i++)
17        {
18            Event e = event[i];
19
20            if(e.l <= e.r)
21            {
22                add(e.l, e.v);
23                add(e.r + 1, -e.v);
24            }
25            else
26            {
27                add(l, e.v);
28                add(e.r + 1, -e.v);
29                add(e.l, e.v);
30            }
31        }
32
33        std::vector<int> qL, qR;
34
35        for(int i = 0; i < q.size(); i++)
36        {
37            LL val = 0;
38
39            for(int j = 0; j < vec[q[i]].size(); j++)
40            {
41                val += count(vec[q[i]][j]);
42
43                if(val >= p[q[i]]) break;
44            }
45
46            if(cnt[q[i]] + val >= p[q[i]])
47            {
48                qL.push_back(q[i]);
49            }
50            else
51            {
52                cnt[q[i]] += val;
53                qR.push_back(q[i]);
54            }
55        }
56
57        for(int i = 0; i < backup.size(); i++) sum[backup[i]]
58            ↳ = 0;
59        solve(l, mid, qL);
60        solve(mid + 1, r, qR);
61    }
62 }

```

6. 图论

6.1 2-SAT

```

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 }

```

6.2 2-SAT (tarjan)

```

1 template<class TAT>void checkmin(TAT &x,TAT y)
2 {
3     if(y < x) x = y;
4 }
5 void tarjan(int u)
6 {
7     dfn[u] = low[u] = ++dt;
8     flag[u] = true;
9     stk.push(u);
10
11     for(int i = 0; i < vec[u].size(); i++)
12     {
13         int v = vec[u][i];
14
15         if(!dfn[v])
16         {
17             tarjan(v);
18             checkmin(low[u], low[v]);
19         }
20         else if(flag[v])
21         {
22             checkmin(low[u], dfn[v]);
23         }
24     }
25
26     if(low[u] == dfn[u])
27     {

```

```

28         ++bcnt;
29         while(stk.top() != u)
30         {
31             block[stk.top()] = bcnt;
32             flag[stk.top()] = false;
33             stk.pop();
34         }
35
36         block[u] = bcnt;
37         flag[u] = false;
38         stk.pop();
39     }
40 }
41 bool solve()
42 {
43     for(int i = 1; i <= 2 * n; i++)
44         if(!dfn[i]) tarjan(i);
45
46     bool ans = true;
47
48     for(int i = 1; i <= n; i++)
49         if(block[2 * i] == block[2 * i - 1])
50         {
51             ans = false;
52             break;
53         }
54
55     return ans;
56 }

```

6.3 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;

```

6.4 点双连通分量

```

1  const bool BCC_VERTEX = 0, BCC_EDGE = 1;
2  struct BCC { // N = N0 + M0. Remember to call
3      ↪ init(&raw_graph).
4      Graph *g, forest; // g is raw graph ptr.
5      int dfn[N], DFN, low[N];
6      int stack[N], top;
7      int expand_to[N]; // Where edge i is expanded to in
8      ↪ expanded graph.
9      // Vertex i expanded to i.
10     int compress_to[N]; // Where vertex i is compressed to.
11     bool vertex_type[N], cut[N], compress_cut[N], branch[M];
12     //std::vector<int> BCC_component[N]; // Cut vertex
13     ↪ belongs to none.
14     __inline void init(Graph *raw_graph) {
15         g = raw_graph;
16     }
17
18 void DFS(int u, int pe) {
19     dfn[u] = low[u] = ++DFN; cut[u] = false;
20     if (!~g->adj[u]) {
21         cut[u] = 1;
22         compress_to[u] = forest.new_node();
23         compress_cut[compress_to[u]] = 1;
24     }
25     for (int e = g->adj[u]; ~e; e = g->nxt[e]) {
26         int v = g->v[e];
27         if ((e ^ pe) > 1 && dfn[v] > 0 && dfn[v] < dfn[u]) {
28             stack[top++] = e;
29             low[u] = std::min(low[u], dfn[v]);
30         }
31         else if (!dfn[v]) {
32             stack[top++] = e; branch[e] = 1;
33             DFS(v, e);
34             low[u] = std::min(low[v], low[u]);
35             if (low[v] >= dfn[u]) {
36                 if (!cut[u]) {
37                     cut[u] = 1;
38                     compress_to[u] = forest.new_node();
39                     compress_cut[compress_to[u]] = 1;
40                 }
41                 int cc = forest.new_node();
42                 forest.bi_ins(compress_to[u], cc);
43                 compress_cut[cc] = 0;
44                 //BCC_component[cc].clear();
45                 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 ...

```

6.5 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],
4      ↪ top, dfs_clock;
5      // tot[] is the size of each BCC, belong[] is the BCC
6      ↪ that each node belongs to
7      pair<int, int> ori[M]; // bridge in raw_graph(raw node)
8      bool is_bridge[M];
9      __inline void init(Graph *raw_graph) {
10         g = raw_graph;
11         memset(is_bridge, false, sizeof(*is_bridge) * g->e);
12         memset(vis + g->base, 0, sizeof(*vis) * g->n);
13     }
14
15 void tarjan(int u, int from) {
16     dfn[u] = low[u] = ++dfs_clock; vis[u] = 1;
17     ↪ stack[top++] = u;
18     for (int p = g->adj[u]; ~p; p = g->nxt[p]) {
19         if ((p ^ 1) == from) continue;
20         int v = g->v[p];
21         if (vis[v]) {
22             if (vis[v] == 1) low[u] = min(low[u], dfn[v]);
23         } else {
24             tarjan(v, p);
25             low[u] = min(low[u], low[v]);
26             if (low[v] > dfn[u]) is_bridge[p / 2] = true;
27         }
28     }
29     if (dfn[u] != low[u]) return;
30     tot[forest.new_node()] = 0;
31     do {
32         belong[stack[top]] = forest.n;
33         vis[stack[top]] = 2;
34         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;

```

6.6 最小树形图

```

1  const int MAXN, INF; // INF >= sum( W_ij )
2  int from[MAXN + 10][MAXN * 2 + 10], n, m, edge[MAXN +
   10][MAXN * 2 + 10];
3  int sel[MAXN * 2 + 10], fa[MAXN * 2 + 10], vis[MAXN * 2 +
   10];
4  int getfa(int x){if(x == fa[x]) return x; return fa[x] =
   getfa(fa[x]);}
5  void liuzhu(){ // 1-base: root is 1, answer = (sel[i], i)
   for i in [2..n]
6     fa[1] = 1;
7     for(int i = 2; i <= n; ++i){
8       sel[i] = 1; fa[i] = i;
9       for(int j = 1; j <= n; ++j) if(fa[j] != i)
10        if(from[j][i] = i, edge[sel[i]][i] > edge[j][i])
11         sel[i] = j;
12   }
13   int limit = n;
14   while(1){
15     int prelimit = limit; memset(vis, 0, sizeof(vis));
16     vis[1] = 1;
17     for(int i = 2; i <= prelimit; ++i) if(fa[i] == i &&
18     !vis[i]){
19       int j = i; while(!vis[j]) vis[j] = i, j =
20       getfa(sel[j]);
21       if(j == 1 || vis[j] != i) continue; vector<int> C;
22       int k = j;
23       do C.push_back(k), k = getfa(sel[k]); while(k != j);
24       ++limit;
25       for(int i = 1; i <= n; ++i){
26         edge[i][limit] = INF, from[i][limit] = limit;
27       }
28       fa[limit] = vis[limit] = limit;
29       for(int i = 0; i < int(C.size()); ++i){
30         int x = C[i], fa[x] = limit;
31         for(int j = 1; j <= n; ++j)
32           if(edge[j][x] != INF && edge[j][limit] >
33             edge[j][x] - edge[sel[x]][x]){
34             edge[j][limit] = edge[j][x] - edge[sel[x]][x];
35             from[j][limit] = x;
36           }
37       }
38       for(int j=1;j<=n;++j) if(getfa(j)==limit)
39         edge[j][limit] = INF;
40       sel[limit] = 1;
41       for(int j = 1; j <= n; ++j)
42         if(edge[sel[limit]][limit] > edge[j][limit])
43           sel[limit] = j;

```

```

36   }
37   if(prelimit == limit) break;
38 }
39 for(int i = limit; i > 1; --i) sel[from[sel[i]][i]] =
   sel[i];
40 }

```

6.7 带花树

```

1  vector<int> link[maxn];
2  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)
11      break; u = pred[match[u]]; }
12    while(true){ v = base[v]; if(InPath[v])
13      break; v = pred[match[v]]; }
14    return v;
15  }
16  void ResetTrace(int u){
17     int v;
18     while(base[u] != newbase){
19       v = match[u];
20       InBlossom[base[u]] = InBlossom[base[v]] = true;
21       u = pred[v];
22       if(base[u] != newbase) pred[u] = v;
23     }
24  }
25  void BlossomContract(int u, int v){
26     newbase = FindCommonAncestor(u, v);
27     for (int i=0; i<n; i++)
28       InBlossom[i] = 0;
29     ResetTrace(u); ResetTrace(v);
30     if(base[u] != newbase) pred[u] = v;
31     if(base[v] != newbase) pred[v] = u;
32     for(int i=0; i<n; i++)
33       if(InBlossom[base[i]]){
34         base[i] = newbase;
35         if(!InQueue[i]) push(i);
36       }
37   }
38   bool FindAugmentingPath(int u){
39     bool found = false;
40     for(int i=0; i<n; i++) pred[i] = -1, base[i] = i;
41     for (int i=0; i<n; i++) InQueue[i] = 0;
42     start = u; finish = -1; head = tail = 0; push(start);
43     while(head < tail){
44       int u = pop();
45       for(int i = link[u].size() - 1; i >= 0; i--){
46         int v = link[u][i];
47         if(base[u] != base[v] && match[u] != v)
48           if(v == start || (match[v] >= 0 && pred[match[v]] >= 0))
49             BlossomContract(u, v);
50           else if(pred[v] == -1){
51             pred[v] = u;
52             if(match[v] >= 0) push(match[v]);
53             else{ finish = v; return true; }
54           }
55       }
56     }
57     return found;
58   }
59   void AugmentPath(){
60     int u = finish, v = w;
61     while(u >= 0){
62       v = pred[u]; w = match[v]; match[v] = u; match[u] = v; u = w; }
63   }

```

```

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)
64         ↳ if(FindAugmentingPath(i)) AugmentPath();

```

6.8 支配树

```

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]]] <
24             ↳ num[sem[mins[u]]]) {
25             mins[u] = mins[v];
26         }
27     }
28     return fs[u];
29 }
30 void merge(int u, int v) { fs[u] = v; }
31 vector<int> buf[N];
32 int buf2[N];
33 void mark(int source) {
34     ord.clear();
35     ++stamp;
36     dfs(source);
37     for (int i = 0; i < (int)ord.size(); ++i) {
38         int u = ord[i];
39         fs[u] = u, mins[u] = -1, buf2[u] = -1;
40     }
41     for (int i = (int)ord.size() - 1; i > 0; --i) {
42         int u = ord[i], p = fa[u];
43         sem[u] = p;
44         for (int j = 0; j < (int)prec[u].size(); ++j) {
45             int v = prec[u][j];
46             if (use[v] != stamp) continue;
47             if (num[v] > num[u]) {
48                 find(v); v = sem[mins[v]];
49             }
50             if (num[v] < num[sem[u]]) {
51                 sem[u] = v;
52             }
53         }
54         buf[sem[u]].push_back(u);
55         mins[u] = u;
56         merge(u, p);
57         while (buf[p].size()) {
58             int v = buf[p].back();
59             buf[p].pop_back();
60             find(v);
61             if (sem[v] == sem[mins[v]]) {
62                 dom[v] = sem[v];
63             } else {
64                 buf2[v] = mins[v];
65             }
66         }

```

```

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 }

```

6.9 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(){
4      int i, j, a, b, c;
5      for(i=0; i<n; i++) for(j=0; j<n; j++) cost[i][j]=0;
6      for(i=0; i<m; i++){
7          scanf("%d %d %d", &a, &b, &c); cost[a][b]+=c;
8          ↳ cost[b][a]+=c;
9      }
10     pop=n; for(i=0; i<n; i++) seq[i]=i;
11 }
12 void Work(){
13     ans=inf; int i, j, k, l, mm, sum, pk;
14     while(pop > 1){
15         for(i=1; i<pop; i++) used[seq[i]]=0; used[seq[0]]=1;
16         for(i=1; i<pop; i++) len[seq[i]]=cost[seq[0]][seq[i]];
17         pk=0; mm=-inf; k=-1;
18         for(i=1; i<pop; i++) if(len[seq[i]] > mm){
19             ↳ mm=len[seq[i]]; k=i; }
20         for(i=1; i<pop; i++){
21             used[seq[l=k]]=1;
22             if(i==pop-2) pk=k;
23             if(i==pop-1) break;
24             mm=-inf;
25             for(j=1; j<pop; j++) if(!used[seq[j]])
26                 if((len[seq[j]]+cost[seq[l]][seq[j]]) > mm)
27                     mm=len[seq[j]], k=j;
28         }
29         sum=0;
30         for(i=0; i<pop; i++) if(i != k)
31             ↳ sum+=cost[seq[k]][seq[i]];
32         ans=min(ans, sum);
33         for(i=0; i<pop; i++)
34             cost[seq[k]][seq[i]]=cost[seq[i]][seq[k]]+=cost[seq[pk]][seq[i]];
35         seq[pk]=seq[--pop];
36     }
37     printf("%d\n", ans);
38 }

```

6.10 最大团搜索

```

1  const int N = 1000 + 7;
2  vector<vector<bool>> > adj;
3  class MaxClique {
4      const vector<vector<bool>> > adj;
5      const int n;
6      vector<int> result, cur_res;
7      vector<vector<int>> > color_set;
8      const double t_limit; // MAGIC
9      int para, level;
10     vector<pair<int, int>> steps;
11 public:
12     class Vertex {
13     public:
14         int i, d;
15         Vertex(int i, int d = 0) : i(i), d(d) {}
16     };
17     void reorder(vector<Vertex> &p) {

```



```

18     for (auto &u : p) {
19         u.d = 0;
20         for (auto v : p) u.d += adj[v.i][u.i];
21     }
22     sort(p.begin(), p.end(), [&](const Vertex &a,
23         ↪ const Vertex &b) { return a.d > b.d; } );
24 }
25 // reuse p[i].d to denote the maximum possible clique
26 ↪ for first i vertices.
27 void init_color(vector<Vertex> &p) {
28     int maxd = p[0].d;
29     for (int i = 0; i < p.size(); i++) p[i].d = min(i,
30         ↪ maxd) + 1;
31 }
32 bool bridge(const vector<int> &s, int x) {
33     for (auto v : s) if (adj[v][x]) return true;
34     return false;
35 }
36 // approximate estimate the p[i].d
37 // Do not care about first mink color class (For better
38 ↪ result, we must get some vertex in some color class
39 ↪ larger than mink )
40 void color_sort(vector<Vertex> &cur) {
41     int totc = 0, ptr = 0, mink =
42         ↪ max((int)result.size() - (int)cur_res.size(),
43         ↪ 0);
44     for (int i = 0; i < cur.size(); i++) {
45         int x = cur[i].i, k = 0;
46         while (k < totc && bridge(color_set[k], x))
47             ↪ k++;
48         if (k == totc) color_set[totc++].clear();
49         color_set[k].push_back(x);
50         if (k < mink) cur[ptr++].i = x;
51     }
52     if (ptr) cur[ptr - 1].d = 0;
53     for (int i = mink; i < totc; i++) {
54         for (auto v : color_set[i]) {
55             cur[ptr++] = Vertex(v, i + 1);
56         }
57     }
58 }
59 void expand(vector<Vertex> &cur) {
60     steps[level].second = steps[level].second -
61         ↪ steps[level].first + steps[level - 1].first;
62     steps[level].first = steps[level - 1].second;
63     while (cur.size()) {
64         if (cur_res.size() + cur.back().d <=
65             ↪ result.size()) return ;
66         int x = cur.back().i;
67         cur_res.push_back(x); cur.pop_back();
68         vector<Vertex> remain;
69         for (auto v : cur) {
70             if (adj[v.i][x]) remain.push_back(v.i);
71         }
72         if (remain.size() == 0) {
73             if (cur_res.size() > result.size()) result
74                 ↪ = cur_res;
75         } else {
76             // Magic ballance.
77             if (1. * steps[level].second / ++para < t_limit)
78                 ↪ reorder(remain);
79             color_sort(remain);
80             steps[level++].second++;
81             expand(remain);
82             level--;
83         }
84         cur_res.pop_back();
85     }
86 }
87 public:
88 MaxClique(const vector<vector<bool> > &adj, int n,
89     ↪ double tt = 0.025) : adj(_adj), n(n), t_limit(tt)
90     ↪ {}

```

```

77     result.clear();
78     cur_res.clear();
79     color_set.resize(n);
80     steps.resize(n + 1);
81     fill(steps.begin(), steps.end(), make_pair(0, 0));
82     level = 1;
83     para = 0;
84 }
85 vector<int> solve() {
86     vector<Vertex> p;
87     for (int i = 0; i < n; i++)
88         ↪ p.push_back(Vertex(i));
89     reorder(p);
90     init_color(p);
91     expand(p);
92     return result;
93 }

```

6.11 弦图判定

```

1 int nT, nStr, last, c[MAXT][26], fail[MAXT], r[MAXN],
2     ↪ l[MAXN], s[MAXN];
3 int allocate(int len) {
4     l[nT] = len;
5     r[nT] = 0;
6     fail[nT] = 0;
7     memset(c[nT], 0, sizeof(c[nT]));
8     return nT++;
9 }
10 void init() {
11     nT = nStr = 0;
12     int newE = allocate(0);
13     int newO = allocate(-1);
14     last = newE;
15     fail[newE] = newO;
16     fail[newO] = newE;
17     s[0] = -1;
18 }
19 void add(int x) {
20     s[++nStr] = x;
21     int now = last;
22     while (s[nStr - l[now] - 1] != s[nStr]) now = fail[now];
23     if (!c[now][x]) {
24         int newnode = allocate(l[now] + 2), &newfail =
25             ↪ fail[newnode];
26         newfail = fail[now];
27         while (s[nStr - l[newfail] - 1] != s[nStr]) newfail =
28             ↪ fail[newfail];
29         newfail = c[newfail][x];
30         c[now][x] = newnode;
31     }
32     last = c[now][x];
33     r[last]++;
34 }
35 void count() {
36     for (int i = nT - 1; i >= 0; i--) {
37         r[fail[i]] += r[i];
38     }
39 }

```

6.12 斯坦纳树

```

1 void SPFA(int *dist)
2 {
3     static int line[maxn + 5];
4     static bool hash[maxn + 5];
5     int f = 0, r = 0;
6
7     for(int i = 1; i <= N; i++)
8         if(dist[i] < inf)
9             {

```



```

10     line[r] = i;
11     hash[i] = true;
12     r = (r + 1) % (N + 1);
13 }
14
15 while(f != r)
16 {
17     int t = line[f];
18     hash[t] = false;
19     f = (f + 1) % (N + 1);
20
21     for(int i = head[t]; i ; i = edge[i].next)
22     {
23         int v = edge[i].v, dt = dist[t] + edge[i].w;
24
25         if(dt < dist[v])
26         {
27             dist[v] = dt;
28
29             if(!hash[v])
30             {
31                 if(dist[v] < dist[line[f]])
32                 {
33                     f = (f + N) % (N + 1);
34                     line[f] = v;
35                 }
36                 else
37                 {
38                     line[r] = v;
39                     r = (r + 1) % (N + 1);
40                 }
41
42                 hash[v] = true;
43             }
44         }
45     }
46 }
47
48 void solve()
49 {
50     for(int i = 1; i <= S; i++)
51     {
52         for(int j = 1; j <= N; j++)
53             for(int k = (i - 1) & i; k ; k = (k - 1) & i)
54                 G[i][j] = std::min(G[i][j], G[k][j] + G[k]
55                                     ↪ ^ i][j]);
56
57         SPFA(G[i]);
58 }

```

6.13 虚树

```

1 bool cmp(const int lhs,const int rhs)
2 {
3     return dfn[lhs] < dfn[rhs];
4 }
5 void build()
6 {
7     std::sort(h + 1, h + 1 + m, cmp);
8
9     int top = 0;
10
11     for (int i = 1; i <= m; i++)
12     {
13         if (!top) father[st[++top] = h[i]] = 0;
14         else
15         {
16             int p = h[i], lca = LCA(h[i],st[top]);
17
18             while(d[st[top]] > d[lca])
19             {
20                 if (d[st[top - 1]] <= d[lca])

```

```

21         father[st[top]] = lca;
22
23         top--;
24     }
25
26     if (st[top] != lca)
27     {
28         t[++tot] = lca;
29         father[lca] = st[top];
30         st[++top] = lca;
31     }
32
33     father[p] = lca;
34     st[++top] = p;
35 }
36 }
37 }

```

6.14 点分治

```

1 template<class TAT>void checkmax(TAT &x,TAT y)
2 {
3     if(x < y) x = y;
4 }
5 template<class TAT>void checkmin(TAT &x,TAT y)
6 {
7     if(y < x) x = y;
8 }
9 void getsize(int u,int fa)
10 {
11     size[u] = 1;
12     smax[u] = 0;
13
14     for(int i = 0; i < G[u].size(); i++)
15     {
16         int v = G[u][i];
17
18         if(v == fa || ban[v]) continue;
19
20         getsize(v, u);
21
22         size[u] += size[v];
23         checkmax(smax[u], size[v]);
24     }
25 }
26 int getroot(int u,int ts,int fa)
27 {
28     checkmax(smax[u], ts - size[u]);
29
30     int res = u;
31
32     for(int i = 0; i < G[u].size(); i++)
33     {
34         int v = G[u][i];
35
36         if(v == fa || ban[v]) continue;
37
38         int w = getroot(v, ts, u);
39
40         if(smax[w] < smax[res]) res = w;
41     }
42
43     return res;
44 }
45 void solve()
46 {
47     static int line[maxn];
48     static std::vector<int> vec;
49     int f = 0, r = 0;
50
51     line[r++] = 1;
52

```

```

53 while(f != r)
54 {
55     int u = line[f++];
56
57     getsize(u, 0);
58     u = getroot(u, size[u], 0);
59
60     ban[u] = true;
61     vec.clear();
62
63     for(int i = 0; i < G[u].size(); i++)
64         if(!ban[G[u][i]]) vec.push_back(G[u][i]);
65
66     /*
67
68     do something you like...
69
70     */
71
72     for(int i = 0; i < vec.size(); i++)
73         line[r++] = vec[i];
74 }
75 }

```

6.15 最小割最大流

```

1 bool BFS()
2 {
3     for(int i = 1; i <= ind; i++) dep[i] = 0;
4
5     dep[S] = 1, line.push(S);
6
7     while(!line.empty())
8     {
9         int now = line.front();
10        line.pop();
11
12        for(int i = head[now], p; i ; i = edge[i].next)
13            if(edge[i].cap && !dep[p = edge[i].v])
14                dep[p] = dep[now] + 1, line.push(p);
15    }
16
17    if(dep[T])
18    {
19        for(int i = 1; i <= ind; i++)
20            cur[i] = head[i];
21        return true;
22    }
23    else
24        return false;
25 }
26 int DFS(int a,int flow)
27 {
28     if(a == T) return flow;
29
30     int ret = 0;
31
32     for(int &i = cur[a], p; i ; i = edge[i].next)
33         if(dep[p = edge[i].v] == dep[a] + 1 &&
34             ↪ edge[i].cap)
35         {
36             int ff = DFS(p, std::min(flow, edge[i].cap));
37
38             flow -= ff, edge[i].cap -= ff;
39             ret += ff, edge[i ^ 1].cap += ff;
40
41             if(!flow) break;
42         }
43
44     return ret;
45 }
46 int solve()
47 {

```

```

47     int totflow = 0;
48
49     while(BFS())
50     {
51         totflow += DFS(S, INF);
52     }
53
54     return totflow;
55 }

```

6.16 最小费用流

```

1 bool SPFA()
2 {
3     static int line[maxv];
4     static bool hash[maxv];
5     register int f = 0, r = 0;
6
7     for(int i = 1; i <= ind; i++)
8     {
9         dist[i] = inf;
10        from[i] = 0;
11    }
12
13    dist[S] = 0, line[r] = S, r = (r + 1) % maxv;
14    hash[S] = true;
15
16    while(f != r)
17    {
18        int x = line[f];
19
20        line[f] = 0, f = (f + 1) % maxv;
21        hash[x] = false;
22
23        for(int i = head[x]; i; i = edge[i].next)
24            if(edge[i].cap)
25            {
26                int v = edge[i].v;
27                int w = dist[x] + edge[i].cost;
28
29                if(w < dist[v])
30                {
31                    dist[v] = w;
32                    from[v] = i;
33
34                    if(!hash[v])
35                    {
36                        if(f != r && dist[v] <=
37                            ↪ dist[line[f]])
38                            f = (f - 1 + maxv) % maxv,
39                            ↪ line[f] = v;
39                        else
40                            line[r] = v, r = (r + 1) %
41                                ↪ maxv;
42
43                        hash[v] = true;
44                    }
45                }
46            }
47
48        return from[T];
49    }
50
51    int back(int x,int flow)
52    {
53        if(from[x])
54        {
55            flow = back(edge[from[x] ^ 1].v, std::min(flow,
56                ↪ edge[from[x]].cap));
57
58            edge[from[x]].cap -= flow;

```

```

57     edge[from[x] ^ 1].cap += flow;
58 }
59
60 return flow;
61 }
62 int solve()
63 {
64     int mincost = 0, maxflow = 0;
65
66     while(SPFA())
67     {
68         int flow = back(T, inf);
69
70         mincost += dist[T] * flow;
71         maxflow += flow;
72     }
73
74     return mincost;
75 }

```

6.17 zkw 费用流

```

1  int S, T, totFlow, totCost;
2
3  int dis[N], slack[N], visit[N];
4
5  int modlable () {
6      int delta = INF;
7      for (int i = 1; i <= T; i++) {
8          if (!visit[i] && slack[i] < delta) delta =
9              ↪ slack[i];
10         slack[i] = INF;
11     }
12     if (delta == INF) return 1;
13     for (int i = 1; i <= T; i++)
14         if (visit[i]) dis[i] += delta;
15     return 0;
16 }
17 int dfs (int x, int flow) {
18     if (x == T) {
19         totFlow += flow;
20         totCost += flow * (dis[S] - dis[T]);
21         return flow;
22     }
23     visit[x] = 1;
24     int left = flow;
25     for (int i = e.last[x]; ~i; i = e.succ[i])
26         if (e.cap[i] > 0 && !visit[e.other[i]]) {
27             int y = e.other[i];
28             if (dis[y] + e.cost[i] == dis[x]) {
29                 int delta = dfs (y, min (left, e.cap[i]));
30                 e.cap[i] -= delta;
31                 e.cap[i ^ 1] += delta;
32                 left -= delta;
33                 if (!left) { visit[x] = 0; return flow; }
34             } else {
35                 slack[y] = min (slack[y], dis[y] +
36                     ↪ e.cost[i] - dis[x]);
37             }
38         }
39     return flow - left;
40 }
41 pair <int, int> minCost () {
42     totFlow = 0; totCost = 0;
43     fill (dis + 1, dis + T + 1, 0);
44     do {
45         do {
46             fill (visit + 1, visit + T + 1, 0);
47         } while (dfs (S, INF));
48     } while (!modlable ());
49     return make_pair (totFlow, totCost);

```

```

50 }

```

6.18 最小割树

```

1  #include<iostream>
2  #include<cstdio>
3  #include<cstdlib>
4  #include<cstring>
5  #include<algorithm>
6  #include<queue>
7  #define inf 0x3f3f3f3f
8  #define N 155
9  using namespace std;
10
11  int
12      ↪ cnt,n,m,dis[N],last[N],a[N],tmp[N],ans[N][N],s,t,mark[N];
13  struct edge{int to,c,next;}e[N*200];
14  queue <int> q;
15  void addedge(int u,int v,int c)
16  {
17
18      ↪ e[++cnt].to=v;e[cnt].c=c;e[cnt].next=last[u];last[u]=cnt;
19
20      ↪ e[++cnt].to=u;e[cnt].c=c;e[cnt].next=last[v];last[v]=cnt;
21  }
22  bool bfs()
23  {
24     memset(dis,0,sizeof(dis));
25     dis[s]=2;
26     while (!q.empty()) q.pop();
27     q.push(s);
28     while (!q.empty())
29     {
30         int u=q.front();
31         q.pop();
32         for (int i=last[u];i;i=e[i].next)
33             if (e[i].c&&!dis[e[i].to])
34             {
35                 dis[e[i].to]=dis[u]+1;
36                 if (e[i].to==t) return 1;
37                 q.push(e[i].to);
38             }
39     }
40     return 0;
41 }
42 int dfs(int x,int maxf)
43 {
44     if (x==t||!maxf) return maxf;
45     int ret=0;
46     for (int i=last[x];i;i=e[i].next)
47         if (e[i].c&&dis[e[i].to]==dis[x]+1)
48         {
49             int f=dfs(e[i].to,min(e[i].c,maxf-ret));
50             e[i].c-=f;
51             e[i^1].c+=f;
52             ret+=f;
53             if (ret==maxf) break;
54         }
55     if (!ret) dis[x]=0;
56     return ret;
57 }
58
59 void dfs(int x)
60 {
61     mark[x]=1;
62     for (int i=last[x];i;i=e[i].next)
63         if (e[i].c&&!mark[e[i].to]) dfs(e[i].to);
64 }
65

```

```

66 void solve(int l,int r)
67 {
68     if (l==r) return;
69     s=a[l];t=a[r];
70     for (int i=2;i<=cnt;i+=2)
71         e[i].c=e[i^1].c=(e[i].c+e[i^1].c)/2;
72     int flow=0;
73     while (bfs()) flow+=dfs(s,inf);
74     memset(mark,0,sizeof(mark));
75     dfs(s);
76     for (int i=1;i<=n;i++)
77         if (mark[i])
78             for (int j=1;j<=n;j++)
79                 if (!mark[j])
80                     ↪ ans[i][j]=ans[j][i]=min(ans[i][j],flow);
81     int i=l,j=r;
82     for (int k=l;k<=r;k++)
83         if (mark[a[k]]) tmp[i++]=a[k];
84         else tmp[j--]=a[k];
85     for (int k=l;k<=r;k++)
86         a[k]=tmp[k];
87     solve(l,i-1);
88     solve(j+1,r);
89 }
90
91 int main()
92 {
93     int cas;
94     scanf("%d",&cas);
95     while (cas--)
96     {
97         scanf("%d%d",&n,&m);
98         cnt=1;
99         for (int i=1;i<=n;i++)
100             a[i]=i;
101         memset(last,0,sizeof(last));
102         memset(ans,inf,sizeof(ans));
103         for (int i=1;i<=m;i++)
104         {
105             int x,y,z;
106             scanf("%d%d%d",&x,&y,&z);
107             addedge(x,y,z);
108         }
109         solve(1,n);
110         int q;
111         scanf("%d",&q);
112         for (int i=1;i<=q;i++)
113         {
114             int x,tot=0;
115             scanf("%d",&x);
116             for (int i=1;i<=n;i++)
117                 for (int j=i+1;j<=n;j++)
118                     if (ans[i][j]<=x) tot++;
119             printf("%d\n",tot);
120         }
121         cout<<endl;
122     }
123     return 0;
124 }

```

6.19 上下界网络流建图

$B(u,v)$ 表示边 (u,v) 流量的下界, $C(u,v)$ 表示边 (u,v) 流量的上界, $F(u,v)$ 表示边 (u,v) 的流量。设 $G(u,v) = F(u,v) - B(u,v)$, 显然有

$$0 \leq G(u,v) \leq C(u,v) - B(u,v)$$

6.19.1 无源汇的上下界可行流

建立超级源点 S^* 和超级汇点 T^* , 对于原图每条边 (u,v) 在新网络中连如下三条边: $S^* \rightarrow v$, 容量为 $B(u,v)$;

$u \rightarrow T^*$, 容量为 $B(u,v)$; $u \rightarrow v$, 容量为 $C(u,v) - B(u,v)$ 。最后求新网络的最大流, 判断从超级源点 S^* 出发的边是否都满流即可, 边 (u,v) 的最终解中的实际流量为 $G(u,v) + B(u,v)$ 。

6.19.2 有源汇的上下界可行流

从汇点 T 到源点 S 连一条上界为 ∞ , 下界为 0 的边。按照无源汇的上下界可行流一样做即可, 流量即为 $T \rightarrow S$ 边上的流量。

6.19.3 有源汇的上下界最大流

1. 在有源汇的上下界可行流中, 从汇点 T 到源点 S 的边改为连一条上界为 ∞ , 下届为 x 的边。 x 满足二分性质, 找到最大的 x 使得新网络存在无源汇的上下界可行流即为原图的最大流。
2. 从汇点 T 到源点 S 连一条上界为 ∞ , 下界为 0 的边, 变成无源汇的网络。按照无源汇的上下界可行流的方法, 建立超级源点 S^* 和超级汇点 T^* , 求一遍 $S^* \rightarrow T^*$ 的最大流, 再将汇点 T 到源点 S 的这条边拆掉, 求一次 $S \rightarrow T$ 的最大流即可。

6.19.4 有源汇的上下界最小流

1. 在有源汇的上下界可行流中, 从汇点 T 到源点 S 的边改为连一条上界为 x , 下界为 0 的边。 x 满足二分性质, 找到最小的 x 使得新网络存在无源汇的上下界可行流即为原图的最小流。
2. 按照无源汇的上下界可行流的方法, 建立超级源点 S^* 与超级汇点 T^* , 求一遍 $S^* \rightarrow T^*$ 的最大流, 但是注意这一次不加上汇点 T 到源点 S 的这条边, 即不使之改为无源汇的网络去求解。求完后, 再加上那条汇点 T 到源点 S 上界 ∞ 的边。因为这条边下界为 0, 所以 S^*, T^* 无影响, 再直接求一次 $S^* \rightarrow T^*$ 的最大流。若超级源点 S^* 出发的边全部满流, 则 $T \rightarrow S$ 边上的流量即为原图的最小流, 否则无解。

7. 其他

7.1 Dancing Links

7.1.1 精确覆盖

```

1 #pragma comment(linker, "/STACK:1024000000,1024000000")
2 #include<iostream>
3 #include<cstdio>
4 #include<cstring>
5 #include<algorithm>
6 #include<map>
7 #include<queue>
8 #include<set>
9 #include<cmath>
10 #include<bitset>
11 #define mem(a,b) memset(a,b,sizeof(a))
12 #define lson i<<1,l,mid
13 #define rson i<<1|1,mid+1,r
14 #define llson j<<1,l,mid
15 #define rrson j<<1|1,mid+1,r
16 #define INF 0x7fffffff
17 #define maxn 1000005
18 typedef long long ll;
19 typedef unsigned long long ull;
20 using namespace std;
21 int head,sz;
22 int U[maxn],D[maxn],L[maxn],R[maxn]; //上下左右链表指针
23 ↪
24 int H[maxn],ROW[maxn],C[maxn],S[maxn],O[maxn];
25 void remove(int c)
26 {
27     L[R[c]]=L[c];
28     R[L[c]]=R[c];
29     for(int i=D[c]; i!=c; i=D[i])

```

```

29     for(int j=R[i]; j!=i; j=R[j])
30     {
31         U[D[j]]=U[j];
32         D[U[j]]=D[j];
33         --S[C[j]];
34     }
35 }
36 void resume(int c)
37 {
38     for(int i=U[c]; i!=c; i=U[i])
39     {
40         for(int j=L[i]; j!=i; j=L[j])
41         {
42             ++S[C[j]];
43             U[D[j]]=j;
44             D[U[j]]=j;
45         }
46     }
47     L[R[c]]=c;
48     R[L[c]]=c;
49 }
50 void init(int m)//m 是列
51 {
52     head=0;//头指针为 0
53     for(int i=0; i<=m; i++)
54     {
55         U[i]=i;
56         D[i]=i;//建立双向十字链表
57         L[i]=i-1;
58         R[i]=i+1;
59         S[i]=0;
60     }
61     R[m]=0;
62     L[0]=m;
63     S[0]=INF+1;
64     sz=m+1;
65     memset(H,0,sizeof(H));
66 }
67 void insert(int i, int j)
68 {
69     if(H[i])
70     {
71         L[sz] = L[H[i]];
72         R[sz] = H[i];
73         L[R[sz]] = sz;
74         R[L[sz]] = sz;
75     }
76     else
77     {
78         L[sz] = sz;
79         R[sz] = sz;
80         H[i] = sz;
81     }
82     U[sz] = U[j];
83     D[sz] = j;
84     U[D[sz]] = sz;
85     D[U[sz]] = sz;
86     C[sz] = j;
87     ROW[sz] = i;
88     ++S[j];
89     ++sz;
90 }
91 bool dfs(int k,int len)
92 {
93     if(R[head]==head)
94     {
95         sort(0,0+len*len);
96         int p=0;
97         for(int i=0; i<len; i++)
98         {
99             for(int j=0; j<len; j++)
100             {
101                 int num=0[p++];

```

```

102                 num=num-(i*len+j)*len;
103                 printf("%d",num);
104             }
105             puts("");
106         }
107         return true;
108     }
109     int s=INF,c;
110     for (int t=R[head]; t!=head; t=R[t])
111         if (S[t]<s) s=S[t],c=t;
112     remove(c);
113     for(int i=D[c]; i!=c; i=D[i])
114     {
115         O[k]=ROW[i];
116         for(int j=R[i]; j!=i; j=R[j])
117             remove(C[j]);
118         if(dfs(k+1,len))
119             return true;
120         for(int j=L[i]; j!=i; j=L[j])
121             resume(C[j]);
122     }
123     resume(c);
124     return false;
125 }
126 void calc(int i,int j,int k,int len)
127 {
128     int r=(i*len+j-1)*len+k;
129     int base=sqrt(len);
130     //第 i 行有数字 k
131     insert(r,i*len+k);
132     //第 j 列有数字 k
133     insert(r,len*len+(j-1)*len+k);
134     //第 k 块有数字 k
135     int block=(j-1)/base*base+i/base;
136     insert(r,len*len*2+block*len+k);
137     //第 i 行 j 列有一个数字 (限制一个出格子只填一个数)
138     insert(r,len*len*3+i*len+j);
139 }
140 void build(char s[][10],int len)//len 表示是几宫数独
141 {
142     int i,j,k;
143     init(len*len*4);
144     for(i=0; i<len; i++)
145         for(j=1; j<=len; j++)
146         {
147             if(s[i][j-1]=='0')
148                 for(k=1; k<=len; k++)
149                     calc(i,j,k,len);
150             else calc(i,j,s[i][j-1]-'0',len);
151         }
152 }
153 int main()
154 {
155     //freopen("1.txt","r",stdin);
156     int t;
157     cin>>t;
158     while(t--)
159     {
160         char s[10][10];
161         for(int i=0; i<9; i++)
162             scanf("%s",s[i]);
163         build(s,9);
164         dfs(0,9);//从根开始搜
165     }
166     return 0;
167 }

```

7.1.2 重复覆盖

1 Problem : 2295 (Radar) Judge Status : Accepted
2 RunId : 4355553 Language : G++ Author : zhuyawei

```

3 Code Render Status : Rendered By HDOJ G++ Code Render
  ↳ Version 0.01 Beta
4 # include<stdio.h>
5 # include<math.h>
6 # include<string.h>
7 # define eps 1e-8
8 # define N 55
9 # define V 3600
10 int n,m,K;
11 int L[V],R[V];
12 int D[V],U[V];
13 int C[V];
14 int S[N],H[N];
15 int ak,size;
16 double dis(double x1,double y1,double x2,double y2)
17 {
18     return sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1));
19 }
20 void Link(int r,int c)
21 {
22     S[c]++;C[size]=c;
23     U[size]=U[c];D[U[c]]=size;
24     D[size]=c;U[c]=size;
25     if(H[r]==-1) H[r]=L[size]=R[size]=size;
26     else
27     {
28         L[size]=L[H[r]];R[L[H[r]]]=size;
29         R[size]=H[r];L[H[r]]=size;
30     }
31     size++;
32 }
33 void remove(int c)
34 {
35     int i;
36     for(i=D[c];i!=c;i=D[i])
37         L[R[i]]=L[i],R[L[i]]=R[i];
38 }
39 void resume(int c)
40 {
41     int i;
42     for(i=U[c];i!=c;i=U[i])
43         L[R[i]]=R[L[i]]=i;
44 }
45 int h()
46 {
47     int i,j,k,count=0;
48     bool visit[N];
49     memset(visit,0,sizeof(visit));
50     for(i=R[0];i;i=R[i])
51     {
52         if(visit[i]) continue;
53         count++;
54         visit[i]=1;
55         for(j=D[i];j!=i;j=D[j])
56         {
57             for(k=R[j];k!=j;k=R[k])
58                 visit[C[k]]=1;
59         }
60     }
61     return count;
62 }
63 void Dance(int k)
64 {
65     int i,j,c,Min,ans;
66     ans=h();
67     if(k+ans>K || k+ans==ak) return;
68     if(!R[0])
69     {
70         if(k<ak) ak=k;
71         return;
72     }
73     for(Min=N,i=R[0];i;i=R[i])
74         if(S[i]<Min) Min=S[i],c=i;

```

```

75     for(i=D[c];i!=c;i=D[i])
76     {
77         remove(i);
78         for(j=R[i];j!=i;j=R[j])
79             remove(j);
80         Dance(k+1);
81         for(j=L[i];j!=i;j=L[j])
82             resume(j);
83         resume(i);
84     }
85     return;
86 }
87 int main()
88 {
89     int i,j,ncase;
90     double x[N],y[N],x1[N],y1[N];
91     double left,right,ans,mid;
92     scanf("%d",&ncase);
93     while(ncase--)
94     {
95         scanf("%d%d%d",&n,&m,&K);
96         for(i=1;i<=n;i++)
97             scanf("%lf%lf",&x[i],&y[i]);
98         for(i=1;i<=m;i++)
99             scanf("%lf%lf",&x1[i],&y1[i]);
100         left=0;
101         right=1416.0;
102         ans=right;
103         while(right>=left)
104         {
105             for(i=0;i<=n;i++)
106             {
107                 S[i]=0;
108                 U[i]=D[i]=i;
109                 L[i+1]=i;R[i]=i+1;
110             }R[n]=0;
111             memset(H,-1,sizeof(H));
112             size=n+1;
113             mid=(left+right)/2;
114             for(i=1;i<=m;i++)
115             {
116                 for(j=1;j<=n;j++)
117                     if(mid>=dis(x1[i],y1[i],x[j],y[j]))
118                         ↳ Link(i,j);
119             }
120             ak=N;
121             Dance(0);
122             if(ak<=K) {ans=mid<ans?mid:ans;right=mid-eps;}
123             else left=mid+eps;
124         }
125         printf("%.6lf\n",ans);
126     }
127     return 0;
128 }

```

7.2 蔡勒公式

```

1 int zeller(int y,int m,int d) {
2     if (m<=2) y--,m+=12; int c=y/100; y%=100;
3     int w=((c>>2)-(c<<1)+y+(y>>2)+(13*(m+1)/5)+d-1)%7;
4     if (w<0) w+=7; return(w);
5 }

```

7.3 五边形数定理

$$p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k-1} p\left(n - \frac{k(3k-1)}{2}\right)$$

```

1 #include<iostream>
2 #include<cstdio>
3 using namespace std;
4 #define LL __int64
5 const int N=100005;

```

```

6 const int MOD=1000000007;
7 LL dp[N],fi[N];
8 LL five(LL x){ return (3*x*x-x)/2; }
9 //五边形数
10 void wbxs(){
11     dp[0]=1;
12     int t=1000; //其实可以等于 sqrt(N)
13     for(int i=-t;i<=t;++i)
14         fi[i+t]=five(i); //Q
15     for(int i=1;i<=100000;++i){
16         int flag=1;
17         for(int j=1;;++j){
18             LL a=fi[j+t],b=fi[-j+t];
19             if(a>i && b>i) break;
20             if(a<=i) dp[i]=(dp[i]+dp[i-a]*flag+MOD)%MOD;
21             //p
22             if(b<=i) dp[i]=(dp[i]+dp[i-b]*flag+MOD)%MOD;
23             flag*=-1;
24         }
25     }
26 int main(){
27     wbxs();
28     int T,n;
29     scanf("%d",&T);
30     while(T--){
31         scanf("%d",&n);
32         printf("%I64d\n",dp[n]);
33     }
34     return 0;
35 }

```

7.4 凸包闵可夫斯基和

```

1 // cv[0..1] 为两个顺时针凸包，其中起点等于终点，求
  // 出的闵可夫斯基和不一定是严格凸包
2 int i[2] = {0, 0}, len[2] = {(int)cv[0].size() - 1,
  // (int)cv[1].size() - 1};
3 vector<P> mnk;
4 mnk.push_back(cv[0][0] + cv[1][0]);
5 do {
6     int d((cv[0][i[0] + 1] - cv[0][i[0]]) * (cv[1][i[1] + 1]
  // - cv[1][i[1]]) >= 0);
7     mnk.push_back(cv[d][i[d] + 1] - cv[d][i[d]] +
  // mnk.back());
8     i[d] = (i[d] + 1) % len[d];
9 } while(i[0] || i[1]);

```

7.5 树上路径求交

```

1 bool cmp(int x,int y)
2 {
3     return dep[x] > dep[y];
4 }
5 std::pair<int,int> path_intersection(int a,int b,int c,int
  // d)
6 {
7     std::vector<int> poi;
8
9     poi.push_back(getlca(a, c));
10    poi.push_back(getlca(a, d));
11    poi.push_back(getlca(b, c));
12    poi.push_back(getlca(b, d));
13
14    std::sort(poi.begin(), poi.end(), cmp);
15
16    return std::make_pair(poi[0], poi[1]);
17 }

```

8. 技巧

8.1 STL 归还空间

```

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 }

```

8.2 大整数取模

```

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 }

```

8.3 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 }

```

8.4 二次随机法

```

1 #include <random>
2
3 int main() {
4     std::mt19937 g(seed); // std::mt19937_64
5     std::cout << g() << std::endl;
6 }

```

8.5 vimrc

```

1 set ruler
2 set number
3 set smartindent
4 set autoindent
5 set tabstop=4
6 set softtabstop=4
7 set shiftwidth=4
8 set hlsearch

```



```

9  set incsearch
10 set autoread
11 set backspace=2
12 set mouse=a
13
14 syntax on
15
16 nmap <C-A> ggVG
17 vmap <C-C> "+y
18
19 filetype plugin indent on
20
21 autocmd FileType cpp set cindent
22 autocmd FileType cpp map <F9> :!g++ % -o %< -g -std=c++11
    ↪ -Wall -Wextra -Wconversion && size %< <CR>
23 autocmd FileType cpp map <C-F9> :!g++ % -o %< -std=c++11
    ↪ -O2 && size %< <CR>
24 autocmd FileType cpp map <F8> :!time ./%< < %<.in <CR>
25 autocmd FileType cpp map <F5> :!time ./%< <CR>
26
27 map <F3> :vnew %<.in <CR>
28 map <F4> :!gedit % <CR>

```

8.6 控制 cout 输出实数精度

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

8.7 让 make 支持 c++11

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

8.8 tuple 相关

```

1 mytuple = std::make_tuple(10, 2.6, 'a');           //
    ↪ packing values into tuple
2 std::tie(myint, std::ignore, mychar) = mytuple;    //
    ↪ unpacking tuple into variables
3 std::get<I>(mytuple) = 20;
4 std::cout << std::get<I>(mytuple) << std::endl;    // get
    ↪ the Ith(const) element

```

8.9 汇编技巧

```

1 03优化
2 #define __ __attribute__((optimize("-O3")))
3 #define _ __inline__ __attribute__((__gnu_inline__,
    ↪ __always_inline__, __artificial__))
4
5 汇编开栈
6 #pragma comment(linker, "/STACK:256000000")
7
8 int __size = 256 << 20;
9 char* __p__ = (char *) malloc(__size__) + __size__;
10
11 int main() {
12     __asm__("movl %0, %%esp\n" :: "r"(__p__));
13     return 0;
14 }

```

9. 提示

9.1 线性规划转对偶

$$\begin{aligned} &\text{maximize } \mathbf{c}^T \mathbf{x} \\ &\text{subject to } \mathbf{A} \mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0 \end{aligned} \iff \begin{aligned} &\text{minimize } \mathbf{y}^T \mathbf{b} \\ &\text{subject to } \mathbf{y}^T \mathbf{A} \geq \mathbf{c}^T, \mathbf{y} \geq 0 \end{aligned}$$

9.2 NTT 素数及其原根

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

9.3 积分表

9.3.1 $ax^2 + bx + c (a > 0)$

$$1. \int \frac{dx}{ax^2+bx+c} = \begin{cases} \frac{2}{\sqrt{4ac-b^2}} \arctan \frac{2ax+b}{\sqrt{4ac-b^2}} + C & (b^2 < 4ac) \\ \frac{1}{\sqrt{b^2-4ac}} \ln \left| \frac{2ax+b-\sqrt{b^2-4ac}}{2ax+b+\sqrt{b^2-4ac}} \right| + C & (b^2 > 4ac) \end{cases}$$

$$2. \int \frac{x}{ax^2+bx+c} dx = \frac{1}{2a} \ln |ax^2+bx+c| - \frac{b}{2a} \int \frac{dx}{ax^2+bx+c}$$

9.3.2 $\sqrt{\pm ax^2 + bx + c} (a > 0)$

$$1. \int \frac{dx}{\sqrt{ax^2+bx+c}} = \frac{1}{\sqrt{a}} \ln |2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C$$

$$2. \int \sqrt{ax^2+bx+c} dx = \frac{2ax+b}{4a} \sqrt{ax^2+bx+c} + \frac{4ac-b^2}{8\sqrt{a^3}} \ln |2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C$$

$$3. \int \frac{x}{\sqrt{ax^2+bx+c}} dx = \frac{1}{a} \sqrt{ax^2+bx+c} - \frac{b}{2\sqrt{a^3}} \ln |2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}| + C$$

$$4. \int \frac{dx}{\sqrt{c+bx-ax^2}} = -\frac{1}{\sqrt{a}} \arcsin \frac{2ax-b}{\sqrt{b^2+4ac}} + C$$

$$5. \int \sqrt{c+bx-ax^2} dx = \frac{2ax-b}{4a} \sqrt{c+bx-ax^2} + \frac{b^2+4ac}{8\sqrt{a^3}} \arcsin \frac{2ax-b}{\sqrt{b^2+4ac}} + C$$

$$6. \int \frac{x}{\sqrt{c+bx-ax^2}} dx = -\frac{1}{a} \sqrt{c+bx-ax^2} + \frac{b}{2\sqrt{a^3}} \arcsin \frac{2ax-b}{\sqrt{b^2+4ac}} + C$$

9.3.3 $\sqrt{\pm \frac{x-a}{x-b}}$ 或 $\sqrt{(x-a)(x-b)}$

$$1. \int \frac{dx}{\sqrt{(x-a)(b-x)}} = 2 \arcsin \sqrt{\frac{x-a}{b-x}} + C \quad (a < b)$$

$$2.$$

$$\int \sqrt{(x-a)(b-x)} dx = \frac{2x-a-b}{4} \sqrt{(x-a)(b-x)} + \frac{(b-a)^2}{4} \arcsin \sqrt{\frac{x-a}{b-x}} + C, (a < b) \quad (1)$$

9.3.4 三角函数的积分

$$1. \int \tan x dx = -\ln |\cos x| + C$$

$$2. \int \cot x dx = \ln |\sin x| + C$$

$$3. \int \sec x dx = \ln \left| \tan \left(\frac{\pi}{4} + \frac{x}{2} \right) \right| + C = \ln |\sec x + \tan x| + C$$

$$4. \int \csc x dx = \ln \left| \tan \frac{x}{2} \right| + C = \ln |\csc x - \cot x| + C$$

$$5. \int \sec^2 x dx = \tan x + C$$

$$6. \int \csc^2 x dx = -\cot x + C$$

$$7. \int \sec x \tan x dx = \sec x + C$$

$$8. \int \csc x \cot x dx = -\csc x + C$$

$$9. \int \sin^2 x dx = \frac{x}{2} - \frac{1}{4} \sin 2x + C$$

$$10. \int \cos^2 x dx = \frac{x}{2} + \frac{1}{4} \sin 2x + C$$

$$11. \int \sin^n x dx = -\frac{1}{n} \sin^{n-1} x \cos x + \frac{n-1}{n} \int \sin^{n-2} x dx$$

$$12. \int \cos^n x dx = \frac{1}{n} \cos^{n-1} x \sin x + \frac{n-1}{n} \int \cos^{n-2} x dx$$

$$13. \int \frac{dx}{\sin^n x} = -\frac{1}{n-1} \frac{\cos x}{\sin^{n-1} x} + \frac{n-2}{n-1} \int \frac{dx}{\sin^{n-2} x}$$

$$14. \int \frac{dx}{\cos^n x} = \frac{1}{n-1} \frac{\sin x}{\cos^{n-1} x} + \frac{n-2}{n-1} \int \frac{dx}{\cos^{n-2} x}$$

15.

$$\begin{aligned} & \int \cos^m x \sin^n x dx \\ &= \frac{1}{m+n} \cos^{m-1} x \sin^{n+1} x + \frac{m-1}{m+n} \int \cos^{m-2} x \sin^n x dx \\ &= -\frac{1}{m+n} \cos^{m+1} x \sin^{n-1} x + \frac{n-1}{m+1} \int \cos^m x \sin^{n-2} x dx \end{aligned}$$

$$16. \int \frac{dx}{a+b \sin x} = \begin{cases} \frac{2}{\sqrt{a^2-b^2}} \arctan \frac{a \tan \frac{x}{2} + b}{\sqrt{a^2-b^2}} + C & (a^2 > b^2) \\ \frac{1}{\sqrt{b^2-a^2}} \ln \left| \frac{a \tan \frac{x}{2} + b - \sqrt{b^2-a^2}}{a \tan \frac{x}{2} + b + \sqrt{b^2-a^2}} \right| + C & (a^2 < b^2) \end{cases}$$

$$17. \int \frac{dx}{a+b \cos x} = \begin{cases} \frac{2}{a+b} \sqrt{\frac{a+b}{a-b}} \arctan \left(\sqrt{\frac{a-b}{a+b}} \tan \frac{x}{2} \right) + C & (a^2 > b^2) \\ \frac{1}{a+b} \sqrt{\frac{a+b}{a-b}} \ln \left| \frac{\tan \frac{x}{2} + \sqrt{\frac{a+b}{b-a}}}{\tan \frac{x}{2} - \sqrt{\frac{a+b}{b-a}}} \right| + C & (a^2 < b^2) \end{cases}$$

$$18. \int \frac{dx}{a^2 \cos^2 x + b^2 \sin^2 x} = \frac{1}{ab} \arctan \left(\frac{b}{a} \tan x \right) + C$$

$$19. \int \frac{dx}{a^2 \cos^2 x - b^2 \sin^2 x} = \frac{1}{2ab} \ln \left| \frac{b \tan x + a}{b \tan x - a} \right| + C$$

$$20. \int x \sin ax dx = \frac{1}{a^2} \sin ax - \frac{1}{a} x \cos ax + C$$

$$21. \int x^2 \sin ax dx = -\frac{1}{a} x^2 \cos ax + \frac{2}{a^2} x \sin ax + \frac{2}{a^3} \cos ax + C$$

$$22. \int x \cos ax dx = \frac{1}{a^2} \cos ax + \frac{1}{a} x \sin ax + C$$

$$23. \int x^2 \cos ax dx = \frac{1}{a} x^2 \sin ax + \frac{2}{a^2} x \cos ax - \frac{2}{a^3} \sin ax + C$$

9.3.5 反三角函数的积分 (其中 $a > 0$)

$$1. \int \arcsin \frac{x}{a} dx = x \arcsin \frac{x}{a} + \sqrt{a^2 - x^2} + C$$

$$2. \int x \arcsin \frac{x}{a} dx = \left(\frac{x^2}{2} - \frac{a^2}{4} \right) \arcsin \frac{x}{a} + \frac{x}{4} \sqrt{x^2 - x^2} + C$$

$$3. \int x^2 \arcsin \frac{x}{a} dx = \frac{x^3}{3} \arcsin \frac{x}{a} + \frac{1}{9} (x^2 + 2a^2) \sqrt{a^2 - x^2} + C$$

$$4. \int \arccos \frac{x}{a} dx = x \arccos \frac{x}{a} - \sqrt{a^2 - x^2} + C$$

$$5. \int x \arccos \frac{x}{a} dx = \left(\frac{x^2}{2} - \frac{a^2}{4} \right) \arccos \frac{x}{a} - \frac{x}{4} \sqrt{a^2 - x^2} + C$$

$$6. \int x^2 \arccos \frac{x}{a} dx = \frac{x^3}{3} \arccos \frac{x}{a} - \frac{1}{9} (x^2 + 2a^2) \sqrt{a^2 - x^2} + C$$

$$7. \int \arctan \frac{x}{a} dx = x \arctan \frac{x}{a} - \frac{a}{2} \ln(a^2 + x^2) + C$$

$$8. \int x \arctan \frac{x}{a} dx = \frac{1}{2} (a^2 + x^2) \arctan \frac{x}{a} - \frac{a}{2} x + C$$

$$9. \int x^2 \arctan \frac{x}{a} dx = \frac{x^3}{3} \arctan \frac{x}{a} - \frac{a}{6} x^2 + \frac{a^3}{6} \ln(a^2 + x^2) + C$$

9.3.6 指数函数的积分

$$1. \int a^x dx = \frac{1}{\ln a} a^x + C$$

$$2. \int e^{ax} dx = \frac{1}{a} e^{ax} + C$$

$$3. \int x e^{ax} dx = \frac{1}{a^2} (ax - 1) e^{ax} + C$$

$$4. \int x^n e^{ax} dx = \frac{1}{a} x^n e^{ax} - \frac{n}{a} \int x^{n-1} e^{ax} dx$$

$$5. \int x a^x dx = \frac{x}{\ln a} a^x - \frac{1}{(\ln a)^2} a^x + C$$

$$6. \int x^n a^x dx = \frac{1}{\ln a} x^n a^x - \frac{n}{\ln a} \int x^{n-1} a^x dx$$

$$7. \int e^{ax} \sin bxdx = \frac{1}{a^2 + b^2} e^{ax} (a \sin bx - b \cos bx) + C$$

$$8. \int e^{ax} \cos bxdx = \frac{1}{a^2 + b^2} e^{ax} (b \sin bx + a \cos bx) + C$$

$$9. \int e^{ax} \sin^n bxdx = \frac{1}{a^2 + b^2 n^2} e^{ax} \sin^{n-1} bx (a \sin bx - nb \cos bx) + \frac{n(n-1)b^2}{a^2 + b^2 n^2} \int e^{ax} \sin^{n-2} bxdx$$

$$10. \int e^{ax} \cos^n bxdx = \frac{1}{a^2 + b^2 n^2} e^{ax} \cos^{n-1} bx (a \cos bx + nb \sin bx) + \frac{n(n-1)b^2}{a^2 + b^2 n^2} \int e^{ax} \cos^{n-2} bxdx$$

9.3.7 对数函数的积分

$$1. \int \ln x dx = x \ln x - x + C$$

$$2. \int \frac{dx}{x \ln x} = \ln |\ln x| + C$$

$$3. \int x^n \ln x dx = \frac{1}{n+1} x^{n+1} \left(\ln x - \frac{1}{n+1} \right) + C$$

$$4. \int (\ln x)^n dx = x (\ln x)^n - n \int (\ln x)^{n-1} dx$$

$$5. \int x^m (\ln x)^n dx = \frac{1}{m+1} x^{m+1} (\ln x)^n - \frac{n}{m+1} \int x^m (\ln x)^{n-1} dx$$