# Contents

# 1 Basic

## 1.1 .vimrc

```
syntax on
set nu ai bs=2 sw=2 et ve=all cb=unnamed mouse=a ruler
    incsearch hlsearch
```

# 2 Math

## 2.1 FFT

```
typedef complex<double> cpx;
const double PI = acos(-1);
vector<cpx> FFT(vector<cpx> &P, bool inv = 0) {
  assert(__builtin_popcount(P.size()) == 1);
  int lg = 31 - __builtin_clz(P.size()), n = 1 << lg;
      // == P.size();
  for (int j = 1, i = 0; j < n - 1; ++j) {
    for (int k = n >> 1; k > (i ^= k); k >>= 1);
    if (j < i) swap(P[i], P[j]);
  } //bit reverse
  auto w1 = exp((2 - 4 * inv) * PI / n * cpx(0, 1)); //
       order is 1<<lg
  for (int i = 1; i <= lg; ++i) {
    auto wn = pow(w1, 1<<(lg - i)); // order is 1<<i
    for (int k = 0; k < (1<<lg); k += 1 << i) {
      cpx base = 1;
      for (int j = 0; j < (1 << i - 1); ++j, base =
          base * wn) {
        auto t = base * P[k + j + (1 << i - 1)];
        auto u = P[k + j];
        P[k + j] = u + t;
        P[k + j + (1 << i - 1)] = u - t;
      }
    }
  }
  if(inv)
    for (int i = 0; i < n; ++i) P[i] /= n;
  return P;
} //faster performance with calling by reference
```

## 2.2 FWT

```
vector<int> fast_OR_transform(vector<int> f, bool
    inverse) {
  for (int i = 0; (2 << i) <= f.size(); ++i)
    for (int j = 0; j < f.size(); j += 2 << i)
      for (int k = 0; k < (1 << i); ++k)
        f[j + k + (1 << i)] += f[j + k] * (inverse? -1
            : 1);
  return f;
}
vector<int> rev(vector<int> A) {
  for (int i = 0; i < A.size(); i += 2) swap(A[i], A[i
      ^ (A.size() - 1)]);
  return A;
}
vector<int> fast_AND_transform(vector<int> f, bool
    inverse) {
  return rev(fast_OR_transform(rev(f), inverse));
}
vector<int> fast_XOR_transform(vector<int> f, bool
    inverse) {
  for (int i = 0; (2 << i) <= f.size(); ++i)
    for (int j = 0; j < f.size(); j += 2 << i)
      for (int k = 0; k < (1 << i); ++k) {
        int u = f[j + k], v = f[j + k + (1 << i)];
        f[j + k + (1 << i)] = u - v, f[j + k] = u + v;
      }
  if (inverse) for (auto &a : f) a /= f.size();
  return f;
}
```

## 2.3 NTT

```
/* p == (a << n) + 1
   n    1 << n      p          a    root
   5    32          97         3    5
   6    64          193        3    5
   7    128         257        2    3
   8    256         257        1    3
   9    512         7681       15   17
   10   1024        12289      12   11
   11   2048        12289      6    11
   12   4096        12289      3    11
   13   8192        40961      5    3
   14   16384       65537      4    3
   15   32768       65537      2    3
   16   65536       65537      1    3
   17   131072      786433     6    10
   18   262144      786433     3    10 (605028353,
       2308, 3)
   19   524288      5767169    11   3
   20   1048576     7340033    7    3
   21   2097152     23068673   11   3
   22   4194304     104857601  25   3
   23   8388608     167772161  20   3
   24   16777216    167772161  10   3
   25   33554432    167772161  5    3 (1107296257, 33,
       10)
   26   67108864    469762049  7    3
   27   134217728   2013265921 15   31 */
LL root = 10, p = 786433, a = 3;
LL powM(LL x, LL b) {
  LL s = 1, m = x % p;
  for (; b; m = m * m % p, b >>= 1)
    if (b&1) s = s * m % p;
  return s;
}
vector<LL> NTT(vector<LL> P, bool inv = 0) {
  assert(__builtin_popcount(P.size()) == 1);
  int lg = 31 - __builtin_clz(P.size()), n = 1 << lg;
      // == P.size();
  for (int j = 1, i = 0; j < n - 1; ++j) {
    for (int k = n >> 1; k > (i ^= k); k >>= 1);
    if (j < i) swap(P[i], P[j]);
  } //bit reverse
  LL w1 = powM(root, a * (inv? p - 2: 1)); // order is
      1<<lg
  for (LL i = 1; i <= lg; ++i) {
    LL wn = powM(w1, 1<<(lg - i)); // order is 1<<i
    for (int k = 0; k < (1<<lg); k += 1 << i) {
      LL base = 1;
      for (int j = 0; j < (1 << i - 1); ++j, base =
          base * wn % p) {
        LL t = base * P[k + j + (1 << i - 1)] % p;
        LL u = P[k + j] % p;
        P[k + j] = (u + t) % p;
        P[k + j + (1 << i - 1)] = (u - t + p) % p;
      }
    }
  }
  if(inv){
    LL invN = powM(n, p - 2);
    transform(P.begin(), P.end(), P.begin(), [&](LL a)
        {return a * invN % p;});
  }
  return P;
} //faster performance with calling by reference
```

## 2.4 permanent

```
typedef vector<vector<LL> > mat;
LL permanent(mat A) {
  LL n = A.size(), ans = 0, *tmp = new LL[n], add;
  for (int pgray = 0, s = 1, gray, i; s < 1 << n; ++s)
      {
    gray = s ^ s >> 1, add = 1;
    i = __builtin_ctz(pgray ^ gray);
    for (int j = 0; j < n; ++j)
      add *= tmp[j] += A[i][j] * (gray>>i&1 ? 1 : -1);
    ans += add * (s&1^n&1? -1 : 1), pgray = gray;
  }
  return ans;
}
// how many ways to put rooks on a matrix with 0,1 as
    constrain
// 1 - ok to put
// 0 - not ok to put
```

## 2.5 Polynomail root

```
const double eps = 1e-12;
const double inf = 1e+12;
double a[ 10 ], x[ 10 ];
int n;
int sign( double x ){return (x < -eps)?(-1):(x>eps);}
double f(double a[], int n, double x){
  double tmp=1,sum=0;
  for(int i=0;i<=n;i++)
  { sum=sum+a[i]*tmp; tmp=tmp*x; }
  return sum;
}
double binary(double l,double r,double a[],int n){
  int sl=sign(f(a,n,l)),sr=sign(f(a,n,r));
  if(sl==0) return l; if(sr==0) return r;
  if(sl*sr>0) return inf;
  while(r-l>eps){
    double mid=(l+r)/2;
    int ss=sign(f(a,n,mid));
    if(ss==0) return mid;
    if(ss*sl>0) l=mid; else r=mid;
  }
  return l;
}
void solve(int n,double a[],double x[],int &nx){
  if(n==1){ x[1]=-a[0]/a[1]; nx=1; return; }
  double da[10], dx[10]; int ndx;
  for(int i=n;i>=1;i--) da[i-1]=a[i]*i;
  solve(n-1,da,dx,ndx);
  nx=0;
  if(ndx==0){
    double tmp=binary(-inf,inf,a,n);
    if (tmp<inf) x[++nx]=tmp;
    return;
  }
  double tmp;
  tmp=binary(-inf,dx[1],a,n);
  if(tmp<inf) x[++nx]=tmp;
  for(int i=1;i<=ndx-1;i++){
    tmp=binary(dx[i],dx[i+1],a,n);
    if(tmp<inf) x[++nx]=tmp;
  }
  tmp=binary(dx[ndx],inf,a,n);
  if(tmp<inf) x[++nx]=tmp;
}
int main() {
  scanf("%d",&n);
  for(int i=n;i>=0;i--) scanf("%lf",&a[i]);
  int nx;
  solve(n,a,x,nx);
  for(int i=1;i<=nx;i++) printf("%.6f\n",x[i]);
}
```

# 3 Number Theory

## 3.1 basic

```
PLL exd_gcd(LL a, LL b) {
  if (a % b == 0) return {0, 1};
  PLL T = exd_gcd(b, a % b);
  return {T.second, T.first - a / b * T.second};
}
LL mul(LL x, LL y, LL mod) {
  LL ans = 0, m = abs(x), s = 0, sgn = (x > 0) xor (y >
      0)? -1: 1;
  for (x = abs(x), y = abs(y); y; y >>= 1, m <<= 1, m =
      m >= mod? m - mod: m)
```

```
    if (y&1) s += m, s = s >= mod? s - mod: s;
  return (s * sgn % mod + mod) % mod;
}
LL dangerous_mul(LL a, LL b, LL mod){ // 10 times
     faster than the above in average, but could be
     prone to wrong answer (extreme low prob?)
  return (a * b - (LL)((long double)a * b / mod) * mod)
        % mod;
}
LL powmod(LL x, LL p, LL mod) {
  LL s = 1, m = x % mod;
  for (; p; m = mul(m, m, mod), p >>= 1)
    if (p&1) s = mul(s, m, mod);
  return s;
}
```

## 3.2 Chinese Remainder Theorem

```
PLL CRT(PLL eq1, PLL eq2) {
  LL m1, m2, x1, x2;
  tie(x1, m1) = eq1, tie(x2, m2) = eq2;
  LL g = __gcd(m1, m2);
  if ((x1 - x2) % g) return {-1, 0}; // NO SOLUTION
  m1 /= g, m2 /= g;
  auto p = exd_gcd(m1, m2);
  LL lcm = m1 * m2 * g, res = mul(mul(p.first, (x2 - x1
      ), lcm), m1, lcm) + x1;
  return {(res % lcm + lcm) % lcm, lcm};
}
```

## 3.3 Discrete Log

```
LL discrete_log(LL b, LL p, LL n) {
  map<LL, LL> att;
  LL m = sqrt((double)p) + 1, M = powmod(b, m * (p - 2)
      , p);
  for (LL cur = 1, i = 0; i < m; ++i, cur = cur * b % p
      )
    if (not att.count(cur)) att[cur] = i;
  for (LL cur = 1, i = 0; i * m < p - 1; ++i, cur = cur
       * M % p)
    if (att.count(n * cur % p))
      return (att[cur * n % p] + i * m) % (p - 1);
  return -1;
}
// find x s.t. b**x % p == n with complexity O(sqrt(N))
// return the smallest
// return -1 if ans doesn't exist
```

## 3.4 Lucas

```
LL fac[100000] = {1};
LL C(LL a, LL b, LL p) {
  for (int i = 1; i <= p; ++i) fac[i] = fac[i - 1] * i
      % p;
  LL ans = 1;
  for (;a; a /= p, b /= p) {
    LL A = a % p, B = b % p;
    if (A < B) return 0;
    (ans += fac[A] * powmod(fac[B] * fac[A - B] % p, p
        - 2, p) % p) %= p;
  }
  return ans;
}
```

## 3.5 Meissel-Lehmer PI

```
LL PI(LL m);
const int MAXM = 1000, MAXN = 650, UPBD = 1000000;
// 650 ~ PI(cbrt(1e11))
LL pi[UPBD] = {0}, phi[MAXM][MAXN];
vector<LL> primes;
void init() {
  fill(pi + 2, pi + UPBD, 1);
  for (LL p = 2; p < UPBD; ++p)
    if (pi[p]) {
      for (LL N = p * p; N < UPBD; N += p)
        pi[N] = 0;
      primes.push_back(p);
    }
  for (int i = 1; i < UPBD; ++i) pi[i] += pi[i - 1];
  for (int i = 0; i < MAXM; ++i)
    phi[i][0] = i;
  for (int i = 1; i < MAXM; ++i)
    for (int j = 1; j < MAXN; ++j)
      phi[i][j] = phi[i][j - 1] - phi[i / primes[j -
          1]][j - 1];
}
LL P_2(LL m, LL n) {
  LL ans = 0;
  for (LL i = n; primes[i] * primes[i] <= m and i <
      primes.size(); ++i)
    ans += PI(m / primes[i]) - i;
  return ans;
}
LL PHI(LL m, LL n) {
  if (m < MAXM and n < MAXN) return phi[m][n];
  if (n == 0) return m;
  LL p = primes[n - 1];
  if (m < UPBD) {
    if (m <= p) return 1;
    if (m <= p * p * p) return pi[m] - n + 1 + P_2(m, n
        );
  }
  return PHI(m, n - 1) - PHI(m / p, n - 1);
}
LL PI(LL m) {
  if (m < UPBD) return pi[m];
  LL y = cbrt(m) + 10, n = pi[y];
  return PHI(m, n) + n - 1 - P_2(m, n);
}
```

## 3.6 Miller Rabin with Pollard rho

```
// Miller_Rabin
LL abs(LL a) {return a > 0? a: -a;}
bool witness(LL a, LL n, LL u, int t) {
  LL x = modpow(a, u, n), nx;
  for (int i = 0; i < t; ++i, x = nx){
    nx = mul(x, x, n);
    if (nx == 1 and x != 1 and x != n - 1) return 1;
  }
  return x != 1;
}
const LL wits[7] = {2, 325, 9375, 28178, 450775,
    9780504, 1795265022};
bool miller_rabin(LL n, int s = 7) {
  if (n < 2) return 0;
  if (n&1^1) return n == 2;
  LL u = n - 1, t = 0, a; // n == (u << t) + 1
  while (u&1^1) u >>= 1, ++t;
  while (s--)
    if (a = wits[s] % n and witness(a, n, u, t)) return
         0;
  return 1;
}
// Pollard_rho
LL f(LL x, LL n) {
  return mul(x, x, n) + 1;
}
LL pollard_rho(LL n) {
  if (n&1^1) return 2;
  while (true) {
    LL x = rand() % (n - 1) + 1, y = 2, d = 1;
    for (int sz = 2; d == 1; y = x, sz <<= 1)
      for (int i = 0; i < sz and d <= 1; ++i)
        x = f(x, n), d = __gcd(abs(x - y), n);
    if (d and n - d) return d;
  }
}
```

### 3.7 Primitive Root

```
vector<LL> factor(LL N) {
  vector<LL> ans;
  for (LL p = 2, n = N; p * p <= n; ++p)
    if (N % p == 0) {
      ans.push_back(p);
      while (N % p == 0) N /= p;
    }
  if (N != 1) ans.push_back(N);
  return ans;
}
LL find_root(LL p) {
  LL ans = 1;
  for (auto q: factor(p - 1)) {
    LL a = rand() % (p - 1) + 1, b = (p - 1) / q;
    while (powmod(a, b, p) == 1) a = rand() % (p - 1) +
         1;
    while (b % q == 0) b /= q;
    ans = mul(ans, powmod(a, b, p), p);
  }
  return ans;
}
bool is_root(LL a, LL p) {
  for (auto q: factor(p - 1))
    if (powmod(a, (p - 1) / q, p) == 1)
      return false;
  return true;
}
```

### 3.8 Permanent

```
typedef vector<vector<LL> > mat;
LL permanent(mat A) {
  LL n = A.size(), ans = 0, *tmp = new LL[n], add;
  for (int pgray = 0, s = 1, gray, i; s < 1 << n; ++s)
      {
    gray = s ^ s >> 1, add = 1;
    i = __builtin_ctz(pgray ^ gray);
    for (int j = 0; j < n; ++j)
      add *= tmp[j] += A[i][j] * (gray>>i&1 ? 1 : -1);
    ans += add * (s&1^n&1? -1 : 1), pgray = gray;
  }
  return ans;
}
// how many ways to put rooks on a matrix with 0,1 as
    constrain
// 1 - ok to put
// 0 - not ok to put
```

## 4 Data Structure

### 4.1 Heavy Light Decomposition

```
struct HLD {
  using Tree = vector<vector<int>>;
  vector<int> par, head, vid, len, inv;

  HLD(const Tree &g) : par(g.size()), head(g.size()),
      vid(g.size()), len(g.size()), inv(g.size()) {
    int k = 0;
    vector<int> size(g.size(), 1);
    function<void(int, int)> dfs_size = [&](int u, int
        p) {
      for (int v : g[u]) {
        if (v != p) {
          dfs_size(v, u);
          size[u] += size[v];
        }
      }
    };
    function<void(int, int, int)> dfs_dcmp = [&](int u,
         int p, int h) {
      par[u] = p;
      head[u] = h;
      vid[u] = k++;
      inv[vid[u]] = u;
      for (int v : g[u]) {
        if (v != p && size[u] < size[v] * 2) {
          dfs_dcmp(v, u, h);
        }
      }
      for (int v : g[u]) {
        if (v != p && size[u] >= size[v] * 2) {
          dfs_dcmp(v, u, v);
        }
      }
    };
    dfs_size(0, -1);
    dfs_dcmp(0, -1, 0);
    for (int i = 0; i < g.size(); ++i) {
      ++len[head[i]];
    }
  }

  template<typename T>
  void foreach(int u, int v, T f) {
    while (true) {
      if (vid[u] > vid[v]) {
        if (head[u] == head[v]) {
          f(vid[v] + 1, vid[u], 0);
          break;
        } else {
          f(vid[head[u]], vid[u], 1);
          u = par[head[u]];
        }
      } else {
        if (head[u] == head[v]) {
          f(vid[u] + 1, vid[v], 0);
          break;
        } else {
          f(vid[head[v]], vid[v], 0);
          v = par[head[v]];
        }
      }
    }
  }
};
```

### 4.2 KDTree

```
#include <bits/stdc++.h>
using namespace std;

struct KDNode {
  vector<int> v;
  KDNode *lc, *rc;
  KDNode(const vector<int> &_v) : v(_v), lc(nullptr),
      rc(nullptr) {}
  static KDNode *buildKDTree(vector<vector<int>> &pnts,
       int lb, int rb, int dpt) {
    if (rb - lb < 1) return nullptr;
    int axis = dpt % pnts[0].size();
    int mb = lb + rb >> 1;
    nth_element(pnts.begin() + lb, pnts.begin() + mb,
        pnts.begin() + rb, [&](const vector<int> &a,
        const vector<int> &b) {
      return a[axis] < b[axis];
    });
    KDNode *t = new KDNode(pnts[mb]);
    t->lc = buildKDTree(pnts, lb, mb, dpt + 1);
    t->rc = buildKDTree(pnts, mb + 1, rb, dpt + 1);
    return t;
  }
  static void release(KDNode *t) {
    if (t->lc) release(t->lc);
    if (t->rc) release(t->rc);
    delete t;
  }
  static void searchNearestNode(KDNode *t, KDNode *q,
      KDNode *&c, int dpt) {
    int axis = dpt % t->v.size();
    if (t->v != q->v && (c == nullptr || dis(q, t) <
        dis(q, c))) c = t;
    if (t->lc && (!t->rc || q->v[axis] < t->v[axis])) {
      searchNearestNode(t->lc, q, c, dpt + 1);
      if (t->rc && (c == nullptr || 1LL * (t->v[axis] -
           q->v[axis]) * (t->v[axis] - q->v[axis]) <
          dis(q, c))) {
        searchNearestNode(t->rc, q, c, dpt + 1);
      }
    } else if (t->rc) {
      searchNearestNode(t->rc, q, c, dpt + 1);
      if (t->lc && (c == nullptr || 1LL * (t->v[axis] -
           q->v[axis]) * (t->v[axis] - q->v[axis]) <
          dis(q, c))) {
        searchNearestNode(t->lc, q, c, dpt + 1);
      }
    }
  }
  static int64_t dis(KDNode *a, KDNode *b) {
    int64_t r = 0;
    for (int i = 0; i < a->v.size(); ++i) {
      r += 1LL * (a->v[i] - b->v[i]) * (a->v[i] - b->v[
          i]);
    }
    return r;
  }
};

signed main() {
  ios::sync_with_stdio(false);
  int T;
  cin >> T;
  for (int ti = 0; ti < T; ++ti) {
    int N;
    cin >> N;
    vector<vector<int>> pnts(N, vector<int>(2));
    for (int i = 0; i < N; ++i) {
      for (int j = 0; j < 2; ++j) {
        cin >> pnts[i][j];
      }
    }
    vector<vector<int>> _pnts = pnts;
    KDNode *root = KDNode::buildKDTree(_pnts, 0, pnts.
        size(), 0);
    for (int i = 0; i < N; ++i) {
      KDNode *q = new KDNode(pnts[i]);
      KDNode *c = nullptr;
      KDNode::searchNearestNode(root, q, c, 0);
      cout << KDNode::dis(c, q) << endl;
      delete q;
    }
    KDNode::release(root);
  }
  return 0;
}
```

# 5 Flow

## 5.1 CostFlow

```
template <class TF, class TC>
struct CostFlow {
  static const int MAXV = 205;
  static const TC INF = 0x3f3f3f3f;
  struct Edge {
    int v, r;
    TF f;
    TC c;
    Edge(int _v, int _r, TF _f, TC _c) : v(_v), r(_r),
        f(_f), c(_c) {}
  };
  int n, s, t, pre[MAXV], pre_E[MAXV], inq[MAXV];
  TF fl;
  TC dis[MAXV], cost;
  vector<Edge> E[MAXV];
  CostFlow(int _n, int _s, int _t) : n(_n), s(_s), t(_t
      ), fl(0), cost(0) {}
  void add_edge(int u, int v, TF f, TC c) {
    E[u].emplace_back(v, E[v].size(), f, c);
    E[v].emplace_back(u, E[u].size() - 1, 0, -c);
  }
  pair<TF, TC> flow() {
    while (true) {
      for (int i = 0; i < n; ++i) {
        dis[i] = INF;
        inq[i] = 0;
      }
      dis[s] = 0;
      queue<int> que;
      que.emplace(s);
      while (not que.empty()) {
        int u = que.front();
        que.pop();
        inq[u] = 0;
        for (int i = 0; i < E[u].size(); ++i) {
          int v = E[u][i].v;
          TC w = E[u][i].c;
          if (E[u][i].f > 0 and dis[v] > dis[u] + w) {
            pre[v] = u;
            pre_E[v] = i;
            dis[v] = dis[u] + w;
            if (not inq[v]) {
              inq[v] = 1;
              que.emplace(v);
            }
          }
        }
      }
      if (dis[t] == INF) break;
      TF tf = INF;
      for (int v = t, u, l; v != s; v = u) {
        u = pre[v];
        l = pre_E[v];
        tf = min(tf, E[u][l].f);
      }
      for (int v = t, u, l; v != s; v = u) {
        u = pre[v];
        l = pre_E[v];
        E[u][l].f -= tf;
        E[v][E[u][l].r].f += tf;
      }
      cost += tf * dis[t];
      fl += tf;
    }
    return {fl, cost};
  }
};
```

## 5.2 Dinic

```
template <class T>
struct Dinic {
  static const int MAXV = 10000;
  static const T INF = 0x3f3f3f3f;
  struct Edge {
    int v;
    T f;
    int re;
    Edge(int _v, T _f, int _re) : v(_v), f(_f), re(_re)
         {}
  };
  int n, s, t, level[MAXV];
  vector<Edge> E[MAXV];
  int now[MAXV];
  Dinic(int _n, int _s, int _t) : n(_n), s(_s), t(_t)
      {}
  void add_edge(int u, int v, T f, bool bidirectional =
       false) {
    E[u].emplace_back(v, f, E[v].size());
    E[v].emplace_back(u, 0, E[u].size() - 1);
    if (bidirectional) {
      E[v].emplace_back(u, f, E[u].size() - 1);
    }
  }
  bool BFS() {
    memset(level, -1, sizeof(level));
    queue<int> que;
    que.emplace(s);
    level[s] = 0;
    while (not que.empty()) {
      int u = que.front();
      que.pop();
      for (auto it : E[u]) {
        if (it.f > 0 and level[it.v] == -1) {
          level[it.v] = level[u] + 1;
          que.emplace(it.v);
        }
      }
    }
    return level[t] != -1;
  }
  T DFS(int u, T nf) {
    if (u == t) return nf;
    T res = 0;
    while (now[u] < E[u].size()) {
      Edge &it = E[u][now[u]];
      if (it.f > 0 and level[it.v] == level[u] + 1) {
        T tf = DFS(it.v, min(nf, it.f));
        res += tf;
        nf -= tf;
        it.f -= tf;
        E[it.v][it.re].f += tf;
        if (nf == 0) return res;
      } else
        ++now[u];
    }
    if (not res) level[u] = -1;
    return res;
  }
  T flow(T res = 0) {
    while (BFS()) {
      T temp;
      memset(now, 0, sizeof(now));
      while (temp = DFS(s, INF)) {
        res += temp;
        res = min(res, INF);
      }
    }
    return res;
  }
};
```

# 6 Geometry

## 6.1 2D Geometry

```
namespace geo {
  using pt = complex<double>;
  using cir = pair<pt, double>;
  using poly = vector<pt>;
  using line = pair<pt, pt>; // point to point
  using plane = pair<pt, pt>;
  pt get_pt() { static double a, b; scanf("%lf%lf", &a,
       &b); return geo::pt(a, b);};
  const double EPS = 1e-10;
  const double PI = acos(-1);
  pt cent(cir C) { return C.first; }
  double radi(cir C) { return C.second; }
  pt st(line H) { return H.first; }
  pt ed(line H) { return H.second; }
  pt vec(line H) { return ed(H) - st(H); }
  int dcmp(double x) { return abs(x) < EPS ? 0 : x > 0
      ? 1 : -1; }
  bool less(pt a, pt b) { return real(a) < real(b) ||
      real(a) == real(b) && imag(a) < imag(b); }
  bool more(pt a, pt b) { return real(a) > real(b) ||
      real(a) == real(b) && imag(a) > imag(b); }
  double dot(pt a, pt b) { return real(conj(a) * b); }
  double cross(pt a, pt b) { return imag(conj(a) * b);
      }
  double sarea(pt a, pt b, pt c) { return cross(b - a,
      c - a); }
  double area(cir c) { return radi(c) * radi(c) * PI; }
  int ori(pt a, pt b, pt c) { return dcmp(sarea(a, b, c
      )); }
  double angle(pt a, pt b) { return acos(dot(a, b) /
      abs(a) / abs(b)); }
  pt rotate(pt a, double rad) { return a * pt(cos(rad),
       sin(rad)); }
  pt normal(pt a) { return pt(-imag(a), real(a)) / abs(
      a); }
  pt normalized(pt a) { return a / abs(a); }

  pt get_line_intersection(line A, line B) {
    pt p = st(A), v = vec(A), q = st(B), w = vec(B);
    return p + v * cross(w, p - q) / cross(v, w);
  }
  double distance_to_line(pt p, line B) {
    return abs(cross(vec(B), p - st(B)) / abs(vec(B)));
  }
  double distance_to_segment(pt p, line B) {
    pt a = st(B), b = ed(B), v1(vec(B)), v2(p - a), v3(
        p - b);
    // similar to previous function
    if (a == b) return abs(p - a);
    if (dcmp(dot(v1, v2)) < 0) return abs(v2);
    else if (dcmp(dot(v1, v3)) > 0) return abs(v3);
    return abs(cross(v1, v2)) / abs(v1);
  }
  pt get_line_projection(pt p, line(B)) {
    pt v = vec(B);
    return st(B) + dot(v, p - st(B)) / dot(v, v) * v;
  }
  bool is_segment_proper_intersection(line A, line B) {
    pt a1 = st(A), a2 = ed(A), b1 = st(B), b2 = ed(B);
    double det1 = ori(a1, a2, b1) * ori(a1, a2, b2);
    double det2 = ori(b1, b2, a1) * ori(b1, b2, a2);
    return det1 < 0 && det2 < 0;
  }
  double area(poly p) {
    if (p.size() < 3) return 0;
    double area = 0;
    for (int i = 1; i < p.size() - 1; ++i)
      area += sarea(p[0], p[i], p[i + 1]);
    return area / 2;
  }
  bool is_point_on_segment(pt p, line B) {
    pt a = st(B), b = ed(B);
    return dcmp(sarea(p, a, b)) == 0 && dcmp(dot(a - p,
         b - p)) < 0;
  }
  bool is_point_in_plane(pt p, line H) {
    return ori(st(H), ed(H), p) > 0;
  }
  int is_point_in_polygon(pt p, poly gon) {
    int wn = 0;
    int n = gon.size();
    for (int i = 0; i < n; ++i) {
      if (is_point_on_segment(p, {gon[i], gon[(i + 1) %
           n]})) return true;
      if (is_point_in_plane(p, {gon[i], gon[(i + 1) % n
          ]})) return false;
    }
    return true;
  }
  poly convex_hull(vector<pt> p) {
    sort(p.begin(), p.end(), less);
    p.erase(unique(p.begin(), p.end()), p.end());
    int n = p.size(), m = 0;
    poly ch(n + 1);
    for (int i = 0; i < n; ++i) { // note that border
        is cleared
      while (m > 1 && ori(ch[m - 2], ch[m - 1], p[i])
          <= 0) --m;
      ch[m++] = p[i];
    }
    for (int i = n - 2, k = m; i >= 0; --i) {
      while (m > k && ori(ch[m - 2], ch[m - 1], p[i])
          <= 0) --m;
      ch[m++] = p[i];
    }
    ch.erase(ch.begin() + m - (n > 1), ch.end());
    return ch;
  }
  cir circumscribed_circle(poly tri) {
    pt B = tri[1] - tri[0];
    pt C = tri[2] - tri[0];
    double det = 2 * cross(B, C);
    pt r = pt(imag(C) * norm(B) - imag(B) * norm(C),
              real(B) * norm(C) - real(C) * norm(B)) /
                  det;
    return {r + tri[0], abs(r)};
  }
  cir inscribed_circle(poly tri) {
    assert(tri.size() == 3);
    pt ans = 0;
    double div = 0;
    for (int i = 0; i < 3; ++i) {
      double l = abs(tri[(i + 1) % 3] - tri[(i + 2) % 3]);
      ans += l * tri[i], div += l;
    }
    ans /= div;
    return {ans, distance_to_line(ans, {tri[0], tri[1]})};
  }
  poly tangent_line_through_point(cir c, pt p) {
    if (dcmp(abs(cent(c) - p) - radi(c)) < 0) return {};
    else if (dcmp(abs(cent(c) - p) - radi(c)) == 0) return {p};
    double theta = acos(radi(c) / abs(cent(c) - p));
    pt norm_v = normalized(p - cent(c));
    return {cent(c) + radi(c) * rotate(norm_v, +theta),
            cent(c) + radi(c) * rotate(norm_v, -theta)};
  }
  vector<pt> get_line_circle_intersection(cir d, line B) {
    pt v = vec(B), p = st(B) - cent(d);
    double r = radi(d), a = norm(v), b = 2 * dot(p, v), c = norm(p) - r * r;
    double det = b * b - 4 * a * c;
    // t^2 * norm(v) + 2 * t * dot(p, v) + norm(p) - r * r = 0
    auto get_point = [=](double t) { return st(B)+ t * v; };
    if (dcmp(det) < 0) return {};
    if (dcmp(det) == 0) return {get_point(-b / 2 / a)};
    return {get_point((-b + sqrt(det)) / 2 / a),
            get_point((-b - sqrt(det)) / 2 / a)};
  }
  vector<pt> get_circle_circle_intersection(cir c, cir d) {
    pt a = cent(c), b = cent(d);
    double r = radi(c), s = radi(d), g = abs(a - b);
    if (dcmp(g) == 0) return {}; // may be C == D
    if (dcmp(r + s - g) < 0 or dcmp(abs(r - s) - g) > 0) return {};
    pt C_to_D = normalized(b - a);
    double theta = acos((r * r + g * g - s * s) / (2 * r * g));
    if (dcmp(theta) == 0) return {a + r * C_to_D};
    else return {a + rotate(r * C_to_D, theta), a + rotate(r * C_to_D, -theta)};
  }
  cir min_circle_cover(vector<pt> A) {
    random_shuffle(A.begin(), A.end());
    cir ans = {0, 0};
    auto is_incir = [&](pt a) { return dcmp(abs(cent(ans) - a) - radi(ans)) < 0; };
    for (int i = 0; i < A.size(); ++i) if (not is_incir(A[i])) {
      ans = {A[i], 0};
      for (int j = 0; j < i; ++j) if (not is_incir(A[j])) {
        ans = {(A[i] + A[j]) / 2., abs(A[i] - A[j]) / 2};
        for (int k = 0; k < j; ++k) if (not is_incir(A[k]))
          ans = circumscribed_circle({A[i], A[j], A[k]});
      }
    }
    return ans;
  }
  poly half_plane_intersection(vector<plane> A) {
    const double INF = 1e19;
    sort(A.begin(), A.end(), [=](plane a, plane b) {
      int res = dcmp(arg(vec(a)) - arg(vec(b)));
      return res == 0 ? is_point_in_plane(st(a), b) : res < 0;
    });
    deque<pt> ans;
    deque<plane> q;
    q.push_back(A[0]);
    for (int i = 1; i < A.size(); ++i) {
      if (dcmp(cross(vec(A[i]), vec(A[i - 1]))) == 0)
        continue;
      while (ans.size() and not is_point_in_plane(ans.back(), A[i]))
        q.pop_back(), ans.pop_back();
      while (ans.size() and not is_point_in_plane(ans.front(), A[i]))
        q.pop_front(), ans.pop_front();
      ans.push_back(get_line_intersection(A[i], q.back()));
      q.push_back(A[i]);
    }
    while (ans.size() and not is_point_in_plane(ans.back(), q.front()))
      ans.pop_back(), q.pop_back();
    while (ans.size() and not is_point_in_plane(ans.front(), q.back()))
      ans.pop_front(), q.pop_front();
    if (q.size() < 3) return {};
    ans.push_back(get_line_intersection(q.back(), q.front()));
    return poly(ans.begin(), ans.end());
  }
};
```

## 6.2 3DConvexHull

```
#define SIZE(X) (int(X.size()))
#define PI 3.14159265358979323846264338327950288
struct Pt{
  Pt cross(const Pt &p) const
  { return Pt(y * p.z - z * p.y, z * p.x - x * p.z, x * p.y - y * p.x); }
} info[N];
int mark[N][N],n, cnt;;
double mix(const Pt &a, const Pt &b, const Pt &c)
{ return a * (b ^ c); }
double area(int a, int b, int c)
{ return norm((info[b] - info[a]) ^ (info[c] - info[a])); }
double volume(int a, int b, int c, int d)
{ return mix(info[b] - info[a], info[c] - info[a], info[d] - info[a]); }
struct Face{
  int a, b, c; Face(){}
  Face(int a, int b, int c): a(a), b(b), c(c) {}
  int &operator [](int k)
  { if (k == 0) return a; if (k == 1) return b; return c; }
};
vector<Face> face;
void insert(int a, int b, int c)
{ face.push_back(Face(a, b, c)); }
void add(int v) {
  vector <Face> tmp; int a, b, c; cnt++;
  for (int i = 0; i < SIZE(face); i++) {
    a = face[i][0]; b = face[i][1]; c = face[i][2];
    if(Sign(volume(v, a, b, c)) < 0)
    mark[a][b] = mark[b][a] = mark[b][c] = mark[c][b] = mark[c][a] = mark[a][c] = cnt;
    else tmp.push_back(face[i]);
  } face = tmp;
  for (int i = 0; i < SIZE(tmp); i++) {
    a = face[i][0]; b = face[i][1]; c = face[i][2];
    if (mark[a][b] == cnt) insert(b, a, v);
    if (mark[b][c] == cnt) insert(c, b, v);
    if (mark[c][a] == cnt) insert(a, c, v);
}}
int Find(){
  for (int i = 2; i < n; i++) {
    Pt ndir = (info[0] - info[i]) ^ (info[1] - info[i]);
    if (ndir == Pt()) continue; swap(info[i], info[2]);
    for (int j = i + 1; j < n; j++) if (Sign(volume(0, 1, 2, j)) != 0) {
      swap(info[j], info[3]); insert(0, 1, 2); insert(0, 2, 1); return 1;
} } return 0; }
```

```
int main() {
  for (; scanf("%d", &n) == 1; ) {
    for (int i = 0; i < n; i++) info[i].Input();
    sort(info, info + n); n = unique(info, info + n) -
        info;
    face.clear(); random_shuffle(info, info + n);
    if (Find()) { memset(mark, 0, sizeof(mark)); cnt =
        0;
      for (int i = 3; i < n; i++) add(i); vector<Pt>
          Ndir;
      for (int i = 0; i < SIZE(face); ++i) {
        Pt p = (info[face[i][0]] - info[face[i][1]]) ^
               (info[face[i][2]] - info[face[i][1]]);
        p = p / norm( p ); Ndir.push_back(p);
      } sort(Ndir.begin(), Ndir.end());
      int ans = unique(Ndir.begin(), Ndir.end()) - Ndir
          .begin();
      printf("%d\n", ans);
    } else printf("1\n");
} }
double calcDist(const Pt &p, int a, int b, int c)
{ return fabs(mix(info[a] - p, info[b] - p, info[c] - p
    ) / area(a, b, c)); }
//compute the minimal distance of center of any faces
double findDist() { //compute center of mass
  double totalWeight = 0; Pt center(.0, .0, .0);
  Pt first = info[face[0][0]];
  for (int i = 0; i < SIZE(face); ++i) {
    Pt p = (info[face[i][0]]+info[face[i][1]]+info[face
        [i][2]]+first)*.25;
    double weight = mix(info[face[i][0]] - first, info[
        face[i][1]]
          - first, info[face[i][2]] - first);
    totalWeight += weight; center = center + p * weight
        ;
  } center = center / totalWeight;
  double res = 1e100; //compute distance
  for (int i = 0; i < SIZE(face); ++i)
    res = min(res, calcDist(center, face[i][0], face[i
        ][1], face[i][2]));
    return res; }
```

## 6.3 Half plane intersection

```
Pt interPnt( Line l1, Line l2, bool &res ){
  Pt p1, p2, q1, q2;
  tie(p1, p2) = l1; tie(q1, q2) = l2;
  double f1 = (p2 - p1) ^ (q1 - p1);
  double f2 = (p2 - p1) ^ (p1 - q2);
  double f = (f1 + f2);
  if( fabs(f) < eps){ res=0; return {0, 0}; }
  res = true;
  return q1 * (f2 / f) + q2 * (f1 / f);
}
bool isin( Line l0, Line l1, Line l2 ){
  // Check inter(l1, l2) in l0
  bool res; Pt p = interPnt(l1, l2, res);
  return ( (l0.SE - l0.FI) ^ (p - l0.FI) ) > eps;
}
/* If no solution, check: 1. ret.size() < 3
 * Or more precisely, 2. interPnt(ret[0], ret[1])
 * in all the lines. (use (l.S - l.F) ^ (p - l.F) > 0
 */
/* --^-- Line.FI --^-- Line.SE --^-- */
vector<Line> halfPlaneInter( vector<Line> lines ){
  int sz = lines.size();
  vector<double> ata(sz), ord(sz);
  for( int i=0; i<sz; i++) {
    ord[i] = i;
    Pt d = lines[i].SE - lines[i].FI;
    ata[i] = atan2(d.Y, d.X);
  }
  sort( ord.begin(), ord.end(), [&](int i, int j) {
    if( fabs(ata[i] - ata[j]) < eps )
      return ( (lines[i].SE - lines[i].FI) ^
               (lines[j].SE - lines[i].FI) ) < 0;
    return ata[i] < ata[j];
  });
  vector<Line> fin;
  for (int i=0; i<sz; i++)
    if (!i or fabs(ata[ord[i]] - ata[ord[i-1]]) > eps)
      fin.PB(lines[ord[i]]);
  deque<Line> dq;
  for (int i=0; i<(int)(fin.size()); i++) {
    while((int)(dq.size()) >= 2 and
        not isin(fin[i], dq[(int)(dq.size())-2],
                         dq[(int)(dq.size())-1]))
      dq.pop_back();
    while((int)(dq.size()) >= 2 and
        not isin(fin[i], dq[0], dq[1]))
      dq.pop_front();
    dq.push_back(fin[i]);
  }
  while( (int)(dq.size()) >= 3 and
      not isin(dq[0], dq[(int)(dq.size())-2],
                      dq[(int)(dq.size())-1]))
    dq.pop_back();
  while( (int)(dq.size()) >= 3 and
      not isin(dq[(int)(dq.size())-1], dq[0], dq[1]))
    dq.pop_front();
  vector<Line> res(dq.begin(),dq.end());
  return res;
}
```

## 6.4 polyUnion

```
#define eps 1e-8
class PY{ public:
  int n;
  Pt pt[5];
  Pt& operator[](const int x){ return pt[x]; }
  void input(){
    int i; n=4;
    for(i=0;i<n;i++) scanf("%lf%lf",&pt[i].x,&pt[i].y);
  }
  double getArea(){
    int i; double s=pt[n-1]^pt[0];
    for(i=0;i<n-1;i++) s+=pt[i]^pt[i+1];
    return s/2;
  }
};
PY py[500];
pair<double,int> c[5000];
inline double segP(Pt &p,Pt &p1,Pt &p2){
  if(SG(p1.x-p2.x)==0) return (p.y-p1.y)/(p2.y-p1.y);
  return (p.x-p1.x)/(p2.x-p1.x);
}
double polyUnion(int n){
  int i,j,ii,jj,ta,tb,r,d;
  double z,w,s,sum,tc,td;
  for(i=0;i<n;i++) py[i][py[i].n]=py[i][0];
  sum=0;
  for(i=0;i<n;i++){
    for(ii=0;ii<py[i].n;ii++){
      r=0;
      c[r++]=make_pair(0.0,0);
      c[r++]=make_pair(1.0,0);
      for(j=0;j<n;j++){
        if(i==j) continue;
        for(jj=0;jj<py[j].n;jj++){
          ta=SG(tri(py[i][ii],py[i][ii+1],py[j][jj]));
          tb=SG(tri(py[i][ii],py[i][ii+1],py[j][jj+1]))
              ;
          if(ta==0 && tb==0){
            if((py[j][jj+1]-py[j][jj])*(py[i][ii+1]-py[
                i][ii])>0 && j<i){
              c[r++]=make_pair(segP(py[j][jj],py[i][ii
                  ],py[i][ii+1]),1);
              c[r++]=make_pair(segP(py[j][jj+1],py[i][
                  ii],py[i][ii+1]),-1);
            }
          }else if(ta>=0 && tb<0){
            tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
            td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
            c[r++]=make_pair(tc/(tc-td),1);
          }else if(ta<0 && tb>=0){
            tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
            td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
            c[r++]=make_pair(tc/(tc-td),-1);
          }
        }
      }
      sort(c,c+r);
      z=min(max(c[0].first,0.0),1.0);
      d=c[0].second; s=0;
      for(j=1;j<r;j++){
        w=min(max(c[j].first,0.0),1.0);
        if(!d) s+=w-z;
        d+=c[j].second; z=w;
      }
      sum+=(py[i][ii]^py[i][ii+1])*s;
    }
  }
  return sum/2;
}
int main(){
  int n,i,j,k;
  double sum,ds;
  scanf("%d",&n); sum=0;
  for(i=0;i<n;i++){
    py[i].input();
    ds=py[i].getArea();
    if(ds<0){
      for(j=0,k=py[i].n-1;j<k;j++,k--) swap(py[i][j], py[i][k]);
      ds=-ds;
    } sum+=ds;
  } printf("%.9f\n",sum/polyUnion(n));
}
```

# 7 Graph

## 7.1 2-SAT

```
#include <cstdio>
#include <vector>
#include <stack>
#include <cstring>
using namespace std;

const int N = 2010;
struct two_SAT {
  int n;
  vector<int> G[N], revG[N];
  stack<int> finish;
  bool sol[N], visit[N];
  int cmp[N];
  void init(int _n) {
    n = _n;
    for (int i = 0; i < N; i++) {
      G[i].clear();
      revG[i].clear();
    }
  }
  void add_edge(int u, int v) {
    // 2 * i -> i is True, 2 * i + 1 -> i is False
    G[u].push_back(v);
    G[v^1].push_back(u^1);
    revG[v].push_back(u);
    revG[u^1].push_back(v^1);
  }
  void dfs(int v) {
    visit[v] = true;
    for ( auto i:G[v] ) {
      if ( !visit[i] ) dfs(i);
    }
    finish.push(v);
  }
  void revdfs(int v, int id) {
    visit[v] = true;
    for ( auto i:revG[v] ) {
      if ( !visit[i] ) revdfs(i,id);
    }
    cmp[v] = id;
  }
  int scc() {
    memset( visit, 0, sizeof(visit) );
    for (int i = 0; i < 2 * n; i++) {
      if ( !visit[i] ) dfs(i);
    }
    int id = 0;
    memset( visit, 0, sizeof(visit) );
    while ( !finish.empty() ) {
      int v = finish.top(); finish.pop();
      if ( visit[v] ) continue;
      revdfs(v,++id);
    }
    return id;
  }
  bool solve() {
    scc();
    for (int i = 0; i < n; i++) {
      if ( cmp[2*i] == cmp[2*i+1] ) return 0;
      sol[i] = ( cmp[2*i] > cmp[2*i+1] );
    }
    return 1;
  }
} sat;

int main() {
  // ( a or not b ) and ( b or c ) and ( not c or not a )
  sat.init(3);
  sat.add_edge( 2*0+1, 2*1+1 );
  sat.add_edge( 2*1+1, 2*2+0 );
  sat.add_edge( 2*2+0, 2*0+1 );
  printf("%d\n", sat.solve() );
  return 0;
}
```

## 7.2 maximal cliques

```
#include <bits/stdc++.h>
using namespace std;

const int N = 60;
typedef long long LL;

struct Bron_Kerbosch {
  int n, res;
  LL edge[N];
  void init(int _n) {
    n = _n;
    for (int i = 0; i <= n; i++) edge[i] = 0;
  }
  void add_edge(int u, int v) {
    if ( u == v ) return;
    edge[u] |= 1LL << v;
    edge[v] |= 1LL << u;
  }
  void go(LL R, LL P, LL X) {
    if ( P == 0 && X == 0 ) {
      res = max( res, __builtin_popcountll(R) ); // notice LL
      return;
    }
    if ( __builtin_popcountll(R) + __builtin_popcountll(P) <= res ) return;
    for (int i = 0; i <= n; i++) {
      LL v = 1LL << i;
      if ( P & v ) {
        go( R | v, P & edge[i], X & edge[i] );
        P &= ~v;
        X |= v;
      }
    }
  }
  int solve() {
    res = 0;
    go( 0LL, ( 1LL << (n+1) ) - 1, 0LL );
    return res;
  }
/*  BronKerbosch1(R, P, X):
      if P and X are both empty:
        report R as a maximal clique
      for each vertex v in P:
        BronKerbosch1(R ▯ {v}, P ▯ N(v), X ▯ N(v))
        P := P \ {v}
        X := X □ {v}
*/
} MaxClique;

int main() {
  MaxClique.init(6);
  MaxClique.add_edge(1,2);
  MaxClique.add_edge(1,5);
  MaxClique.add_edge(2,5);
  MaxClique.add_edge(4,5);
  MaxClique.add_edge(3,2);
  MaxClique.add_edge(4,6);
  MaxClique.add_edge(3,4);
  cout << MaxClique.solve() << "\n";
  return 0;
}
```

## 7.3 Tarjan SCC

```
#include <cstdio>
#include <vector>
#include <stack>
#include <cstring>
using namespace std;

const int N = 10010;
struct Tarjan {
  int n;
  vector<int> G[N], revG[N];
  stack<int> finish;
  bool visit[N];
  int cmp[N];
  void init(int _n) {
    n = _n;
    for (int i = 0; i <= n; i++) {
      G[i].clear();
      revG[i].clear();
    }
  }
  void add_edge(int u, int v) {
    G[u].push_back(v);
    revG[v].push_back(u);
  }
  void dfs(int v) {
    visit[v] = true;
    for ( auto i:G[v] ) {
      if ( !visit[i] ) dfs(i);
    }
    finish.push(v);
  }
  void revdfs(int v, int id) {
    visit[v] = true;
    for ( auto i:revG[v] ) {
      if ( !visit[i] ) revdfs(i,id);
    }
    cmp[v] = id;
  }
  int solve() {
    memset( visit, 0, sizeof(visit) );
    for (int i = 0; i < n; i++) {
      if ( !visit[i] ) dfs(i);
    }
    int id = 0;
    memset( visit, 0, sizeof(visit) );
    while ( !finish.empty() ) {
      int v = finish.top(); finish.pop();
      if ( visit[v] ) continue;
      revdfs(v,++id);
    }
    return id;
  }
} scc;

int main() {
  int V, E;
  scanf("%d %d", &V, &E);
  scc.init(V);
  for (int i = 0; i < E; i++) {
    int u, v;
    scanf("%d %d", &u, &v);
    scc.add_edge(u-1,v-1);
  }
  printf("%d\n", scc.solve() );
  return 0;
}
```

## 7.4 CentroidDecomposition

```
vector<int> adj[N];
int p[N], vis[N];
int sz[N], M[N]; // subtree size of u and M(u)

inline void maxify(int &x, int y) { x = max(x, y); }
int centroidDecomp(int x) {
  vector<int> q;
  { // bfs
    size_t pt = 0;
    q.push_back(x);
    p[x] = -1;
    while (pt < q.size()) {
      int now = q[pt++];
      sz[now] = 1;
      M[now] = 0;
      for (auto &nxt : adj[now])
        if (!vis[nxt] && nxt != p[now])
          q.push_back(nxt), p[nxt] = now;
    }
  }

  // calculate subtree size in reverse order
  reverse(q.begin(), q.end());
  for (int &nd : q)
    if (p[nd] != -1) {
      sz[p[nd]] += sz[nd];
      maxify(M[p[nd]], sz[nd]);
    }
  for (int &nd : q)
    maxify(M[nd], (int)q.size() - sz[nd]);

  // find centroid
  int centroid = *min_element(q.begin(), q.end(),
                              [&](int x, int y) {
                                  return M[x] < M[y];
                                  });

  vis[centroid] = 1;
  for (auto &nxt : adj[centroid]) if (!vis[nxt])
    centroidDecomp(nxt);
  return centroid;
}
```

## 7.5 MinMeanCycle

```
/* minimum mean cycle O(VE) */
struct MMC{
#define E 101010
#define V 1021
#define inf 1e9
#define eps 1e-6
  struct Edge { int v,u; double c; };
  int n, m, prv[V][V], prve[V][V], vst[V];
  Edge e[E];
  vector<int> edgeID, cycle, rho;
  double d[V][V];
  void init( int _n )
  { n = _n; m = 0; }
  // WARNING: TYPE matters
  void addEdge( int vi , int ui , double ci )
  { e[ m ++ ] = { vi , ui , ci }; }
  void bellman_ford() {
    for(int i=0; i<n; i++) d[0][i]=0;
    for(int i=0; i<n; i++) {
      fill(d[i+1], d[i+1]+n, inf);
      for(int j=0; j<m; j++) {
        int v = e[j].v, u = e[j].u;
        if(d[i][v]<inf && d[i+1][u]>d[i][v]+e[j].c) {
          d[i+1][u] = d[i][v]+e[j].c;
          prv[i+1][u] = v;
          prve[i+1][u] = j;
        }
      }
    }
  }
  double solve(){
    // returns inf if no cycle, mmc otherwise
    double mmc=inf;
    int st = -1;
    bellman_ford();
    for(int i=0; i<n; i++) {
      double avg=-inf;
      for(int k=0; k<n; k++) {
        if(d[n][i]<inf-eps) avg=max(avg,(d[n][i]-d[k][i
            ])/(n-k));
        else avg=max(avg,inf);
      }
      if (avg < mmc) tie(mmc, st) = tie(avg, i);
    }
    FZ(vst); edgeID.clear(); cycle.clear(); rho.clear()
        ;
    for (int i=n; !vst[st]; st=prv[i--][st]) {
      vst[st]++;
      edgeID.PB(prve[i][st]);
      rho.PB(st);
    }
    while (vst[st] != 2) {
      int v = rho.back(); rho.pop_back();
      cycle.PB(v);
      vst[v]++;
    }
    reverse(ALL(edgeID));
    edgeID.resize(SZ(cycle));
    return mmc;
  }
} mmc;
```

## 7.6 BCC

```
struct BccVertex {
  int n,nScc,step,dfn[MXN],low[MXN];
  vector<int> E[MXN],sccv[MXN];
  int top,stk[MXN];
  void init(int _n) {
    n = _n; nScc = step = 0;
    for (int i=0; i<n; i++) E[i].clear();
  }
  void addEdge(int u, int v)
  { E[u].PB(v); E[v].PB(u); }
  void DFS(int u, int f) {
    dfn[u] = low[u] = step++;
    stk[top++] = u;
    for (auto v:E[u]) {
      if (v == f) continue;
      if (dfn[v] == -1) {
        DFS(v,u);
        low[u] = min(low[u], low[v]);
        if (low[v] >= dfn[u]) {
          int z;
          sccv[nScc].clear();
          do {
            z = stk[--top];
            sccv[nScc].PB(z);
          } while (z != v);
          sccv[nScc++].PB(u);
        }
      }else
        low[u] = min(low[u],dfn[v]);
    }
  }
  vector<vector<int>> solve() {
    vector<vector<int>> res;
    for (int i=0; i<n; i++)
      dfn[i] = low[i] = -1;
    for (int i=0; i<n; i++)
      if (dfn[i] == -1) {
        top = 0;
        DFS(i,i);
      }
    REP(i,nScc) res.PB(sccv[i]);
    return res;
  }
}graph;
```

## 7.7 DirectedGraphMinCycle

```
// works in O(N M)
#define INF 100000000000000LL
#define N 5010
#define M 200010
struct edge{
  int to; LL w;
  edge(int a=0, LL b=0): to(a), w(b){}
};
struct node{
  LL d; int u, next;
  node(LL a=0, int b=0, int c=0): d(a), u(b), next(c){}
}b[M];
struct DirectedGraphMinCycle{
  vector<edge> g[N], grev[N];
  LL dp[N][N], p[N], d[N], mu;
  bool inq[N];
  int n, bn, bsz, hd[N];
  void b_insert(LL d, int u){
    int i = d/mu;
    if(i >= bn) return;
    b[++bsz] = node(d, u, hd[i]);
    hd[i] = bsz;
  }
  void init( int _n ){
    n = _n;
    for( int i = 1 ; i <= n ; i ++ )
      g[ i ].clear();
  }
  void addEdge( int ai , int bi , LL ci )
  { g[ai].push_back(edge(bi,ci)); }
  LL solve(){
    fill(dp[0], dp[0]+n+1, 0);
    for(int i=1; i<=n; i++){
      fill(dp[i]+1, dp[i]+n+1, INF);
      for(int j=1; j<=n; j++) if(dp[i-1][j] < INF){
        for(int k=0; k<(int)g[j].size(); k++)
          dp[i][g[j][k].to] =min(dp[i][g[j][k].to],
                                 dp[i-1][j]+g[j][k].w);
      }
    }
    mu=INF; LL bunbo=1;
    for(int i=1; i<=n; i++) if(dp[n][i] < INF){
      LL a=-INF, b=1;
      for(int j=0; j<=n-1; j++) if(dp[j][i] < INF){
        if(a*(n-j) < b*(dp[n][i]-dp[j][i])){
          a = dp[n][i]-dp[j][i];
          b = n-j;
        }
      }
      if(mu*b > bunbo*a)
        mu = a, bunbo = b;
    }
    if(mu < 0) return -1; // negative cycle
    if(mu == INF) return INF; // no cycle
    if(mu == 0) return 0;
    for(int i=1; i<=n; i++)
      for(int j=0; j<(int)g[i].size(); j++)
        g[i][j].w *= bunbo;
    memset(p, 0, sizeof(p));
    queue<int> q;
    for(int i=1; i<=n; i++){
      q.push(i);
      inq[i] = true;
    }
    while(!q.empty()){
      int i=q.front(); q.pop(); inq[i]=false;
      for(int j=0; j<(int)g[i].size(); j++){
        if(p[g[i][j].to] > p[i]+g[i][j].w-mu){
          p[g[i][j].to] = p[i]+g[i][j].w-mu;
          if(!inq[g[i][j].to]){
            q.push(g[i][j].to);
            inq[g[i][j].to] = true;
          }
        }
      }
    }
    for(int i=1; i<=n; i++) grev[i].clear();
    for(int i=1; i<=n; i++)
      for(int j=0; j<(int)g[i].size(); j++){
        g[i][j].w += p[i]-p[g[i][j].to];
        grev[g[i][j].to].push_back(edge(i, g[i][j].w));
      }
    LL mldc = n*mu;
    for(int i=1; i<=n; i++){
      bn=mldc/mu, bsz=0;
      memset(hd, 0, sizeof(hd));
      fill(d+i+1, d+n+1, INF);
      b_insert(d[i]=0, i);
      for(int j=0; j<=bn-1; j++) for(int k=hd[j]; k; k=
          b[k].next){
        int u = b[k].u;
        LL du = b[k].d;
        if(du > d[u]) continue;
        for(int l=0; l<(int)g[u].size(); l++) if(g[u][l
            ].to > i){
          if(d[g[u][l].to] > du + g[u][l].w){
            d[g[u][l].to] = du + g[u][l].w;
            b_insert(d[g[u][l].to], g[u][l].to);
          }
        }
      }
      for(int j=0; j<(int)grev[i].size(); j++) if(grev[
          i][j].to > i)
        mldc=min(mldc,d[grev[i][j].to] + grev[i][j].w);
    }
    return mldc / bunbo;
  }
} graph;
```

## 7.8 DynamicMST

```
/* Dynamic MST O( Q lg^2 Q )
 (qx[i], qy[i])->chg weight of edge No.qx[i] to qy[i]
 delete an edge: (i, \infty)
 add an edge: change from \infty to specific value
 */
const int SZ=M+3*MXQ;
int a[N],*tz;
int find(int xx){
  int root=xx; while(a[root]) root=a[root];
  int next; while((next=a[xx])){a[xx]=root; xx=next; }
  return root;
}
bool cmp(int aa,int bb){ return tz[aa]<tz[bb]; }
int kx[N],ky[N],kt, vd[N],id[M], app[M];
bool extra[M];
void solve(int *qx,int *qy,int Q,int n,int *x,int *y,
    int *z,int m1,long long ans){
  if(Q==1){
    for(int i=1;i<=n;i++) a[i]=0;
    z[ qx[0] ]=qy[0]; tz = z;
    for(int i=0;i<m1;i++) id[i]=i;
    sort(id,id+m1,cmp); int ri,rj;
    for(int i=0;i<m1;i++){
      ri=find(x[id[i]]); rj=find(y[id[i]]);
      if(ri!=rj){ ans+=z[id[i]]; a[ri]=rj; }
    }
    printf("%lld\n",ans);
    return;
  }
  int ri,rj;
  //contract
  kt=0;
  for(int i=1;i<=n;i++) a[i]=0;
  for(int i=0;i<Q;i++){
    ri=find(x[qx[i]]); rj=find(y[qx[i]]); if(ri!=rj) a[
        ri]=rj;
  }
  int tm=0;
  for(int i=0;i<m1;i++) extra[i]=true;
  for(int i=0;i<Q;i++) extra[ qx[i] ]=false;
  for(int i=0;i<m1;i++) if(extra[i]) id[tm++]=i;
  tz=z; sort(id,id+tm,cmp);
  for(int i=0;i<tm;i++){
    ri=find(x[id[i]]); rj=find(y[id[i]]);
    if(ri!=rj){
      a[ri]=rj; ans += z[id[i]];
      kx[kt]=x[id[i]]; ky[kt]=y[id[i]]; kt++;
    }
  }
  for(int i=1;i<=n;i++) a[i]=0;
  for(int i=0;i<kt;i++) a[ find(kx[i]) ]=find(ky[i]);
  int n2=0;
  for(int i=1;i<=n;i++) if(a[i]==0)
  vd[i]=++n2;
  for(int i=1;i<=n;i++) if(a[i])
  vd[i]=vd[find(i)];
  int m2=0, *Nx=x+m1, *Ny=y+m1, *Nz=z+m1;
  for(int i=0;i<m1;i++) app[i]=-1;
  for(int i=0;i<Q;i++) if(app[qx[i]]==-1){
    Nx[m2]=vd[ x[ qx[i] ] ]; Ny[m2]=vd[ y[ qx[i] ] ];
        Nz[m2]=z[ qx[i] ];
    app[qx[i]]=m2; m2++;
  }
  for(int i=0;i<Q;i++){ z[ qx[i] ]=qy[i]; qx[i]=app[qx[
      i]]; }
  for(int i=1;i<=n2;i++) a[i]=0;
  for(int i=0;i<tm;i++){
    ri=find(vd[ x[id[i]] ]);  rj=find(vd[ y[id[i]] ]);
    if(ri!=rj){
      a[ri]=rj; Nx[m2]=vd[ x[id[i]] ];
      Ny[m2]=vd[ y[id[i]] ]; Nz[m2]=z[id[i]]; m2++;
    }
  }
  int mid=Q/2;
  solve(qx,qy,mid,n2,Nx,Ny,Nz,m2,ans);
  solve(qx+mid,qy+mid,Q-mid,n2,Nx,Ny,Nz,m2,ans);
}
int x[SZ],y[SZ],z[SZ],qx[MXQ],qy[MXQ],n,m,Q;
void init(){
  scanf("%d%d",&n,&m);
  for(int i=0;i<m;i++) scanf("%d%d%d",x+i,y+i,z+i);
  scanf("%d",&Q);
  for(int i=0;i<Q;i++){ scanf("%d%d",qx+i,qy+i); qx[i
      ]--; }
}
void work(){ if(Q) solve(qx,qy,Q,n,x,y,z,m,0); }
int main(){init(); work(); }
```

## 7.9 DynamicMST

```
struct WeightGraph {
  static const int INF = INT_MAX;
  static const int N = 514;
  struct edge{
    int u,v,w; edge(){}
    edge(int ui,int vi,int wi)
      :u(ui),v(vi),w(wi){}
  };
  int n,n_x;
  edge g[N*2][N*2];
  int lab[N*2];
  int match[N*2],slack[N*2],st[N*2],pa[N*2];
  int flo_from[N*2][N+1],S[N*2],vis[N*2];
  vector<int> flo[N*2];
  queue<int> q;
  int e_delta(const edge &e){
    return lab[e.u]+lab[e.v]-g[e.u][e.v].w*2;
  }
  void update_slack(int u,int x){
    if(!slack[x]||e_delta(g[u][x])<e_delta(g[slack[x]][
        x]))slack[x]=u;
  }
  void set_slack(int x){
    slack[x]=0;
    for(int u=1;u<=n;++u)
      if(g[u][x].w>0&&st[u]!=x&&S[st[u]]==0)
        update_slack(u,x);
  }
  void q_push(int x){
    if(x<=n)q.push(x);
    else for(size_t i=0;i<flo[x].size();i++)
    q_push(flo[x][i]);
}
void set_st(int x,int b){
  st[x]=b;
  if(x>n)for(size_t i=0;i<flo[x].size();++i)
    set_st(flo[x][i],b);
}
int get_pr(int b,int xr){
  int pr=find(flo[b].begin(),flo[b].end(),xr)-flo[b].
      begin();
  if(pr%2==1){
    reverse(flo[b].begin()+1,flo[b].end());
    return (int)flo[b].size()-pr;
  }else return pr;
}
void set_match(int u,int v){
  match[u]=g[u][v].v;
  if(u<=n) return;
  edge e=g[u][v];
  int xr=flo_from[u][e.u],pr=get_pr(u,xr);
  for(int i=0;i<pr;++i)set_match(flo[u][i],flo[u][i
      ^1]);
  set_match(xr,v);
  rotate(flo[u].begin(),flo[u].begin()+pr,flo[u].end
      ());
}
void augment(int u,int v){
  for(;;){
    int xnv=st[match[u]];
    set_match(u,v);
    if(!xnv)return;
    set_match(xnv,st[pa[xnv]]);
    u=st[pa[xnv]],v=xnv;
  }
}
int get_lca(int u,int v){
  static int t=0;
  for(++t;u||v;swap(u,v)){
    if(u==0)continue;
    if(vis[u]==t)return u;
    vis[u]=t;
    u=st[match[u]];
    if(u)u=st[pa[u]];
  }
  return 0;
}
void add_blossom(int u,int lca,int v){
  int b=n+1;
  while(b<=n_x&&st[b])++b;
  if(b>n_x)++n_x;
  lab[b]=0,S[b]=0;
  match[b]=match[lca];
  flo[b].clear();
  flo[b].push_back(lca);
  for(int x=u,y;x!=lca;x=st[pa[y]])
    flo[b].push_back(x),flo[b].push_back(y=st[match[x
        ]]),q_push(y);
  reverse(flo[b].begin()+1,flo[b].end());
  for(int x=v,y;x!=lca;x=st[pa[y]])
    flo[b].push_back(x),flo[b].push_back(y=st[match[x
        ]]),q_push(y);
  set_st(b,b);
  for(int x=1;x<=n_x;++x)g[b][x].w=g[x][b].w=0;
  for(int x=1;x<=n;++x)flo_from[b][x]=0;
  for(size_t i=0;i<flo[b].size();++i){
    int xs=flo[b][i];
    for(int x=1;x<=n_x;++x)
      if(g[b][x].w==0||e_delta(g[xs][x])<e_delta(g[b
          ][x]))
        g[b][x]=g[xs][x],g[x][b]=g[x][xs];
    for(int x=1;x<=n;++x)
      if(flo_from[xs][x])flo_from[b][x]=xs;
  }
  set_slack(b);
}
void expand_blossom(int b){
  for(size_t i=0;i<flo[b].size();++i)
    set_st(flo[b][i],flo[b][i]);
  int xr=flo_from[b][g[b][pa[b]].u],pr=get_pr(b,xr);
  for(int i=0;i<pr;i+=2){
    int xs=flo[b][i],xns=flo[b][i+1];
    pa[xs]=g[xns][xs].u;
    S[xs]=1,S[xns]=0;
    slack[xs]=0,set_slack(xns);
    q_push(xns);
  }
  S[xr]=1,pa[xr]=pa[b];
  for(size_t i=pr+1;i<flo[b].size();++i){
    int xs=flo[b][i];
    S[xs]=-1,set_slack(xs);
  }
  st[b]=0;
}
bool on_found_edge(const edge &e){
  int u=st[e.u],v=st[e.v];
  if(S[v]==-1){
    pa[v]=e.u,S[v]=1;
    int nu=st[match[v]];
    slack[v]=slack[nu]=0;
    S[nu]=0,q_push(nu);
  }else if(S[v]==0){
    int lca=get_lca(u,v);
    if(!lca)return augment(u,v),augment(v,u),true;
    else add_blossom(u,lca,v);
  }
  return false;
}
bool matching(){
  memset(S+1,-1,sizeof(int)*n_x);
  memset(slack+1,0,sizeof(int)*n_x);
  q=queue<int>();
  for(int x=1;x<=n_x;++x)
    if(st[x]==x&&!match[x])pa[x]=0,S[x]=0,q_push(x);
  if(q.empty())return false;
  for(;;){
    while(q.size()){
      int u=q.front();q.pop();
      if(S[st[u]]==1)continue;
      for(int v=1;v<=n;++v)
        if(g[u][v].w>0&&st[u]!=st[v]){
          if(e_delta(g[u][v])==0){
            if(on_found_edge(g[u][v]))return true;
          }else update_slack(u,st[v]);
        }
    }
    int d=INF;
    for(int b=n+1;b<=n_x;++b)
      if(st[b]==b&&S[b]==1)d=min(d,lab[b]/2);
    for(int x=1;x<=n_x;++x)
      if(st[x]==x&&slack[x]){
        if(S[x]==-1)d=min(d,e_delta(g[slack[x]][x]));
        else if(S[x]==0)d=min(d,e_delta(g[slack[x]][x
            ])/2);
      }
    for(int u=1;u<=n;++u){
      if(S[st[u]]==0){
        if(lab[u]<=d)return 0;
        lab[u]-=d;
      }else if(S[st[u]]==1)lab[u]+=d;
    }
    for(int b=n+1;b<=n_x;++b)
      if(st[b]==b){
        if(S[st[b]]==0)lab[b]+=d*2;
        else if(S[st[b]]==1)lab[b]-=d*2;
      }
    q=queue<int>();
    for(int x=1;x<=n_x;++x)
      if(st[x]==x&&slack[x]&&st[slack[x]]!=x&&e_delta
          (g[slack[x]][x])==0)
        if(on_found_edge(g[slack[x]][x]))return true;
    for(int b=n+1;b<=n_x;++b)
      if(st[b]==b&&S[b]==1&&lab[b]==0)expand_blossom(
          b);
  }
  return false;
}
pair<long long,int> solve(){
  memset(match+1,0,sizeof(int)*n);
  n_x=n;
  int n_matches=0;
  long long tot_weight=0;
  for(int u=0;u<=n;++u)st[u]=u,flo[u].clear();
  int w_max=0;
  for(int u=1;u<=n;++u)
```

```
      for(int v=1;v<=n;++v){
        flo_from[u][v]=(u==v?u:0);
        w_max=max(w_max,g[u][v].w);
      }
    for(int u=1;u<=n;++u)lab[u]=w_max;
    while(matching())++n_matches;
    for(int u=1;u<=n;++u)
      if(match[u]&&match[u]<u)
        tot_weight+=g[u][match[u]].w;
    return make_pair(tot_weight,n_matches);
  }
  void add_edge( int ui , int vi , int wi ){
    g[ui][vi].w = g[vi][ui].w = wi;
  }
  void init( int _n ){
    n = _n;
    for(int u=1;u<=n;++u)
      for(int v=1;v<=n;++v)
        g[u][v]=edge(u,v,0);
  }
} graph;
```

# 8 String

## 8.1 AC automaton

```
// SIGMA[0] will not be considered
const string SIGMA = "_0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
vector<int> INV_SIGMA;
const int SGSZ = 63;

struct PMA {
  PMA *next[SGSZ];  // next[0] is for fail
  vector<int> ac;
  PMA *last; // state of longest accepted string that is pre of this
  PMA() : last(nullptr) { fill(next, next + SGSZ, nullptr); }
};

template<typename T>
PMA *buildPMA(const vector<T> &p) {
  PMA *root = new PMA;
  for (int i = 0; i < p.size(); ++i) {  // make trie
    PMA *t = root;
    for (int j = 0; j < p[i].size(); ++j) {
      int c = INV_SIGMA[p[i][j]];
      if (t->next[c] == nullptr) t->next[c] = new PMA;
      t = t->next[c];
    }
    t->ac.push_back(i);
  }
  queue<PMA *> que;  // make failure link using bfs
  for (int c = 1; c < SGSZ; ++c) {
    if (root->next[c]) {
      root->next[c]->next[0] = root;
      que.push(root->next[c]);
    } else root->next[c] = root;
  }
  while (!que.empty()) {
    PMA *t = que.front();
    que.pop();
    for (int c = 1; c < SGSZ; ++c) {
      if (t->next[c]) {
        que.push(t->next[c]);
        PMA *r = t->next[0];
        while (!r->next[c]) r = r->next[0];
        t->next[c]->next[0] = r->next[c];
        t->next[c]->last = r->next[c]->ac.size() ? r->next[c] : r->next[c]->last;
      }
    }
  }
  return root;
}
```

```
void destructPMA(PMA *root) {
  queue<PMA *> que;
  que.emplace(root);
  while (!que.empty()) {
    PMA *t = que.front();
    que.pop();
    for (int c = 1; c < SGSZ; ++c) {
      if (t->next[c] && t->next[c] != root) que.emplace(t->next[c]);
    }
    delete t;
  }
}

template<typename T>
map<int, int> match(const T &t, PMA *v) {
  map<int, int> res;
  for (int i = 0; i < t.size(); ++i) {
    int c = INV_SIGMA[t[i]];
    while (!v->next[c]) v = v->next[0];
    v = v->next[c];
    for (int j = 0; j < v->ac.size(); ++j) ++res[v->ac[j]];
    for (PMA *q = v->last; q; q = q->last) {
      for (int j = 0; j < q->ac.size(); ++j) ++res[q->ac[j]];
    }
  }
  return res;
}
```

## 8.2 Gusfield

```
template<typename T>
vector<int> gusfield(const T &s) {
  vector<int> z(s.size(), s.size()); // z[i] := max k for z[0, k) = z[i, i + k)
  for (int i = 1, L = 0, R = 0; i < s.size(); ++i) {
    if (R < i) {
      L = R = i;
      while (R < s.size() && s[R] == s[R - L]) ++R;
      z[i] = R - L;
      --R;
    } else {
      int k = i - L;
      if (z[k] < R - i + 1) {
        z[i] = z[k];
      } else {
        L = i;
        while (R < s.size() && s[R] == s[R - L]) ++R;
        z[i] = R - L;
        --R;
      }
    }
  }
  return z;
}
```

## 8.3 KMP

```
template<typename T>
vector<int> build_kmp(const T &s) {
  vector<int> f(s.size());
  int fp = f[0] = -1;
  for (int i = 1; i < s.size(); ++i) {
    while (~fp && s[fp + 1] != s[i]) fp = f[fp];
    if (s[fp + 1] == s[i]) ++fp;
    f[i] = fp;
  }
  return f;
}
```

## 8.4 Manacher

```
template<typename T, int INF>
vector<int> manacher(const T &s) { // p = "INF" + s.
    join("INF") + "INF", returns radius on p
  vector<int> p(s.size() * 2 + 1, INF);
  for (int i = 0; i < s.size(); ++i) {
    p[i << 1 | 1] = s[i];
  }
  vector<int> w(p.size());
  for (int i = 1, j = 0, r = 0; i < p.size(); ++i) {
    int t = min(r >= i ? w[2 * j - i] : 0, r - i + 1);
    for ( ; i - t >= 0 && i + t < p.size(); ++t) {
      if (p[i - t] != p[i + t]) break;
    }
    w[i] = --t;
    if (i + t > r) r = i + t, j = i;
  }
  return w;
}
```

## 8.5 Suffix Array

```
// -----------O(NlgNlgN)----------
vector<int> sa_db(const string &s) {
  int n = s.size();
  vector<int> sa(n), r(n), t(n);
  for (int i = 0; i < n; ++i) r[sa[i] = i] = s[i];
  for (int h = 1; t[n - 1] != n - 1; h *= 2) {
    auto cmp = [&](int i, int j) {
      if (r[i] != r[j]) return r[i] < r[j];
      return i + h < n && j + h < n ? r[i + h] < r[j +
          h] : i > j;
    };
    sort(sa.begin(), sa.end(), cmp);
    for (int i = 0; i + 1 < n; ++i) t[i + 1] = t[i] +
        cmp(sa[i], sa[i + 1]);
    for (int i = 0; i < n; ++i) r[sa[i]] = t[i];
  }
  return sa;
}

// O(N) -- CF: 1e6->31ms,18MB;1e7->296ms;158MB;3e7->856
    ms,471MB
bool is_lms(const string &t, int i) {
  return i > 0 && t[i - 1] == 'L' && t[i] == 'S';
}

template<typename T>
vector<int> induced_sort(const T &s, const string &t,
    const vector<int> &lmss, int sigma = 256) {
  vector<int> sa(s.size(), -1);

  vector<int> bin(sigma + 1);
  for (auto it = s.begin(); it != s.end(); ++it) {
    ++bin[*it + 1];
  }

  int sum = 0;
  for (int i = 0; i < bin.size(); ++i) {
    sum += bin[i];
    bin[i] = sum;
  }

  vector<int> cnt(sigma);
  for (auto it = lmss.rbegin(); it != lmss.rend(); ++it
      ) {
    int ch = s[*it];
    sa[bin[ch + 1] - 1 - cnt[ch]] = *it;
    ++cnt[ch];
  }

  cnt = vector<int>(sigma);
  for (auto it = sa.begin(); it != sa.end(); ++it) {
    if (*it <= 0 || t[*it - 1] == 'S') continue;
    int ch = s[*it - 1];
    sa[bin[ch] + cnt[ch]] = *it - 1;
    ++cnt[ch];
  }

  cnt = vector<int>(sigma);
  for (auto it = sa.rbegin(); it != sa.rend(); ++it) {
    if (*it <= 0 || t[*it - 1] == 'L') continue;
    int ch = s[*it - 1];
    sa[bin[ch + 1] - 1 - cnt[ch]] = *it - 1;
    ++cnt[ch];
  }

  return sa;
}

template<typename T>
vector<int> sa_is(const T &s, int sigma = 256) {
  string t(s.size(), 0);
  t[s.size() - 1] = 'S';
  for (int i = int(s.size()) - 2; i >= 0; --i) {
    if (s[i] < s[i + 1]) t[i] = 'S';
    else if (s[i] > s[i + 1]) t[i] = 'L';
    else t[i] = t[i + 1];
  }

  vector<int> lmss;
  for (int i = 0; i < s.size(); ++i) {
    if (is_lms(t, i)) {
      lmss.emplace_back(i);
    }
  }

  vector<int> sa = induced_sort(s, t, lmss, sigma);
  vector<int> sa_lms;
  for (int i = 0; i < sa.size(); ++i) {
    if (is_lms(t, sa[i])) {
      sa_lms.emplace_back(sa[i]);
    }
  }

  int lmp_ctr = 0;
  vector<int> lmp(s.size(), -1);
  lmp[sa_lms[0]] = lmp_ctr;
  for (int i = 0; i + 1 < sa_lms.size(); ++i) {
    int diff = 0;
    for (int d = 0; d < sa.size(); ++d) {
      if (s[sa_lms[i] + d] != s[sa_lms[i + 1] + d] ||
          is_lms(t, sa_lms[i] + d) != is_lms(t, sa_lms[
              i + 1] + d)) {
        diff = 1; // something different in range of
            lms
        break;
      } else if (d > 0 && is_lms(t, sa_lms[i] + d) &&
          is_lms(t, sa_lms[i + 1] + d)) {
        break; // exactly the same
      }
    }
    if (diff) ++lmp_ctr;
    lmp[sa_lms[i + 1]] = lmp_ctr;
  }

  vector<int> lmp_compact;
  for (int i = 0; i < lmp.size(); ++i) {
    if (~lmp[i]) {
      lmp_compact.emplace_back(lmp[i]);
    }
  }

  if (lmp_ctr + 1 < lmp_compact.size()) {
    sa_lms = sa_is(lmp_compact, lmp_ctr + 1);
  } else {
    for (int i = 0; i < lmp_compact.size(); ++i) {
      sa_lms[lmp_compact[i]] = i;
    }
  }

  vector<int> seed;
  for (int i = 0; i < sa_lms.size(); ++i) {
    seed.emplace_back(lmss[sa_lms[i]]);
  }

  return induced_sort(s, t, seed, sigma);
} // s must end in char(0)

// O(N) lcp, note that s must end in '\0'
vector<int> build_lcp(const string &s, const vector<int
    > &sa, const vector<int> &rank) {
  int n = s.size();
  vector<int> lcp(n);
  for (int i = 0, h = 0; i < n; ++i) {
    if (rank[i] == 0) continue;
    int j = sa[rank[i] - 1];
    if (h > 0) --h;
    for ( ; j + h < n && i + h < n; ++h) {
      if (s[j + h] != s[i + h]) break;
    }
    lcp[rank[i] - 1] = h;
  }
  return lcp; // lcp[i] := lcp(s[sa[i]..-1], s[sa[i +
      1]..-1])
}

// O(N) build segment tree for lcp
vector<int> build_lcp_rmq(const vector<int> &lcp) {
  vector<int> sgt(lcp.size() << 2);
  function<void(int, int, int)> build = [&](int t, int
      lb, int rb) {
    if (rb - lb == 1) return sgt[t] = lcp[lb], void();
    int mb = lb + rb >> 1;
    build(t << 1, lb, mb);
    build(t << 1 | 1, mb, rb);
    sgt[t] = min(sgt[t << 1], sgt[t << 1 | 1]);
  };
  build(1, 0, lcp.size());
  return sgt;
}

// O(|P| + lg |T|) pattern searching, returns last
    index in sa
int match(const string &p, const string &s, const
    vector<int> &sa, const vector<int> &rmq) { // rmq
    is segtree on lcp
  int t = 1, lb = 0, rb = s.size(); // answer in [lb,
      rb)
  int lcplp = 0; // lcp(char(0), p) = 0
  while (rb - lb > 1) {
    int mb = lb + rb >> 1;
    int lcplm = rmq[t << 1];
    if (lcplp < lcplm) t = t << 1 | 1, lb = mb;
    else if (lcplp > lcplm) t = t << 1, rb = mb;
    else {
      int lcpmp = lcplp;
      while (lcpmp < p.size() && p[lcpmp] == s[sa[mb] +
           lcpmp]) ++lcpmp;
      if (lcpmp == p.size() || p[lcpmp] > s[sa[mb] +
          lcpmp]) t = t << 1 | 1, lb = mb, lcplp =
          lcpmp;
      else t = t << 1, rb = mb;
    }
  }
  if (lcplp < p.size()) return -1;
  return sa[lb];
}
```

## 8.6 Suffix Automaton

```
template<typename T>
struct SuffixAutomaton {
  vector<map<int, int>> edges;// edges[i]  : the
      labeled edges from node i
  vector<int> link;           // link[i]   : the parent
       of i
  vector<int> length;         // length[i] : the length
       of the longest string in the ith class
  int last;                   // the index of the
      equivalence class of the whole string
  vector<bool> is_terminal;   // is_terminal[i] : some
      suffix ends in node i (unnecessary)
  vector<int> occ;            // occ[i] : number of
      matches of maximum string of node i (unnecessary)
  SuffixAutomaton(const T &s) : edges({map<int, int>()
      }), link({-1}), length({0}), last(0), occ({0}) {
    for (int i = 0; i < s.size(); ++i) {
      edges.push_back(map<int, int>());
      length.push_back(i + 1);
      link.push_back(0);
      occ.push_back(1);
      int r = edges.size() - 1;
      int p = last; // add edges to r and find p with
          link to q
      while (p >= 0 && edges[p].find(s[i]) == edges[p].
          end()) {
        edges[p][s[i]] = r;
        p = link[p];
      }
      if (~p) {
        int q = edges[p][s[i]];
        if (length[p] + 1 == length[q]) { // no need to
             split q
          link[r] = q;
        } else { // split q, add qq
          edges.push_back(edges[q]);  // copy edges of
              q
          length.push_back(length[p] + 1);
          link.push_back(link[q]);  // copy parent of q
          occ.push_back(0);
          int qq = edges.size() - 1; // qq is new
              parent of q and r
          link[q] = qq;
          link[r] = qq;
          while (p >= 0 && edges[p][s[i]] == q) { //
              what points to q points to qq
            edges[p][s[i]] = qq;
            p = link[p];
          }
        }
      }
      last = r;
    } // below unnecessary
    is_terminal = vector<bool>(edges.size());
    for (int p = last; p > 0; p = link[p]) is_terminal[
        p] = 1; // is_terminal calculated
    vector<int> cnt(link.size()), states(link.size());
        // sorted states by length
    for (int i = 0; i < link.size(); ++i) ++cnt[length[
        i]];
    for (int i = 0; i < s.size(); ++i) cnt[i + 1] +=
        cnt[i];
    for (int i = link.size() - 1; i >= 0; --i) states
        [--cnt[length[i]]] = i;
    for (int i = link.size() - 1; i >= 1; --i) occ[link
        [states[i]]] += occ[states[i]]; // occ
        calculated
  }
};
```

# 9 Formulas

## 9.1 Pick's theorem

Pick's theorem provides a simple formula for calculating the area $A$ of this polygon in terms of the number $i$ of lattice points in the interior located in the polygon and the number $b$ of lattice points on the boundary placed on the polygon's perimeter:

$$A = i + \frac{b}{2} - 1$$

## 9.2 Graph Properties

1. Euler's Formula $V - E + F = 2$
2. For a planar graph, $F = E - V + n + 1$, n is the numbers of components
3. For a planar graph, $E \leq 3V - 6$ For a connected graph $G$, let $I(G)$ be the size of maximum independent set, $M(G)$ be the size of maximum matching, $Cv(G)$ be the size of minimum vertex cover, $Ce(G)$ be the size of minimum edge cover。
4. For any connected graph:

   (a) $I(G) + Cv(G) = |V|$
   (b) $M(G) + Ce(G) = |V|$

5. For any bipartite:

   (a) $I(G) = Cv(G)$
   (b) $M(G) = Ce(G)$

```
double l=0,=m,stop=1.0/n/n;
while(r-l>=stop){
  double(mid);
  if((n*m-sol.maxFlow(s,t))/2>eps)l=mid;
  else r=mid;
}
build(l);
sol.maxFlow(s,t);
vector<int> ans;
for(int i=1;i<=n;++i)
  if(sol.vis[i])ans.push_back(i);
```

## 9.3 Number Theory

1. $g(m) = \sum_{d|m} f(d) \Leftrightarrow f(m) = \sum_{d|m} \mu(d) \times g(m/d)$
2. $\phi(x), \mu(x)$ are Möbius inverse
3. $\sum_{i=1}^{n} \sum_{j=1}^{m} [\text{gcd}(i,j) = 1] = \sum \mu(d) \left\lfloor \frac{n}{d} \right\rfloor \left\lfloor \frac{m}{d} \right\rfloor$
4. $\sum_{i=1}^{n} \sum_{j=1}^{n} lcm(i,j) = n \sum_{d|n} d \times \phi(d)$

## 9.4 Combinatorics

1. Harmonic series $H_n = \ln(n) + \gamma + 1/(2n) - 1/(12n^2) + 1/(120n^4)$
2. $\gamma = 0.57721566490153286060651209008240243104215$
3. Gray Code: $= n \oplus (n >> 1)$
4. Catalan Number: $\frac{C_n^{kn}}{n(k-1)+1}$, $C_m^n = \frac{n!}{m!(n-m)!}$
5. $\gamma(n+1) = n!$
6. $H(n,m) \cong x_1 + x_2 \ldots + x_n = k, num = C_k^{n+k-1}$
7. $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$
8. Stirling number of $2^{nd}$ kind: $n$ 人分 $k$ 組方法數目
   (a) $S(0,0) = S(n,n) = 1$
   (b) $S(n,0) = 0$
   (c) $S(n,k) = kS(n-1,k) + S(n-1,k-1)$
9. Bell number,$n$ 人分任意多組方法數目
   (a) $B_0 = 1$
   (b) $B_n = \sum_{i=0}^{n} S(n,i)$
   (c) $B_{n+1} = \sum_{k=0}^{n} C_k^n B_k$
   (d) $B_{p+n} \equiv B_n + B_{n+1} mod p$, p is prime
   (e) $B_{p^m+n} \equiv mB_n + B_{n+1} mod p$, p is prime
   (f) From $B_0 : 1, 1, 2, 5, 15, 52,$
   $203, 877, 4140, 21147, 115975$
10. Derangement, 錯排, 🄵有人在自己位置上
   (a) $D_n = n!(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} \ldots + (-1)^n \frac{1}{n!})$
   (b) $D_n = (n-1)(D_{n-1} + D_{n-2}), D_0 = 1, D_1 = 0$
   (c) From $D_0 : 1, 0, 1, 2, 9, 44,$
   $265, 1854, 14833, 133496$
11. Binomial Equality
   (a) $\sum_k \binom{r}{m+k}\binom{s}{n-k} = \binom{r+s}{m+n}$
   (b) $\sum_k \binom{l}{m+k}\binom{s}{n+k} = \binom{l+s}{l-m+n}$
   (c) $\sum_k \binom{l}{m+k}\binom{s+k}{n}(-1)^k = (-1)^{l+m}\binom{s-m}{n-l}$
   (d) $\sum_{k \le l} \binom{l-k}{m}\binom{s}{k-n}(-1)^k = (-1)^{l+m}\binom{s-m-1}{l-n-m}$
   (e) $\sum_{0 \le k \le l} \binom{l-k}{m}\binom{q+k}{n} = \binom{l+q+1}{m+n+1}$
   (f) $\binom{r}{k} = (-1)^k \binom{k-r-1}{k}$
   (g) $\binom{r}{m}\binom{m}{k} = \binom{r}{k}\binom{r-k}{m-k}$
   (h) $\sum_{k \le n} \binom{r+k}{k} = \binom{r+n+1}{n}$
   (i) $\sum_{0 \le k \le n} \binom{k}{m} = \binom{n+1}{m+1}$
   (j) $\sum_{k \le m} \binom{m+r}{k} x^k y^k = \sum_{k \le m} \binom{-r}{k}(-x)^k (x+y)^{m-k}$

## 9.5 🄵次, 🄵次和

1. $a^b \% P = a^{b \% \varphi(p) + \varphi(p)}, b \geq \varphi(p)$
2. $1^3 + 2^3 + 3^3 + \ldots + n^3 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4}$
3. $1^4 + 2^4 + 3^4 + \ldots + n^4 = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$
4. $1^5 + 2^5 + 3^5 + \ldots + n^5 = \frac{n^6}{6} + \frac{n^5}{2} + \frac{5n^4}{12} - \frac{n^2}{12}$
5. $0^k + 1^k + 2^k + \ldots + n^k = P(k), P(k) = \frac{(n+1)^{k+1} - \sum_{i=0}^{k-1} C_i^{k+1} P(i)}{k+1}, P(0) = n+1$
6. $\sum_{k=0}^{m-1} k^n = \frac{1}{n+1} \sum_{k=0}^{n} C_k^{n+1} B_k m^{n+1-k}$
7. $\sum_{j=0}^{m} C_j^{m+1} B_j = 0, B_0 = 1$
8. 除了 $B_1 = -1/2$，剩下的奇數項都是 0
9. $B_2 = 1/6, B_4 = -1/30, B_6 = 1/42, B_8 = -1/30, B_{10} = 5/66, B_{12} = -691/2730, B_{14} = 7/6, B_{16} = -3617/510, B_{18} = 43867/798, B_{20} = -174611/330,$

## 9.6 Burnside's lemma

1. $|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$
2. $X^g = t^{c(g)}$
3. $G$ 表示有幾種轉法，$X^g$ 表示在那種轉法下，有幾種是會保持對稱的，$t$ 是🄵色數，$c(g)$ 是循環節不動的面數。
4. 正立方體塗三🄵色，轉 0 有 $3^6$ 個元素不變，轉 90 有 6 種，每種有 $3^3$ 不變，180 有 $3 \times 3^4$，120(角) 有 $8 \times 3^2$，180(邊) 有 $6 \times 3^3$，全部 $\frac{1}{24}\left(3^6 + 6 \times 3^3 + 3 \times 3^4 + 8 \times 3^2 + 6 \times 3^3\right) = 57$

## 9.7 Count on a tree

1. Rooted tree: $s_{n+1} = \frac{1}{n} \sum_{i=1}^{n} (i \times a_i \times \sum_{j=1}^{\lfloor n/i \rfloor} a_{n+1-i \times j})$
2. Unrooted tree:
   (a) Odd:$a_n - \sum_{i=1}^{n/2} a_i a_{n-i}$
   (b) Even:$Odd + \frac{1}{2} a_{n/2}(a_{n/2} + 1)$
3. Spanning Tree
   (a) 完全圖 $n^n - 2$
   (b) 一般圖 (Kirchhoff's theorem)$M[i][i] = degree(V_i)$,$M[i][j] = -1$,if have $E(i,j)$,0 if no edge. delete any one row and col in $A$, $ans = det(A)$