ACM ICPC Reference

University of Brasilia

July 10, 2018

Contents

.vimrc

```
set ai ts=4 sw=4 sta nu rnu sc stl+=%F
syntax on
```

.bashrc

```
alias cmp='g++ -Wall -Wshadow -Wconversion -fsanitize=
  address -std=c++11'
```

Estrutura de Dados

Merge Sort Tree

```
struct MergeTree{
    int n;
    vector<vector<int>> st;

    void build(int p, int L, int R, const int v[]){
        if(L == R){
            st[p].push_back(v[L]);
            return;
        }
        int mid = (L+R)/2;
        build(2*p, L, mid, v);
        build(2*p+1, mid+1, R, v);
        st[p].resize(R-L+1);
        merge(st[2*p].begin(), st[2*p].end(),
               st[2*p+1].begin(), st[2*p+1].end(),
               st[p].begin());
    }

    int query(int p, int L, int R, int i, int j, int x)
      const{
        if(L > j || R < i) return 0;
        if(L >= i && R <= j){
            int id = lower_bound(st[p].begin(), st[p].end
              (), x) - st[p].begin();
            return int(st[p].size()) - id;
        }
        int mid = (L+R)/2;
        return query(2*p, L, mid, i, j, x) +
            query(2*p+1, mid+1, R, i, j, x);
    }

public:
    MergeTree(int sz, const int v[]): n(sz), st(4*sz){
        build(1, 1, n, v);
    }

    //number of elements >= x on segment [i, j]
    int query(int i, int j, int x) const{
        if(i > j) swap(i, j);
        return query(1, 1, n, i, j, x);
    }
};
```

Ordered Set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

#include <ext/pb_ds/detail/standard_policies.hpp>

using namespace __gnu_pbds; // or pb_ds;

template<typename T, typename B = null_type>
using oset = tree<T, B, less<T>, rb_tree_tag,
  tree_order_statistics_node_update>;
// find_by_order / order_of_key
```

Convex Hull Trick

```
const ll is_query = -(1LL<<62);
struct Line{
    ll m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const{
        if (rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line>{ // will
  maintain upper hull for maximum
    bool bad(iterator y){
        auto z = next(y);
        if(y == begin()){
            if(z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if(z == end()) return y->m == x->m && y->b <= x->
          b;
        return (x->b - y->b)*(z->m - y->m) >= (y->b - z->
          b)*(y->m - x->m);
    }
    void insert_line(ll m, ll b){
        auto y = insert({ m, b });
        y->succ = [=]{ return next(y) == end() ? 0 : &*
          next(y); };
        if(bad(y)) { erase(y); return; }
        while(next(y) != end() && bad(next(y))) erase(
          next(y));
        while(y != begin() && bad(prev(y))) erase(prev(y)
          );
    }
    ll eval(ll x){
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

Min queue

```
class minQ{
    deque<tuple<int, int, int> > p;
    int delta, sz;
public:
    minQ() : delta(0), sz(0) {}

    inline int size() const{ return sz; }
    inline void add(int x){ delta += x; }
    inline void push(int x, int id){
        x -= delta, sz++;
        int t = 1;
        while(p.size() > 0 && get<0>(p.back()) >= x)
            t += get<1>(p.back()), p.pop_back();
        p.emplace_back(x, t, id);
    }
    inline void pop(){
        get<1>(p.front())--, sz--;
        if(!get<1>(p.front())) p.pop_front();
    }
    int getmin()const{ return get<0>(p.front())+delta; }
    int getid() const{ return get<2>(p.front()); }
};
```

Paradigmas

FFT

```
// typedef complex<double> base;
struct base{
    double r, i;
    base(double r_ = 0, double i_ = 0) : r(r_), i(i_) {}
    base operator*(base &o){
        return {r*o.r - i*o.i, r*o.i + o.r*i};
    }
    double real() const{ return r; }
    void operator*=(base &o){ r*o.r-i*o.i,r*o.i+o.r*i; }
    void operator+=(base &o){ r += o.r, i += o.i; }
    void operator/=(double &o){ r /= o, i /= o; }
    void operator-=(base &o){ r -= o.r, i -= o.i; }
    base operator+(base &o){ return {r+o.r, i+o.i}; }
    base operator-(base &o){ return {r-o.r, i-o.i}; }
};

double PI = acos(-1);

void fft(vector<base> &a, bool inv){
    int n = (int)a.size();

    for(int i = 1, j = 0; i < n; i++){
        int bit = n >> 1;
        for(; j >= bit; bit >>= 1) j -= bit;
        j += bit;
        if(i < j) swap(a[i], a[j]);
    }

    for(int len = 2; len <= n; len <<= 1) {
        double ang = 2*PI/len * (inv ? -1 : 1);
        base wlen(cos(ang), sin(ang));
        for(int i = 0; i < n; i += len){
            base w(1);
            for(int j = 0; j < len/2; j++){
                base u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }
    if(inv) for(int i = 0; i < n; i++) a[i] /= n;
}
void multiply(const vector<int> &a, const vector<int> &b
  , vector<int> &res){
    vector<base> fa(a.begin(), a.end());
    vector<base> fb(b.begin(), b.end());
    size_t n = 1;
    while(n < a.size()) n <<= 1;
    while(n < b.size()) n <<= 1;
    n <<= 1;
    fa.resize(n), fb.resize(n);

    fft(fa, false), fft(fb, false);
    for(size_t i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    res.resize (n);
    for(size_t i = 0; i < n; ++i)
        res[i] = int(fa[i].real() + 0.5);
}
```

NTT

```
const int mod = 7340033;
const int root = 5;
const int root_1 = 4404020;
const int root_pw = 1<<20;

void fft (vector<int> & a, bool invert) {
    int n = (int) a.size();

    for (int i=1, j=0; i<n; ++i) {
        int bit = n >> 1;
        for (; j>=bit; bit>>=1)
            j -= bit;
        j += bit;
        if (i < j)
            swap (a[i], a[j]);
    }

    for (int len=2; len<=n; len<<=1) {
        int wlen = invert ? root_1 : root;
        for (int i=len; i<root_pw; i<<=1)
            wlen = int (wlen * 1ll * wlen % mod);
        for (int i=0; i<n; i+=len) {
            int w = 1;
            for (int j=0; j<len/2; ++j) {
                int u = a[i+j], v = int (a[i+j+len/2] * 1
                  ll * w % mod);
                a[i+j] = u+v < mod ? u+v : u+v-mod;
                a[i+j+len/2] = u-v >= 0 ? u-v : u-v+mod;
                w = int (w * 1ll * wlen % mod);
            }
        }
    }
    if (invert) {
        int nrev = reverse (n, mod);
        for (int i=0; i<n; ++i)
            a[i] = int (a[i] * 1ll * nrev % mod);
    }
}
```

Matemática

Euclides Extendido

```
// a*x + b*y = gcd(a, b), <gcd, x, y>
tuple<int, int, int> euclidesExt(int a, int b) {
    if(b == 0) return make_tuple(a, 1, 0);
    int q, w, e;
    tie(q, w, e) = euclidesExt(b, a % b);
    return make_tuple(q, e, w - e * (a / b));
}
```

Preffix inverse

```
inv[1] = 1;
for(int i = 2; i < p; i++)
    inv[i] = (p - (p/i) * inv[p%i] % p) % p;
```

Pollard Rho

```
ll rho(ll n){
    if(n % 2 == 0) return 2;

    ll d, c = llrand() % n, x = llrand() % n, y = x;
    do{
        x = add(mul(x, x, n), c, n);
        y = add(mul(y, y, n), c, n);
        y = add(mul(y, y, n), c, n);
        d = __gcd(abs(x - y), n);
```

```
    }while(d == 1);
    return d;
}
```

Miller Rabin

```
bool rabin(ll n){
    if(n <= 1) return 0;
    if(n <= 3) return 1;
    ll s = 0, d = n - 1;
    while(d % 2 == 0) d /= 2, s++;
    for(int k = 0; k < 64; k++){
        ll a = (llrand() % (n - 3)) + 2;
        ll x = fexp(a, d, n);
        if(x != 1 && x != n-1){
            for(int r = 1; r < s; r++){
                x = mul(x, x, n);
                if(x == 1) return 0;
                if(x == n-1) break;
            }
            if(x != n-1) return 0;
        }
    }
    return 1;
}
```

Totiente

```
ll totiente(ll n){
    ll ans = n;
    for(ll i = 2; i*i <= n; i++){
        if(n % i == 0){
            ans = ans / i * (i - 1);
            while(n % i == 0) n /= i;
        }
    }

    if(n > 1) ans = ans / n * (n - 1);
    return ans;
}
```

Mulmod TOP

```
constexpr uint64_t mod = (1ull<<61) - 1;
uint64_t modmul(uint64_t a, uint64_t b){
    uint64_t l1 = (uint32_t)a, h1 = a>>32, l2 = (
      uint32_t)b, h2 = b>>32;
    uint64_t l = l1*l2, m = l1*h2 + l2*h1, h = h1*h2;
    uint64_t ret = (l&mod) + (l>>61) + (h << 3) + (m >>
      29) + (m << 35 >> 3) + 1;
    ret = (ret & mod) + (ret>>61);
    ret = (ret & mod) + (ret>>61);
    return ret-1;
}
```

Grafos

Dinic

```
const int N = 205;
const int E = 30006;
vector<int> g[N];

int ne;
struct Edge{
    int from, to;
    ll cap;
} edge[E];

int lvl[N], vis[N], pass, source, target, px[N];

ll run(int s, ll minE){
    if(s == target) return minE;

    ll ans = 0;

    for(; px[s] < g[s].size(); px[s]++){
        int e = g[s][ px[s] ];
        auto &v = edge[e], &rev = edge[e^1];
        if(lvl[v.to] != lvl[s]+1 || !v.cap) continue;
        ll tmp = run(v.to, min(minE, v.cap));
        v.cap -= tmp, rev.cap += tmp;
        ans += tmp, minE -= tmp;
        if(minE == 0) break;
    }
    return ans;
}

int bfs(){

    queue<int> q;
    q.push(source);
    lvl[source] = 1;
    vis[source] = ++pass;

    while(!q.empty()){
        int u = q.front(); q.pop();
        px[u] = 0;

        for(int e : g[u]){
            auto v = edge[e];
            if(v.cap <= 0 || vis[v.to] == pass) continue;
            vis[v.to] = pass;
            lvl[v.to] = lvl[u]+1;
            q.push(v.to);
        }
    }
    return vis[target] == pass;
}

ll flow(){
    ll ans = 0;
    while(bfs()) ans += run(source, oo);
    return ans;
}

void addEdge(int u, int v, ll c){
    edge[ne] = {u, v, c};
    g[u].pb(ne++);
}
```

Min Cost Max Flow

```
const ll oo = 1e18;
const int N = 505;
const int E = 30006;

vector<int> g[N];

int ne;

struct Edge{
    int from, to;
    ll cap, cost;
} edge[E];

int lvl[N], vis[N], pass, source, target, p[N], px[N];
```

```
ll d[N];

ll back(int s, ll minE){
    if(s == source) return minE;

    int e = p[s];

    ll f = back(edge[e].from, min(minE, edge[e].cap));
    edge[e].cap -= f;
    edge[e^1].cap += f;
    return f;
}

int dijkstra(){
    forn(i, N) d[i] = oo;

    priority_queue<pair<ll, int> > q;

    d[source] = 0;

    q.emplace(0, source);

    while(!q.empty()){
        ll dis = -q.top().ff;
        int u = q.top().ss; q.pop();

        if(dis > d[u]) continue;

        for(int e : g[u]){
            auto v = edge[e];
            if(v.cap <= 0) continue;
            if(d[u] + v.cost < d[v.to]){
                d[v.to] = d[u] + v.cost;
                p[v.to] = e;
                q.emplace(-d[v.to], v.to);
            }
        }
    }
    return d[target] != oo;
}

pair<ll, ll> mincost(){
    ll ans = 0, mf = 0;
    while(dijkstra()){
        ll f = back(target, oo);
        mf += f;
        ans += f * d[target];
    }
    return {mf, ans};
}

void addEdge(int u, int v, ll c, ll cost){
    edge[ne] = {u, v, c, cost};
    g[u].pb(ne++);
}
```

Small to Large

```
void cnt_sz(int u, int p = -1){
    sz[u] = 1;

    for(int v : g[u]) if(v != p)
        cnt_sz(v, u), sz[u] += sz[v];
}

void add(int u, int p, int big = -1){
    // Update info about this vx in global answer
    for(int v : g[u]) if(v != p && v != big)
        add(v, u);
}

void dfs(int u, int p, int keep){

    int big = -1, mmx = -1;

    for(int v : g[u]) if(v != p && sz[v] > mmx)
        mmx = sz[v], big = v;

    for(int v : g[u]) if(v != p && v != big)
        dfs(v, u, 0);

    if(big != -1) dfs(big, u, 1);

    add(u, p, big);

    for(auto x : q[u]){
        // answer all queries for this vx
    }

    if(!keep){
        // Remove data from this subtree
    }
}
```

Junior e Falta de Ideias

```
#include <bits/stdc++.h>

#define ff first
#define ss second
#define mp make_pair

using namespace std;

typedef long long ll;

vector<pair<int,int>> G[500005];
int subtree[500005], treesize, k;
bool vis[500005];
ll dist[500005], ans;

int dfs(int v, int p){
    subtree[v] = 1;
    for(pair<int,int> x : G[v])
        if(x.ff != p && !vis[x.ff]) subtree[v] += dfs(x.
          ff,v);
    return subtree[v];
}

int centroid(int v, int p){
    for(pair<int,int> x : G[v]){
        if(x.ff == p || vis[x.ff]) continue;
        if(subtree[x.ff]*2 > treesize) return centroid(x.
          ff,v);
    }
    return v;
}

void procurar_ans(int v, int p, int d_atual, ll custo){
    ans = min(ans, dist[k-d_atual] + custo);
    if(d_atual == k) return;
    for(pair<int,int> x : G[v]){
        if(!vis[x.ff] && x.ff != p)
            procurar_ans(x.ff,v,d_atual+1,custo+x.ss);
    }
```

```
}

void atualiza_distancia(int v, int p, int d_atual, ll
  custo){
   dist[d_atual] = min(dist[d_atual], custo);
   if(d_atual == k) return;
   for(pair<int,int> x : G[v]){
      if(!vis[x.ff] && x.ff != p)
         atualiza_distancia(x.ff,v,d_atual+1,custo+x.
           ss);
   }
}

void decomp(int v, int p){
   treesize = dfs(v,v);
   // if(treesize < k) return;
   int cent = centroid(v,v);
   vis[cent] = 1;

   for(int i = 1; i <= treesize; i++)
      dist[i] = 1e18;

   for(pair<int,int> x : G[cent]){
      if(!vis[x.ff]){
         procurar_ans(x.ff,cent,1,x.ss);
         atualiza_distancia(x.ff,cent,1,x.ss);
      }
   }

   for(pair<int,int> x : G[cent]){
      if(!vis[x.ff])
         decomp(x.ff, cent);
   }
}

int main(){
   int n,i,a,b;

   scanf("%d%d", &n,&k);
   for(i = 2; i <= n; i++){
      scanf("%d%d", &a,&b);
      G[i].push_back(mp(a,b));
      G[a].push_back(mp(i,b));
   }
   ans = 1e18;
   decomp(1,-1);

   printf("%lld\n", ans == 1e18 ? -1 : ans);

   return 0;
}
```

Kosaraju

```
vector<int> g[N], gt[N], S;

int vis[N], cor[N], tempo = 1;

void dfs(int u){
   vis[u] = 1;
   for(int v : g[u]) if(!vis[v]) dfs(v);
   S.push_back(u);
}
int e;
void dfst(int u){
   cor[u] = e;
   for(int v : gt[u]) if(!cor[v]) dfst(v);
}

int main(){

   for(int i = 1; i <= n; i++) if(!vis[i]) dfs(i);

   e = 0;
   reverse(S.begin(), S.end());
   for(int u : S) if(!cor[u])
      e++, dfst(u);

   return 0;
}
```

Tarjan

```
void dfs(int u, int p = -1){
   low[u] = num[u] = ++t;
   for(int v : g[u]){
      if(!num[v]){
         dfs(v, u);
         if(low[v] >= num[u]) u PONTO DE ARTICULACAO;
         if(low[v] > num[u]) ARESTA u->v PONTE;
         low[u] = min(low[u], low[v]);
      }
      else if(v != p) low[u] = min(low[u], num[v]);
   }
}
```

Max Clique

```
long long adj[N], dp[N];

for(int i = 0; i < n; i++){
   for(int j = 0; j < n; j++){
      int x;
      scanf("%d",&x);
      if(x || i == j)
         adj[i] |= 1LL << j;
   }
}

int resto = n - n/2;
int C = n/2;
for(int i = 1; i < (1 << resto); i++){
   int x = i;
   for(int j = 0; j < resto; j++)
      if(i & (1 << j))
         x &= adj[j + C] >> C;
   if(x == i){
      dp[i] = __builtin_popcount(i);
   }
}

for(int i = 1; i < (1 << resto); i++)
   for(int j = 0; j < resto; j++)
      if(i & (1 << j))
         dp[i] = max(dp[i], dp[i ^ (1 << j)]);

int maxCliq = 0;
for(int i = 0; i < (1 << C); i++){
   int x = i, y = (1 << resto) - 1;
   for(int j = 0; j < C; j++)
      if(i & (1 << j))
         x &= adj[j] & ((1 << C) - 1), y &= adj[j] >>
           C;
   if(x != i) continue;
   maxCliq = max(maxCliq, __builtin_popcount(i) + dp[y
     ]);
}
```

Strings

Aho Corasick

```
void init_aho(){
   queue<int> q;

   q.push(0);

   while(!q.empty()){
      int t = q.front(); q.pop();

      for(int i = 0; i < 52; i++) if(trie[t][i]){
         int x = trie[t][i];
         Q.push(x);

         if(t){
            fn[x] = fn[t];

            while(fn[x] && trie[fn[x]][i] == 0) fn[x]
              = fn[fn[x]];
            if(trie[fn[x]][i]) fn[x] = trie[fn[x]][i];
         }
      }
   }
}
```

Suffix Array

```
char s[N];
int n, sa[N], tsa[N], lcp[N], r[N], nr[N], c[N];

void sort(int k, int mx){
   mx++;
   memset(c, 0, sizeof(int) * mx);
   for(int i = 0; i < n; i++) c[i + k < n ? r[i+k]+1 :
     1]++;
   partial_sum(c, c+mx, c);
   int t;
   for(int i = 0; i < n; i++)
      t = sa[i]+k < n ? r[ sa[i]+k ] : 0,
      tsa[ c[t]++ ] = sa[i];
   memcpy(sa, tsa, sizeof(int) * n);
}

void build_sa(){

   for(int i = 0; i < n; i++) sa[i] = i, r[i] = s[i];

   int t = 300, a, b;
   for(int sz = 1; sz < n; sz *= 2){
      sort(sz, t), sort(0, t);
      t = nr[ sa[0] ] = 0;
      for(int i = 1; i < n; i++){
         a = sa[i]+sz < n ? r[ sa[i]+sz ] : -1;
         b = sa[i-1]+sz < n ? r[ sa[i-1]+sz ] : -1;
         nr[ sa[i] ] = r[ sa[i] ] == r[ sa[i-1] ] && a
            == b ? t : ++t;
      }
      if(t == n-1) break;
      memcpy(r, nr, sizeof(int) * n);
   }
}

void build_lcp(){ // lcp[i] = lcp(s[:i], s[:i+1])
   int k = 0;
   for(int i = 0; i < n; i++) r[ sa[i] ] = i;

   for(int i = 0; i < n; i++){
      if(r[i] == n-1) k = 0;
      else{
         int j = sa[r[i]+1];
         while(i+k < n && j+k < n && s[i+k] == s[j+k])
            k++;
      }
      lcp[r[i]] = k;
      if(k) k--;
   }
}
```

Z Algorithm

```
vector<int> z_algo(const string &s) {
   int n = s.size(), L = 0, R = 0;
   vector<int> z(n, 0);
   for(int i = 1; i < n; i++){
      if(i <= R) z[i] = min(z[i-L], R - i + 1);
      while(z[i]+i < n && s[ z[i]+i ] == s[ z[i] ])
         z[i]++;
      if(i+z[i]-1 > R) L = i, R = i + z[i] - 1;
   }
   return z;
}
```

Prefix function/KMP

```
vector<int> preffix_function(const string &s){
   int n = s.size();
   vector<int> b(n+1);
   b[0] = -1;
   int i = 0, j = -1;
   while(i < n){
      while(j >= 0 && s[i] != s[j]) j = b[j];
      b[++i] = ++j;
   }
   return b;
}

void kmp(const string &t, const string &p){
   vector<int> b = preffix_function(p);
   int n = t.size(), m = p.size();
   int j = 0;
   for(int i = 0; i < n; i++){
      while(j >= 0 && t[i] != p[j]) j = b[j];
      j++;
      if(j == m){
         //patern of p found on t
         j = b[j];
      }
   }
}
```

Min rotation

```
int min_rotation(int *s, int N) {
  REP(i, N) s[N+i] = s[i];

  int a = 0;
  REP(b, N) REP(i, N) {
    if (a+i == b || s[a+i] < s[b+i]) { b += max(0, i-1);
       break; }
    if (s[a+i] > s[b+i]) { a = b; break; }
  }
  return a;
}
```

All palindrome

```
void manacher(char *s, int N, int *rad) {
  static char t[2*MAX];
  int m = 2*N - 1;

  REP(i, m) t[i] = -1;
  REP(i, N) t[2*i] = s[i];

  int x = 0;
  FOR(i, 1, m) {
    int &r = rad[i] = 0;
    if (i <= x+rad[x]) r = min(rad[x+x-i], x+rad[x]-i);
    while (i-r-1 >= 0 && i+r+1 < m && t[i-r-1] == t[i+r
      +1]) ++r;
    if (i+r >= x+rad[x]) x = i;
  }

  REP(i, m) if (i-rad[i] == 0 || i+rad[i] == m-1) ++rad[
    i];
  REP(i, m) rad[i] /= 2;
}
```

Palindromic Tree

```
const int MAXN = 105000;

struct node {
    int next[26];
    int len;
    int sufflink;
    int num;
};

int len;
char s[MAXN];
node tree[MAXN];
int num; // node 1 - root with len -1, node 2 - root
  with len 0
int suff; // max suffix palindrome
long long ans;

bool addLetter(int pos) {
    int cur = suff, curlen = 0;
    int let = s[pos] - 'a';

    while(true){
        curlen = tree[cur].len;
        if (pos-1 - curlen >= 0 && s[pos-1 - curlen] == s
          [pos])
            break;
        cur = tree[cur].sufflink;
    }
    if (tree[cur].next[let]) {
        suff = tree[cur].next[let];
        return false;
    }

    num++;
    suff = num;
    tree[num].len = tree[cur].len + 2;
    tree[cur].next[let] = num;

    if (tree[num].len == 1){
        tree[num].sufflink = 2;
        tree[num].num = 1;
        return true;
    }

    while (true){
        cur = tree[cur].sufflink;
        curlen = tree[cur].len;
        if(pos-1 - curlen >= 0 && s[pos-1 - curlen] == s[
          pos]){
            tree[num].sufflink = tree[cur].next[let];
            break;
        }
    }

    tree[num].num = 1 + tree[tree[num].sufflink].num;

    return true;
}

void initTree() {
    num = 2; suff = 2;
    tree[1].len = -1; tree[1].sufflink = 1;
    tree[2].len = 0; tree[2].sufflink = 1;
}

int main() {

    initTree();

    for (int i = 0; i < len; i++) {
        addLetter(i);
    }

    return 0;
}
```

Geometria

2D basics

```
typedef double coord;
double eps = 1e-7;
bool eq(coord a, coord b){ return abs(a - b) <= eps; }

struct vec{
    coord x, y; int id;
    vec(coord a = 0, coord b = 0) : x(a), y(b) {}
    vec operator+(const vec &o) const{
        return {x + o.x, y + o.y};
    }
    vec operator-(const vec &o) const{
        return {x - o.x, y - o.y};
    }
    vec operator*(coord t) const{
        return {x * t, y * t};
    }
    vec operator/(coord t) const{
        return {x / t, y / t};
    }
    coord operator*(const vec &o) const{ // cos
        return x * o.x + y * o.y;
    }
    coord operator^(const vec &o) const{ // sin
        return x * o.y - y * o.x;
    }
    bool operator==(const vec &o) const{
        return eq(x, o.x) && eq(y, o.y);
    }
    bool operator<(const vec &o) const{
        if(!eq(x, o.x)) return x < o.x;
        return y < o.y;
```

```
    }
    coord cross(const vec &a, const vec &b) const{
        return (a-(*this)) ^ (b-(*this));
    }
    coord dot(const vec &a, const vec &b) const{
        return (a-(*this)) * (b-(*this));
    }
    coord len() const{
        return sqrt(x * x + y * y); // <
    }
    vec unit() const{
        return operator/(len());
    }
    int quad() const{
        if(x > 0 && y >=0) return 0;
        if(x <=0 && y > 0) return 1;
        if(x < 0 && y <=0) return 2;
        return 3;
    }
    bool comp(const vec &a, const vec &b) const{
        return (a - *this).comp(b - *this);
    }
    bool comp(vec b){
        if(quad() != b.quad()) return quad() < b.quad();
        if(!eq(operator^(b), 0)) return operator^(b) > 0;
        return (*this) * (*this) < b * b;
    }
    template<class T>
    void sort_by_angle(T first, T last) const{
        std::sort(first, last, [=](const vec &a, const
          vec &b){
           return comp(a, b);
        });
    }
    vec rot90() const{ return {-y, x}; }
    vec rot(double a) const{
        return {cos(a)*x - sin(a)*y, sin(a)*x + cos(a)*y
          };
    }
};

struct line{
    coord a, b, c; vec n;
    line(vec q, vec w){ // (w - q) ^ ((x, y) - q) = 0
        a = -(w.y-q.y);
        b = w.x-q.x;
        c = -(a * q.x + b * q.y);
        n = {a, b};
    }
    coord dist(const vec &o) const{
        return abs(eval(o)) / n.len();
    }
    bool contains(const vec &o) const{
        return eq(a * o.x + b * o.y + c, 0);
    }
    coord dist(const line &o) const{
        if(!parallel(o)) return 0;
        if(!eq(o.a * b, o.b * a)) return 0;
        if(!eq(a, 0))
            return abs(c - o.c * a / o.a) / n.len();
        if(!eq(b, 0))
            return abs(c - o.c * b / o.b) / n.len();
        return abs(c - o.c);
    }
    bool parallel(const line &o) const{
        return eq(n ^ o.n, 0);
    }
    bool operator==(const line &o) const{
        if(!eq(a*o.b, b*o.a)) return false;
        if(!eq(a*o.c, c*o.a)) return false;
        if(!eq(c*o.b, b*o.c)) return false;
        return true;
    }
    bool intersect(const line &o) const{
        return !parallel(o) || *this == o;
    }
    vec inter(const line &o) const{
        if(parallel(o)){
            if(*this == o){ }
            else{ /* dont intersect */ }
        }

        auto tmp = n ^ o.n;
        return {(o.c*b - c*o.b)/tmp, (o.a*c - a*o.c)/tmp
          };
    }
    vec at_x(coord x) const{
        return {x, (-c-a*x)/b};
    }
    vec at_y(coord y) const{
        return {(-c-b*y)/a, y};
    }
    coord eval(const vec &o) const{
        return a * o.x + b * o.y + c;
    }
};

struct segment{
    vec p, q;
    segment(vec a = vec(), vec b = vec()): p(a), q(b) {}
    bool onstrip(const vec &o) const{ // onstrip strip
        return p.dot(o, q) >= -eps && q.dot(o, p) >= -eps
          ;
    }
    coord len() const{
        return (p-q).len();
    }
    coord dist(const vec &o) const{
        if(onstrip(o)) return line(p, q).dist(o);
        return min((o-q).len(), (o-p).len());
    }
    bool contains(const vec &o) const{
        return eq(line(p, q).eval(o), 0) && onstrip(o);
    }
    bool intersect(const segment &o) const{
        auto d1 = p.cross(q, o.p);
        if(eq(d1, 0) && contains(o.p)) return true;
        auto d2 = p.cross(q, o.q);
        if(eq(d2, 0) && contains(o.q)) return true;
        auto d3 = o.p.cross(o.q, q);
        if(eq(d3, 0) && o.contains(q)) return true;
        auto d4 = o.p.cross(o.q, p);
        if(eq(d4, 0) && o.contains(p)) return true;
        return d1 * d2 < 0 && d3 * d4 < 0;
    }
    bool intersect(const line &o) const{
        return o.eval(p) * o.eval(q) <= 0;
    }
    coord dist(const segment &o) const{
        if(line(p, q).parallel(line(o.p, o.q))){
            if(onstrip(o.p) || onstrip(o.q)
            || o.onstrip(p) || o.onstrip(q))
                return line(p, q).dist(line(o.p, o.q));
        }
```

```
        else if(intersect(o)) return 0;
        return min(min(dist(o.p), dist(o.q)),
                min(o.dist(p), o.dist(q)));
    }
    coord dist(const line &o) const{
        if(line(p, q).parallel(o))
            return line(p, q).dist(o);
        else if(intersect(o)) return 0;
        return min(o.dist(p), o.dist(q));
    }
};

struct hray{
    vec p, q;
    hray(vec a = vec(), vec b = vec()): p(a), q(b){}
    bool onstrip(const vec &o) const{ // onstrip strip
        return p.dot(q, o) >= -eps;
    }
    coord dist(const vec &o) const{
        if(onstrip(o)) return line(p, q).dist(o);
        return (o-p).len();
    }
    bool intersect(const segment &o) const{
        if(!o.intersect(line(p,q))) return false;
        if(line(o.p, o.q).parallel(line(p,q)))
            return contains(o.p) || contains(o.q);
        return contains(line(p,q).inter(line(o.p,o.q)));
    }
    bool contains(const vec &o) const{
        return eq(line(p, q).eval(o), 0) && onstrip(o);
    }
    coord dist(const segment &o) const{
        if(line(p, q).parallel(line(o.p, o.q))){
            if(onstrip(o.p) || onstrip(o.q))
                return line(p, q).dist(line(o.p, o.q));
            return o.dist(p);
        }
        else if(intersect(o)) return 0;
        return min(min(dist(o.p), dist(o.q)),
                o.dist(p));
    }
    bool intersect(const hray &o) const{
        if(!line(p, q).parallel(line(o.p, o.q)))
            return false;
        auto pt = line(p, q).inter(line(o.p, o.q));
        return contains(pt) && o.contains(pt); // <<
    }
    bool intersect(const line &o) const{
        if(line(p, q).parallel(o)) return line(p, q)== o;
        if(o.contains(p) || o.contains(q)) return true;
        return (o.eval(p) >= -eps)^(o.eval(p)<o.eval(q));
        return contains(o.inter(line(p, q)));
    }
    coord dist(const line &o) const{
        if(line(p,q).parallel(o))
            return line(p,q).dist(o);
        else if(intersect(o)) return 0;
        return o.dist(p);
    }
    coord dist(const hray &o) const{
        if(line(p, q).parallel(line(o.p, o.q))){
            if(onstrip(o.p) || o.onstrip(p))
                return line(p,q).dist(line(o.p, o.q));
            return (p-o.p).len();
        }
        else if(intersect(o)) return 0;
        return min(dist(o.p), o.dist(p));
    }
};

double heron(coord a, coord b, coord c){
    coord s = (a + b + c) / 2;
    return sqrt(s * (s - a) * (s - b) * (s - c));
}
```

Nearest Points

```
struct pt {
    int x, y, id;
};

inline bool cmp_x (const pt & a, const pt & b) {
    return a.x < b.x || a.x == b.x && a.y < b.y;
}

inline bool cmp_y (const pt & a, const pt & b) {
    return a.y < b.y;
}

pt a[MAXN];

double mindist;
int ansa, ansb;

inline void upd_ans (const pt & a, const pt & b) {
    double dist = sqrt ((a.x-b.x)*(a.x-b.x) + (a.y-b.y)
      *(a.y-b.y) + .0);
    if (dist < mindist)
        mindist = dist, ansa = a.id, ansb = b.id;
}

void rec (int l, int r) {
    if (r - l <= 3) {
        for (int i=l; i<=r; ++i)
            for (int j=i+1; j<=r; ++j)
                upd_ans (a[i], a[j]);
        sort (a+l, a+r+1, &cmp_y);
        return;
    }

    int m = (l + r) >> 1;
    int midx = a[m].x;
    rec (l, m), rec (m+1, r);
    static pt t[MAXN];
    merge (a+l, a+m+1, a+m+1, a+r+1, t, &cmp_y);
    copy (t, t+r-l+1, a+l);

    int tsz = 0;
    for (int i=l; i<=r; ++i)
        if (abs (a[i].x - midx) < mindist) {
            for (int j=tsz-1; j>=0 && a[i].y - t[j].y <
              mindist; --j)
                upd_ans (a[i], t[j]);
            t[tsz++] = a[i];
        }
}

sort (a, a+n, &cmp_x);
mindist = 1E20;
rec (0, n-1);
```

Determinante

```
const double EPS = 1E-9;
int n;
vector < vector<double> > a (n, vector<double> (n));
```

```
double det = 1;
for (int i=0; i<n; ++i) {
    int k = i;
    for (int j=i+1; j<n; ++j)
        if (abs (a[j][i]) > abs (a[k][i]))
            k = j;
    if (abs (a[k][i]) < EPS) {
        det = 0;
        break;
    }
    swap (a[i], a[k]);
    if (i != k)
        det = -det;
    det *= a[i][i];
    for (int j=i+1; j<n; ++j)
        a[i][j] /= a[i][i];
    for (int j=0; j<n; ++j)
        if (j != i && abs (a[j][i]) > EPS)
            for (int k=i+1; k<n; ++k)
                a[j][k] -= a[i][k] * a[j][i];
}

cout << det;
```

Convex Hull

```
vector<vec> monotone_chain_ch(vector<vec> P){
    sort(P.begin(), P.end());

    vector<vec> L, U;
    for(auto p : P){
        while(L.size() >= 2 && L[L.size() - 2].cross(L.
          back(), p) < 0)
            L.pop_back();

        L.push_back(p);
    }

    reverse(P.begin(), P.end());
    for(auto p : P){
        while(U.size() >= 2 && U[U.size() - 2].cross(U.
          back(), p) < 0)
            U.pop_back();

        U.push_back(p);
    }

    L.pop_back(), U.pop_back();

    L.reserve(L.size() + U.size());
    L.insert(L.end(), U.begin(), U.end());

    return L;
}
```