Lib das Boyzinhas

Conteúdo

1 Basics

1.1 basic

```
 STL find() retorna iterador se achou, se nao achou
  retorna container.end()

Complexidade:
-> Set, map:  O(logN)
-> List, vector, deques, arrays: O(n)
-> Unordered maps without collisions: O(1)

 STL lower_bound() e upper_bound()
 Precisa sortar antes
 Lower retorna primeiro maior ou igual a target
 Upper retorna maior que o target
 Complexidade O(logN)
```

```
vector<int> container(1e3);
int target = 0;
auto it = find(container.begin(), container.end(), target);

auto it_lower = lower_bound(container.begin(), container.end()
    , target);
auto it_lower = upper_bound(container.begin(), container.end()
    , target);
```

---

2 Data Structures

2.1 SegTree

```
Code by SamuellH12
-> Segment Tree com:
  - Query em Range
  - Update em Ponto

build (1, 1, n, lista);
query (1, 1, n, a, b);
update(1, 1, n, i, x);

|   n    | tamanho
| [a, b] | intervalo da busca
|   i    | posicao a ser modificada
|   x    | novo valor da posicao i
| lista  | vector de elementos originais

Build:  O(N)
Query:  O(log N)
Update: O(log N)
```

```
const int MAXN = 1e6 + 5;
int seg[4*MAXN];

int query(int no, int l, int r, int a, int b){
  if(b <  l || r <  a) return 0;
  if(a <= l && r <= b) return seg[no];

  int m=(l+r)/2, e=no*2, d=no*2+1;

  return query(e, l, m, a, b) + query(d, m+1, r, a, b);
}

void update(int no, int l, int r, int pos, int v){
  if(pos < l || r < pos) return;
  if(l == r){seg[no] = v; return; }

  int m=(l+r)/2, e=no*2, d=no*2+1;

  update(e, l,   m, pos, v);
  update(d, m+1, r, pos, v);

  seg[no] = seg[e] + seg[d];
}

void build(int no, int l, int r, vector<int> &lista){
  if(l == r){ seg[no] = lista[l]; return; }

  int m=(l+r)/2, e=no*2, d=no*2+1;

  build(e, l,   m, lista);
  build(d, m+1, r, lista);

  seg[no] = seg[e] + seg[d];
}
```

2.2 SegTreeLazy

```
Code by SamuellH12
-> Segment Tree - Lazy Propagation com:
- Query em Range
- Update em Range

build (1, 1, n, lista);
query (1, 1, n, a, b);
update(1, 1, n, a, b, x);

|   n    | o tamanho maximo da lista
| [a, b] | o intervalo da busca ou update
|   x    | o novo valor a ser somada no intervalo [a, b]
| lista  | o array de elementos originais

Build:  O(N)
Query:  O(log N)
Update: O(log N)
Unlazy: O(1)
```

```
const int MAXN = 1e6 + 5;
int seg[4*MAXN];
int lazy[4*MAXN];

void unlazy(int no, int l, int r){
  if(lazy[no] == 0) return;

  int m=(l+r)/2, e=no*2, d=no*2+1;

  seg[no] += (r-l+1) * lazy[no];

  if(l != r){
    lazy[e] += lazy[no];
    lazy[d] += lazy[no];
  }

  lazy[no] = 0;
}

int query(int no, int l, int r, int a, int b){
  unlazy(no, l, r);
  if(b <  l || r <  a) return 0;
  if(a <= l && r <= b) return seg[no];

  int m=(l+r)/2, e=no*2, d=no*2+1;

  return query(e, l, m, a, b) + query(d, m+1, r, a, b);
}

void update(int no, int l, int r, int a, int b, int v){
  unlazy(no, l, r);
  if(b <  l || r <  a) return;
  if(a <= l && r <= b)
  {
    lazy[no]+= v;
    unlazy(no, l, r);
    return;
  }

  int m=(l+r)/2, e=no*2, d=no*2+1;

  update(e, l,   m, a, b, v);
  update(d, m+1, r, a, b, v);

  seg[no] = seg[e] + seg[d];
}

void build(int no, int l, int r, vector<int> &lista){
  if(l == r){ seg[no] = lista[l-1]; return; }

  int m=(l+r)/2, e=no*2, d=no*2+1;

  build(e, l,   m, lista);
  build(d, m+1, r, lista);

  seg[no] = seg[e] + seg[d];
}
```

2.3 BIT

```
 BIT - Fenwick Tree

 Complexidade:
 - Build: O(n)
 - Single Update: O(log n)
 - Query: O(log n)
```

```
struct BIT {
  vector<int> bit;
  int N;

  BIT(){}
```

```
    BIT(const vector<int>& a) {
        N = a.size();
        bit.assign(N + 1, 0);

        for (int i = 1; i <= N; ++i)
            bit[i] = a[i - 1];

        for (int i = 1; i <= N; ++i) {
            int j = i + (i & -i);
            if (j <= N)
                bit[j] += bit[i];
        }
    }

  void update(int pos, int val){
    for(; pos < N; pos += pos&(-pos))
      bit[pos] += val;
  }

  int query(int pos){
    int sum = 0;
    for(; pos > 0; pos -= pos&(-pos))
      sum += bit[pos];
    return sum;
  }
};
```

2.4 MergeSortTree

```
    MergeSort Tree

    Se for construida sobre um array:
      count(i, j, a, b) retorna quantos
      elementos de v[i..j] pertencem a [a, b]
      report(i, j, a, b) retorna os indices dos
      elementos de v[i..j] que pertencem a [a, b]
      retorna o vetor ordenado
    Se for construida sobre pontos (x, y):
        count(x1, x2, y1, y2) retorna quantos pontos
        pertencem ao retangulo (x1, y1), (x2, y2)
        report(x1, x2, y1, y2) retorna os indices dos pontos
     que
        pertencem ao retangulo (x1, y1), (x2, y2)
        retorna os pontos ordenados lexicograficamente
        (assume x1 <= x2, y1 <= y2)

    kth(y1, y2, k) retorna o indice do ponto com k-esimo
     menor x dentre os pontos que possuem y em [y1, y2] (0
     based)
    Se quiser usar para achar k-esimo valor em range,
     construir com ms_tree t(v, true), e chamar kth(l, r, k)

Usa O(n log(n)) de memoria
    Complexidades:
    construir - O(n log(n))
    count - O(log(n))
    report - O(log(n) + k) para k indices retornados
    kth - O(log(n))
```

```
template <typename T = int> struct ms_tree {
    vector<tuple<T, T, int>> v;
  int n;
  vector<vector<tuple<T, T, int>>> t; // {y, idx, left}
  vector<T> vy;

ms_tree(vector<pair<T, T>>& vv) : n(vv.size()), t(4*n), vy(n) {
      for (int i = 0; i < n; i++) v.push_back({vv[i].first, vv[i].second, i});
  sort(v.begin(), v.end());
  build(1, 0, n-1);
  for (int i = 0; i < n; i++) vy[i] = get<0>(t[1][i+1]);
}
ms_tree(vector<T>& vv, bool inv = false) { // inv: inverte indice e valor
  vector<pair<T, T>> v2;
  for (int i = 0; i < vv.size(); i++)
      inv ? v2.push_back({vv[i], i}) : v2.push_back({i, vv[i]});
  *this = ms_tree(v2);
}
void build(int p, int l, int r) {
  t[p].push_back({get<0>(v[l]), get<0>(v[r]), 0}); // {min_x, max_x, 0}
  if (l == r) return t[p].push_back({get<1>(v[l]), get<2>(v[l]), 0});
  int m = (l+r)/2;
  build(2*p, l, m), build(2*p+1, m+1, r);

  int L = 0, R = 0;
  while (t[p].size() <= r-l+1) {
    int left = get<2>(t[p].back());
    if (L > m-l or (R+m+1 <= r and t[2*p+1][1+R] < t[2*p][1+L])) {
      t[p].push_back(t[2*p+1][1 + R++]);
      get<2>(t[p].back()) = left;
      continue;
    }
    t[p].push_back(t[2*p][1 + L++]);
    get<2>(t[p].back()) = left+1;
  }
}

int get_l(T y) { return lower_bound(vy.begin(), vy.end(), y) - vy.begin(); }
int get_r(T y) { return upper_bound(vy.begin(), vy.end(), y) - vy.begin(); }

int count(T x1, T x2, T y1, T y2) {
      function<int(int, int, int)> dfs = [&](int p, int l, int r) {
        if (l == r or x2 < get<0>(t[p][0]) or get<1>(t[p][0]) < x1) return 0;
    if (x1 <= get<0>(t[p][0]) and get<1>(t[p][0]) <= x2)
        return r-l;
    int nl = get<2>(t[p][l]), nr = get<2>(t[p][r]);
    return dfs(2*p, nl, nr) + dfs(2*p+1, l-nl, r-nr);
  };
  return dfs(1, get_l(y1), get_r(y2));
}
vector<int> report(T x1, T x2, T y1, T y2) {
      vector<int> ret;
  function<void(int, int, int)> dfs = [&](int p, int l, int r) {
        if (l == r or x2 < get<0>(t[p][0]) or get<1>(t[p][0]) < x1) return;
    if (x1 <= get<0>(t[p][0]) and get<1>(t[p][0]) <= x2) {
              for (int i = l; i < r; i++) ret.push_back(get<1>(t[p][i+1]));
      return;
    }
    int nl = get<2>(t[p][l]), nr = get<2>(t[p][r]);
    dfs(2*p, nl, nr), dfs(2*p+1, l-nl, r-nr);
  };
    dfs(1, get_l(y1), get_r(y2));
    return ret;
  }
  int kth(T y1, T y2, int k) {
      function<int(int, int, int)> dfs = [&](int p, int l, int r) {
        if (k >= r-l) {
          k -= r-l;
      return -1;
    }
    if (r-l == 1) return get<1>(t[p][l+1]);
    int nl = get<2>(t[p][l]), nr = get<2>(t[p][r]);
    int left = dfs(2*p, nl, nr);
    if (left != -1) return left;
    return dfs(2*p+1, l-nl, r-nr);
  };
  return dfs(1, get_l(y1), get_r(y2));
  }
};
```

2.5 PrefixSum2D

```
Code by SamuellH12
Complexidade:
-> Calcular:  O(N^2)
-> Queries:   O(1)
```

```
const int MAXN = 1e3 + 5;
int ps [MAXN][MAXN];

void calcPS2d(){
  for (int i = 1; i < MAXN; i++) ps[0][i] += ps[0][i - 1];  // inicializo a la linha
  for (int i = 1; i < MAXN; i++) ps[i][0] += ps[i - 1][0];  // inicializo a la coluna

  for (int i = 1; i < MAXN; i++)
    for (int j = 1; j < MAXN; j++)
      ps[i][j] += ps[i - 1][j] + ps[i][j - 1] - ps[i - 1][j - 1];
}
int queryPS2d(int xi, int yi, int xf, int yf){ return ps[xf][yf] - ps[xf][yi-1] - ps[xi-1][yf] + ps[xi-1][yi-1]; }
```

2.6 Lichao

```
  Li-Chao Tree
  Adiciona retas (ax+b) e computa o minimo entre as retas em
    um dado x
  Cuidado com overflow, se tiver tenta comprimir o x ou usar
    convex hull trick

  O(log(MA - MI)) de tempo
  O(n) memoria
```

```
template<ll MI = ll(-1e9), ll MA = ll(1e9)> struct lichao {
  struct line {
    ll a, b;
    array<int, 2> ch;
    line(ll a_ = 0, ll b_ = LINF) :
```

```
    a(a_), b(b_), ch({-1, -1}) {}
  ll operator ()(ll x) { return a*x + b; }
};
vector<line> ln;

int ch(int p, int d) {
  if (ln[p].ch[d] == -1) {
    ln[p].ch[d] = ln.size();
    ln.emplace_back();
  }
  return ln[p].ch[d];
}
lichao() { ln.emplace_back(); }

void add(line s, ll l=MI, ll r=MA, int p=0) {
  ll m = (l+r)/2;
  bool L = s(l) < ln[p](l);
  bool M = s(m) < ln[p](m);
  bool R = s(r) < ln[p](r);
  if (M) swap(ln[p], s), swap(ln[p].ch, s.ch);
  if (s.b == LINF) return;
  if (L != M) add(s, l, m-1, ch(p, 0));
  else if (R != M) add(s, m+1, r, ch(p, 1));
}
ll query(int x, ll l=MI, ll r=MA, int p=0) {
  ll m = (l+r)/2, ret = ln[p](x);
  if (ret == LINF) return ret;
  if (x < m) return min(ret, query(x, l, m-1, ch(p, 0)));
  return min(ret, query(x, m+1, r, ch(p, 1)));
}
};
```

2.7 MergSortTree

```
MergeSort Tree

Se for construida sobre um array:
  count(i, j, a , b) retorna quantos elementos de v[i.. j]
   pertencem a [a,b]
  report(i, j, a, b) retorna os indices dos elementos de
   v[i..j] que pertencem a [a,b]
  retorna o vetor ordenado

Se for construida sobre pontos(x,y):
  count(x1, x2, y1, y2) retorna quantos pontos pertencem ao
   retangulo (x1,y1), (x2,y2)
  report(x1, x2, y1, y2) retorna os indices dos pontos que
   pertencem ao retangulo (x1,y1), (x2, y2)
  retorna os pontos ordenados lexicograficamente (assume x1
   <= x2, y1 <= y2)

kth(y1, y2, k) retorna o indice do ponto com k-esimo menor
   x dentro os pontos que possuem y em [y1,y2] (0 based)
se quiser usar para achar k-esimo valor em range, construir
   com ms_tree t(v,true), e chamar kth(l, r, k)

Vetor t {y, idx, left}
inv = true inverte indice e valor

O(n log(n)) Memoria

Build O(n (log n))
Count O(log(n))
Report O(log(n) + k) para k indices retornados
kth - O(log(n))
```

```
template <typename T = int> struct ms_tree {
  vector<tuple<T, T, int>> v;
  int n;
  vector<vector<tuple<T, T, int>>> t;
  vector<T> vy;

  ms_tree(vector<pair<T, T>>& vv) : n(vv.size()), t(4*n), vy(n
      ) {
    for (int i = 0; i < n; i++) v.push_back({vv[i].first, vv[i
        ].second, i});
    sort(v.begin(), v.end());
    build(1, 0, n-1);
    for (int i = 0; i < n; i++) vy[i] = get<0>(t[1][i+1]);
  }
  ms_tree(vector<T>& vv, bool inv = false) {
    vector<pair<T, T>> v2;
    for (int i = 0; i < vv.size(); i++)
      inv ? v2.push_back({vv[i], i}) : v2.push_back({i, vv[i
          ]});
    *this = ms_tree(v2);
  }
  void build(int p, int l, int r) {
    t[p].push_back({get<0>(v[l]), get<0>(v[r]), 0});
    if (l == r) return t[p].push_back({get<1>(v[l]), get<2>(v[
        l]), 0});
    int m = (l+r)/2;
    build(2*p, l, m), build(2*p+1, m+1, r);

    int L = 0, R = 0;
    while (t[p].size() <= r-l+1) {
      int left = get<2>(t[p].back());
      if (L > m-l or (R+m+1 <= r and t[2*p+1][1+R] < t[2*p][1+
          L])) {
        t[p].push_back(t[2*p+1][1 + R++]);
        get<2>(t[p].back()) = left;
        continue;
      }
      t[p].push_back(t[2*p][1 + L++]);
      get<2>(t[p].back()) = left+1;
    }
  }

  int get_l(T y) { return lower_bound(vy.begin(), vy.end(), y)
      - vy.begin(); }
  int get_r(T y) { return upper_bound(vy.begin(), vy.end(), y)
      - vy.begin(); }

  int count(T x1, T x2, T y1, T y2) {
    function<int(int, int, int)> dfs = [&](int p, int l, int r
        ) {
      if (l == r or x2 < get<0>(t[p][0]) or get<1>(t[p][0]) <
          x1) return 0;
      if (x1 <= get<0>(t[p][0]) and get<1>(t[p][0]) <= x2)
          return r-l;
      int nl = get<2>(t[p][l]), nr = get<2>(t[p][r]);
      return dfs(2*p, nl, nr) + dfs(2*p+1, l-nl, r-nr);
    };
    return dfs(1, get_l(y1), get_r(y2));
  }
  vector<int> report(T x1, T x2, T y1, T y2) {
    vector<int> ret;
    function<void(int, int, int)> dfs = [&](int p, int l, int
        r) {
      if (l == r or x2 < get<0>(t[p][0]) or get<1>(t[p][0]) <
          x1) return;
      if (x1 <= get<0>(t[p][0]) and get<1>(t[p][0]) <= x2) {
        for (int i = l; i < r; i++) ret.push_back(get<1>(t[p][
            i+1]));
        return;
      }
      int nl = get<2>(t[p][l]), nr = get<2>(t[p][r]);
      dfs(2*p, nl, nr), dfs(2*p+1, l-nl, r-nr);
    };
    dfs(1, get_l(y1), get_r(y2));
    return ret;
  }
  int kth(T y1, T y2, int k) {
    function<int(int, int, int)> dfs = [&](int p, int l, int r
        ) {
      if (k >= r-l) {
        k -= r-l;
        return -1;
      }
      if (r-l == 1) return get<1>(t[p][l+1]);
      int nl = get<2>(t[p][l]), nr = get<2>(t[p][r]);
      int left = dfs(2*p, nl, nr);
      if (left != -1) return left;
      return dfs(2*p+1, l-nl, r-nr);
    };
    return dfs(1, get_l(y1), get_r(y2));
  }
};
```

2.8 Cht

```
Convex Hull Trick Estatico
Adds tem que serem feitos em ordem de slope
Queries tem que ser feitas em ordem de x

Add O(1) amortizado
Get O(1) amortizado
```

```
// Convex Hull Trick Estatico
//
// adds tem que serem feitos em ordem de slope
// queries tem que ser feitas em ordem de x
//
// add O(1) amortizado, get O(1) amortizado

struct CHT {
  int it;
  vector<ll> a, b;
  CHT():it(0){}
  ll eval(int i, ll x){
    return a[i]*x + b[i];
  }
  bool useless(){
    int sz = a.size();
    int r = sz-1, m = sz-2, l = sz-3;
#warning cuidado com overflow!
    return (b[l] - b[r])*(a[m] - a[l]) <
      (b[l] - b[m])*(a[r] - a[l]);
  }
  void add(ll A, ll B){
    a.push_back(A); b.push_back(B);
    while (!a.empty()){
      if ((a.size() < 3) || !useless()) break;
      a.erase(a.end() - 2);
      b.erase(b.end() - 2);
    }
    it = min(it, int(a.size()) - 1);
  }
  ll get(ll x){
    while (it+1 < a.size()){
      if (eval(it+1, x) > eval(it, x)) it++;
```

```
      else break;
    }
    return eval(it, x);
  }
};
```

2.9 chtDinamico

```
Convex Hull Trick Dinamico
Para double, use LINF = 1/.0, div (a,b) = a/b

update(x) atualiza o ponto de intersecao da reta x
overlap(x) verifica se a reta x sobrepoe a proxima
add(a,b) adiciona reta da forma ax + b
query(x) computa maximo de ax + b para entre as retas

O(log(n)) amortizado por insercao
O(log(n)) por query

Cuidado com overflow
```

```
struct Line {
  mutable ll a, b, p;
  bool operator<(const Line& o) const { return a < o.a; }
  bool operator<(ll x) const { return p < x; }
};

struct dynamic_hull : multiset<Line, less<>> {
  ll div(ll a, ll b) {
    return a / b - ((a ^ b) < 0 and a % b);
  }

  void update(iterator x) {
    if (next(x) == end()) x->p = LINF;
    else if (x->a == next(x)->a) x->p = x->b >= next(x)->b ?
         LINF : -LINF;
    else x->p = div(next(x)->b - x->b, x->a - next(x)->a);
  }

  bool overlap(iterator x) {
    update(x);
    if (next(x) == end()) return 0;
    if (x->a == next(x)->a) return x->b >= next(x)->b;
    return x->p >= next(x)->p;
  }

  void add(ll a, ll b) {
    auto x = insert({a, b, 0});
    while (overlap(x)) erase(next(x)), update(x);
    if (x != begin() and !overlap(prev(x))) x = prev(x),
         update(x);
    while (x != begin() and overlap(prev(x)))
      x = prev(x), erase(next(x)), update(x);
  }

  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);

    return l.a * x + l.b;
  }
};
```

3 Geometry

3.1 Geometry - General

```
PONTO & VETOR
 th e radianos
 angle calcula o angulo do vetor com o eixo x
 sarea calcula area com sinal
 col se p, q e r sao colin.
 ccw e counter-clockwise (antihorario)
 rotaciona 90 graus


 RETA
 isvert - se e vertical
 isinseg - ponto pertence ao segmento
 get_t - ponto intersecao
 proj - projecao cartesiana
 inter - intercecao de dois segmentos
 interseg - se dois segmentos se interceptam
 distseg - distancia entre dois segmentos

 POLIGONO

 cut_polygon -> corta poligono com a reta r O(n)
 dist_rect -> distancia entre os retangulos a e b (lados
  paralelos aos eixos), assume que ta representado
  (inferior esquerdo, superior direito)
 pol_area -> area do poligono
 inpol -> O(n) retorna 0 se ta fora,1 se ta no interior e
  2 se ta na borda
 interpol -> se dois poligonos se intersectam - O(n*m)
 distpol -> distancia entre poligonos
 convex hull - O(n log(n)) nao pode ter ponto colinear no
  convex hull
 is_inside -> se o ponto ta dentro do hull - O(log(n))
 extreme -> ponto extremo em relacao a cmp(p, q) = p mais
  extremo q
 copiado de https://github.com/gustavoM32/caderno-zika


 CIRCUNFERENCIA
 getcenter -> centro da circunf dado 3 pontos
 circ_line_inter -> intersecao da circunf (c, r) e reta ab
 circ_inter -> intersecao da circunf (a, r) e (b, R),
 assume que as retas tem p < q
 operator< e == comparador pro set pra fazer sweep line
  com segmentos
 assume que os segmentos tem p < q , comparador pro set
  pra fazer sweep angle com segmentos
```

```
typedef double ld;
const ld DINF = 1e18;
const ld pi = acos(-1.0);
const ld eps = 1e-9;

#define sq(x) ((x)*(x))

bool eq(ld a, ld b) {
  return abs(a - b) <= eps;
}

struct pt {
    ld x, y;
  pt(ld x_ = 0, ld y_ = 0) : x(x_), y(y_) {}
  bool operator < (const pt p) const {
      if (!eq(x, p.x)) return x < p.x;
    if (!eq(y, p.y)) return y < p.y;
    return 0;
  }
  bool operator == (const pt p) const {
      return eq(x, p.x) and eq(y, p.y);
  }
  pt operator + (const pt p) const { return pt(x+p.x, y+p.y);
     }
  pt operator - (const pt p) const { return pt(x-p.x, y-p.y);
     }
  pt operator * (const ld c) const { return pt(x*c  , y*c  );
     }
  pt operator / (const ld c) const { return pt(x/c  , y/c  );
     }
  ld operator * (const pt p) const { return x*p.x + y*p.y; }
  ld operator ^ (const pt p) const { return x*p.y - y*p.x; }
  friend istream& operator >> (istream& in, pt& p) {
      return in >> p.x >> p.y;
  }
};

struct line {
    pt p, q;
  line() {}
  line(pt p_, pt q_) : p(p_), q(q_) {}
  friend istream& operator >> (istream& in, line& r) {
      return in >> r.p >> r.q;
  }
};


ld dist(pt p, pt q) {
    return hypot(p.y - q.y, p.x - q.x);
}

ld dist2(pt p, pt q) {
    return sq(p.x - q.x) + sq(p.y - q.y);
}

ld norm(pt v) {
    return dist(pt(0, 0), v);
}

ld angle(pt v) {
    ld ang = atan2(v.y, v.x);
  if (ang < 0) ang += 2*pi;
  return ang;
}

ld sarea(pt p, pt q, pt r) {
    return ((q-p)^(r-q))/2;
}

bool col(pt p, pt q, pt r) {
    return eq(sarea(p, q, r), 0);
}

bool ccw(pt p, pt q, pt r) {
    return sarea(p, q, r) > eps;
}

pt rotate(pt p, ld th) {
    return pt(p.x * cos(th) - p.y * sin(th),
    p.x * sin(th) + p.y * cos(th));
}

pt rotate90(pt p) {
    return pt(-p.y, p.x);
}
```

```
bool isvert(line r) { // se r eh vertical
  return eq(r.p.x, r.q.x);
}

bool isinseg(pt p, line r) {
    pt a = r.p - p, b = r.q - p;
  return eq((a ^ b), 0) and (a * b) < eps;
}

ld get_t(pt v, line r) {
    return (r.p^r.q) / ((r.p-r.q)^v);
}

pt proj(pt p, line r) {
    if (r.p == r.q) return r.p;
  r.q = r.q - r.p; p = p - r.p;
  pt proj = r.q * ((p*r.q) / (r.q*r.q));
  return proj + r.p;
}

pt inter(line r, line s) {
    if (eq((r.p - r.q) ^ (s.p - s.q), 0)) return pt(DINF, DINF
        );
  r.q = r.q - r.p, s.p = s.p - r.p, s.q = s.q - r.p;
  return r.q * get_t(r.q, s) + r.p;
}

bool interseg(line r, line s) {
    if (isinseg(r.p, s) or isinseg(r.q, s)
    or isinseg(s.p, r) or isinseg(s.q, r)) return 1;

  return ccw(r.p, r.q, s.p) != ccw(r.p, r.q, s.q) and
    ccw(s.p, s.q, r.p) != ccw(s.p, s.q, r.q);
}

ld disttoline(pt p, line r) {
    return 2 * abs(sarea(p, r.p, r.q)) / dist(r.p, r.q);
}

ld disttoseg(pt p, line r) {
    if ((r.q - r.p)*(p - r.p) < 0) return dist(r.p, p);
  if ((r.p - r.q)*(p - r.q) < 0) return dist(r.q, p);
  return disttoline(p, r);
}

ld distseg(line a, line b) {
    if (interseg(a, b)) return 0;

  ld ret = DINF;
  ret = min(ret, disttoseg(a.p, b));
  ret = min(ret, disttoseg(a.q, b));
  ret = min(ret, disttoseg(b.p, a));
  ret = min(ret, disttoseg(b.q, a));

  return ret;
}
vector<pt> cut_polygon(vector<pt> v, line r) {
    vector<pt> ret;
  for (int j = 0; j < v.size(); j++) {
        if (ccw(r.p, r.q, v[j])) ret.push_back(v[j]);
    if (v.size() == 1) continue;
    line s(v[j], v[(j+1)%v.size()]);
    pt p = inter(r, s);
    if (isinseg(p, s)) ret.push_back(p);
  }
  ret.erase(unique(ret.begin(), ret.end()), ret.end());
  if (ret.size() > 1 and ret.back() == ret[0]) ret.pop_back();
  return ret;
}

ld dist_rect(pair<pt, pt> a, pair<pt, pt> b) {
    ld hor = 0, vert = 0;
  if (a.second.x < b.first.x) hor = b.first.x - a.second.x;
  else if (b.second.x < a.first.x) hor = a.first.x - b.second.
      x;
  if (a.second.y < b.first.y) vert = b.first.y - a.second.y;
  else if (b.second.y < a.first.y) vert = a.first.y - b.second
      .y;
  return dist(pt(0, 0), pt(hor, vert));
}

ld polarea(vector<pt> v) {
    ld ret = 0;
  for (int i = 0; i < v.size(); i++)
    ret += sarea(pt(0, 0), v[i], v[(i + 1) % v.size()]);
  return abs(ret);
}

int inpol(vector<pt>& v, pt p) {
    int qt = 0;
  for (int i = 0; i < v.size(); i++) {
        if (p == v[i]) return 2;
    int j = (i+1)%v.size();
    if (eq(p.y, v[i].y) and eq(p.y, v[j].y)) {
            if ((v[i]-p)*(v[j]-p) < eps) return 2;
      continue;
    }
    bool baixo = v[i].y+eps < p.y;
    if (baixo == (v[j].y+eps < p.y)) continue;
    auto t = (p-v[i])^(v[j]-v[i]);
    if (eq(t, 0)) return 2;
    if (baixo == (t > eps)) qt += baixo ? 1 : -1;
  }
  return qt != 0;
}

bool interpol(vector<pt> v1, vector<pt> v2) {
    int n = v1.size(), m = v2.size();
  for (int i = 0; i < n; i++) if (inpol(v2, v1[i])) return 1;
  for (int i = 0; i < n; i++) if (inpol(v1, v2[i])) return 1;
  for (int i = 0; i < n; i++) for (int j = 0; j < m; j++)
    if (interseg(line(v1[i], v1[(i+1)%n]), line(v2[j], v2[(j
        +1)%m]))) return 1;
  return 0;
}

ld distpol(vector<pt> v1, vector<pt> v2) {
    if (interpol(v1, v2)) return 0;

  ld ret = DINF;

  for (int i = 0; i < v1.size(); i++) for (int j = 0; j < v2.
      size(); j++)
    ret = min(ret, distseg(line(v1[i], v1[(i + 1) % v1.size()
        ]),
    line(v2[j], v2[(j + 1) % v2.size()])));
  return ret;
}

vector<pt> convex_hull(vector<pt> v) {
    sort(v.begin(), v.end());
  v.erase(unique(v.begin(), v.end()), v.end());
  if (v.size() <= 1) return v;
  vector<pt> l, u;
  for (int i = 0; i < v.size(); i++) {
        while (l.size() > 1 and !ccw(l.end()[-2], l.end()[-1],
            v[i]))
        l.pop_back();
        l.push_back(v[i]);
    }
    for (int i = v.size() - 1; i >= 0; i--) {
        while (u.size() > 1 and !ccw(u.end()[-2], u.end()[-1],
            v[i]))
        u.pop_back();
        u.push_back(v[i]);
    }
    l.pop_back(); u.pop_back();
    for (pt i : u) l.push_back(i);
    return l;
}

struct convex_pol {
    vector<pt> pol;

    convex_pol() {}
    convex_pol(vector<pt> v) : pol(convex_hull(v)) {}

    bool is_inside(pt p) {
        if (pol.size() == 0) return false;
        if (pol.size() == 1) return p == pol[0];
        int l = 1, r = pol.size();
        while (l < r) {
            int m = (l+r)/2;
            if (ccw(p, pol[0], pol[m])) l = m+1;
            else r = m;
        }
        if (l == 1) return isinseg(p, line(pol[0], pol[1]));
        if (l == pol.size()) return false;
        return !ccw(p, pol[l], pol[l-1]);
    }

    int extreme(const function<bool(pt, pt)>& cmp) {
        int n = pol.size();
        auto extr = [&](int i, bool& cur_dir) {
            cur_dir = cmp(pol[(i+1)%n], pol[i]);
            return !cur_dir and !cmp(pol[(i+n-1)%n], pol[i]);
        };
        bool last_dir, cur_dir;
        if (extr(0, last_dir)) return 0;
        int l = 0, r = n;
        while (l+1 < r) {
            int m = (l+r)/2;
            if (extr(m, cur_dir)) return m;
            bool rel_dir = cmp(pol[m], pol[l]);
            if ((!last_dir and cur_dir) or
            (last_dir == cur_dir and rel_dir == cur_dir)) {
                l = m;
                last_dir = cur_dir;
            } else r = m;
        }
        return l;
    }
    int max_dot(pt v) {
        return extreme([&](pt p, pt q) { return p*v > q*v; });
  }
  pair<int, int> tangents(pt p) {
        auto L = [&](pt q, pt r) { return ccw(p, r, q); };
    auto R = [&](pt q, pt r) { return ccw(p, q, r); };
    return {extreme(L), extreme(R)};
  }
};


pt getcenter(pt a, pt b, pt c) {
    b = (a + b) / 2;
```

```
  c = (a + c) / 2;
  return inter(line(b, b + rotate90(a - b)),
    line(c, c + rotate90(a - c)));
}

vector<pt> circ_line_inter(pt a, pt b, pt c, ld r) {
    vector<pt> ret;
  b = b-a, a = a-c;
  ld A = b*b;
  ld B = a*b;
  ld C = a*a - r*r;
  ld D = B*B - A*C;
  if (D < -eps) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+eps))/A);
  if (D > eps) ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}

vector<pt> circ_inter(pt a, pt b, ld r, ld R) {
    vector<pt> ret;
  ld d = dist(a, b);
  if (d > r+R or d+min(r, R) < max(r, R)) return ret;
  ld x = (d*d-R*R+r*r)/(2*d);
  ld y = sqrt(r*r-x*x);
  pt v = (b-a)/d;
  ret.push_back(a+v*x + rotate90(v)*y);
  if (y > 0) ret.push_back(a+v*x - rotate90(v)*y);
  return ret;
}

bool operator <(const line& a, const line& b) {
    pt v1 = a.q - a.p, v2 = b.q - b.p;
  if (!eq(angle(v1), angle(v2))) return angle(v1) < angle(v2);
  return ccw(a.p, a.q, b.p);
}
bool operator ==(const line& a, const line& b) {
    return !(a < b) and !(b < a);
}

struct cmp_sweepline {
    bool operator () (const line& a, const line& b) const {
        if (a.p == b.p) return ccw(a.p, a.q, b.q);
    if (!eq(a.p.x, a.q.x) and (eq(b.p.x, b.q.x) or a.p.x+eps <
        b.p.x))
        return ccw(a.p, a.q, b.p);
    return ccw(a.p, b.q, b.p);
  }
};

pt dir;
struct cmp_sweepangle {
    bool operator () (const line& a, const line& b) const {
        return get_t(dir, a) + eps < get_t(dir, b);
  }
};
```

3.2 ClosestPair

```
    Closest pairs - Par de pontos que tem a menor distancia
     Euclidiana entre si
    O(n log n)
```

```
pii ClosestPair(vector<PT<ll>>& pts) {
    ll dist = (pts[0]-pts[1]).dist2();
    pii ans(0, 1);
    int n = pts.size();
    vector<int> p(n);
    iota(begin(p),end(p),0);
    sort(p.begin(), p.end(), [&](int a, int b) { return pts[a
        ].x < pts[b].x; });
    set<pii> points;
    auto sqr = [](long long x) -> long long { return x * x; };
    for(int l = 0, r = 0; r < n; r++) {
        while(sqr(pts[p[r]].x - pts[p[l]].x) > dist) {
        points.erase(pii(pts[p[l]].y, p[l]));
        l++;
        }
        ll delta = sqrt(dist) + 1;
        auto itl = points.lower_bound(pii(pts[p[r]].y - delta,
            -1));
        auto itr = points.upper_bound(pii(pts[p[r]].y + delta,
            n + 1));
        for(auto it = itl; it != itr; it++) {
        ll curDist = (pts[p[r]] - pts[it->second]).dist2();
        if(curDist < dist) {
            dist = curDist;
            ans = pii(p[r], it->second);
        }
        }
        points.insert(pii(pts[p[r]].y, p[r]));
    }
    if(ans.first > ans.second)
        swap(ans.first, ans.second);
    return ans;
}
```

3.3 Ch/

```
Given a vector of points, return the convex hull in
CCW order.
A convex hull is the smallest convex polygon that
contains all the points.

If you want colinear points in border, change the >=0 to >0
    in the while's.
WARNING:if collinear and all input PT are collinear, may have
    duplicated points (the round trip)
```

```
vector<PT> ConvexHull(vector<PT> pts, bool sorted=false){
  if(!sorted) sort(begin(pts), end(pts));
  pts.resize(unique(begin(pts), end(pts)) - begin(pts));
  if(pts.size() <= 1) return pts;

  int s=0, n=pts.size();
  vector<PT> h(2*n+1);

  for(int i=0; i<n; h[s++] = pts[i++])
    while(s > 1 && (pts[i] - h[s-2]) % (h[s-1] - h[s-2]) >= 0
        )
      s--;

  for(int i=n-2, t=s; ~i; h[s++] = pts[i--])
    while(s > t && (pts[i] - h[s-2]) % (h[s-1] - h[s-2]) >= 0
        )
      s--;

  h.resize(s-1);
  return h;
} //PT operators needed: {- % == <}
```

```
/*BLOCK_DESC_BEGIN Check **if a point is inside convex hull**
    (CCW, no collinear).
If strict == true, then pt on boundary return false
O(log N)
BLOCK_DESC_END*/
bool isInside(const vector<PT>& h, PT p, bool strict = true){
  int a = 1, b = h.size() - 1, r = !strict;
  if(h.size() < 3) return r && onSegment(h[0], h.back(), p);
  if(h[0].cross(h[a], h[b]) > 0) swap(a, b);
  if(h[0].cross(h[a], p) >= r || h[0].cross(h[b], p) <= -r)
      return false;
  while(abs(a-b) > 1){
    int c = (a + b) / 2;
    if(h[0].cross(h[c], p) > 0) b = c;
    else a = c;
  }
  return h[a].cross(h[b], p) < r;
}

/*BLOCK_DESC_BEGIN Check **if a point is inside convex hull**
    \\ O(log N) BLOCK_DESC_END*/
bool isInside(const vector<PT> &h, PT p){
  if(h[0].cross(p, h[1]) > 0 || h[0].cross(p, h.back()) < 0)
      return false;
    int n = h.size(), l=1, r = n-1;
    while(l != r){
        int mid = (l+r+1)/2;
        if(h[0].cross(p, h[mid]) < 0) l = mid;
        else r = mid - 1;
    }
    return h[l].cross(h[(l+1)%n], p) >= 0;
}


/*BLOCK_DESC_BEGIN
Given a convex hull h and a point p, returns the indice of h
    **where the dot product is maximized**.
This code assumes that there are NO 3 colinear points!
BLOCK_DESC_END*/
int maximizeScalarProduct(const vector<PT> &h, PT v) {
    int ans = 0, n = h.size();
    if(n < 20){
    for(int i=0; i<n; i++)
            if(v*h[ans] < v*h[i])
                ans = i;
    return ans;
    }

  for(int rep=0; rep<2; rep++){
    int l = 2, r = n-1;
    while(l != r){
      int mid = (l+r+1)/2;
      int f = v*h[mid] >= v*h[mid-1];

      if(rep) f |= v*h[mid-1] < v*h[0];
      else    f &= v*h[mid]  >= v*h[0];

      if(f) l = mid;
      else  r = mid - 1;
    }
    if(v*h[ans] < v*h[l]) ans = l;
  }
  if(v*h[ans] < v*h[1] ) ans = 1;
  return ans;
}
```

3.4 Point

```
len() -> O(sqrt(p*p))
cross() ->  (a-p) % (b-p)
quad() -> Cartesian plane quadrant |0++|1-+|2--|3+-|
proj() -> projection size from A to B
angle( ) -> Angle between vectors p and q [-pi, pi] |
   acos(a*b/a.len()/b.len())
polarAngle() -> Angle to x-axis [-pi, pi]
```

```
struct PT {
  ll x, y;
  PT(ll x=0, ll y=0) : x(x), y(y) {}

  PT operator+(const PT&a)const{return PT(x+a.x, y+a.y);}
  PT operator-(const PT&a)const{return PT(x-a.x, y-a.y);}
  ll operator*(const PT&a)const{return  (x*a.x + y*a.y);}
  ll operator%(const PT&a)const{return  (x*a.y - y*a.x);}
  PT operator*(ll c) const{ return PT(x*c, y*c); }
  PT operator/(ll c) const{ return PT(x/c, y/c); }
  bool operator==(const PT&a) const{ return x == a.x && y == a
      .y; }
  bool operator< (const PT&a) const{ return x != a.x ? x < a.x
      : y < a.y; }

  ld len() const { return hypot(x,y); }
  ll cross(const PT&a, const PT&b) const{ return (a-*this) % (
      b-*this); }
  int quad() { return (x<0)^3*(y<0); }
  bool ccw(PT q, PT r){ return (q-*this) % (r-q) > 0;}
};

ld dist(PT p, PT q){ return sqrtl((p-q)*(p-q)); }
ld proj(PT p, PT q){ return p*q / q.len(); }


const ld PI = acos(-1.0L);
ld angle(PT p, PT q){ return atan2(p%q, p*q); }
ld polarAngle(PT p){  return atan2(p.y, p.x); }
bool cmp_ang(PT p, PT q){ return p.quad() != q.quad() ? p.quad
    ()<q.quad() : q.ccw(PT(0,0), p); }

PT rotateCCW90(PT p){ return PT(-p.y, p.x); }
PT rotateCW90(PT p){ return PT(p.y, -p.x); }
PT rotateCCW(PT p, ld t){
  ld c = cos(t), s = sin(t);
  return PT(p.x*c - p.y*s, p.x*s + p.y*c);
}
```

3.5 Poligons

```
#

/*BLOCK_DESC_BEGIN
Returns **twice area of a simple polygon**. area*2 (Shoelace
    Formula: signed cross product sum)
BLOCK_DESC_END*/
ll Area2x(vector<PT>& p){
  ll area = 0;
  for(int i=2; i < p.size(); i++)
    area += (p[i]-p[0]) % (p[i-1]-p[0]);
  return abs(area);
}

/*BLOCK_DESC_BEGIN
Returns if a point is **inside a triangle** (or in the border)
    .
BLOCK_DESC_END*/
bool ptInsideTriangle(PT p, PT a, PT b, PT c){
  if((b-a) % (c-b) < 0) swap(a, b);
  if(onSegment(a,b,p)) return 1;
  if(onSegment(b,c,p)) return 1;
  if(onSegment(c,a,p)) return 1;
  bool x = (b-a) % (p-b) < 0;
  bool y = (c-b) % (p-c) < 0;
  bool z = (a-c) % (p-a) < 0;
  return x == y && y == z;
}


/*BLOCK_DESC_BEGIN
Returns the **center of mass for a polygon**. O(n)
BLOCK_DESC_END*/
PT polygonCenter(const vector<PT>& v){
  PT res(0, 0); double A = 0;
  for(int i=0, j=v.size()-1; i<v.size(); j=i++){
    res = res + (v[i]+v[j]) * (v[j]%v[i]);
    A += v[j] % v[i];
  }
  return res / A / 3;
}


/*BLOCK_DESC_BEGIN
\begin{minipage}{0.85\textwidth}
**PolygonCut**: Returns the vertices of the polygon cut away
    at the left of the line s->e.
polygonCut(p, PT(0,0), PT(1,0));
\end{minipage}\hfill \begin{minipage}{0.15\textwidth} \
    includegraphics[height=4\baselineskip]{geometry/PolygonCut
    } \end{minipage} BLOCK_DESC_END*/

vector<PT> polygonCut(const vector<PT>& poly, PT s, PT e){
  vector<PT> res;
  for(int i=0; i<poly.size(); i++){
    PT cur = poly[i], prev = i ? poly[i-1] : poly.back();
    auto a = s.cross(e, cur), b = s.cross(e, prev);
    if((a < 0) != (b < 0)) res.push_back(cur + (prev - cur) *
        (a / (a - b)));
    if(a < 0) res.push_back(cur);
  }
  return res;
}

/*BLOCK_DESC_BEGIN
Pick's theorem for **lattice points** in a simple polygon. (
    lattice points = integer points)
Area = insidePts + boundPts/2 - 1
2A - b + 2 = 2i
BLOCK_DESC_END*/
ll cntInsidePts(ll area_db, ll bound){ return (area_db + 2LL -
    bound)/2; }
ll latticePointsInSeg(PT a, PT b){
  ll dx = abs(a.x - b.x);
  ll dy = abs(a.y - b.y);
  return gcd(dx, dy) + 1;
}
```

3.6 Segment

```
bool onSegment(PT s, PT e, PT p){
  return p.cross(s, e) == 0 && (s-p) * (e-p) <= 0;
}

/*BLOCK_DESC_BEGIN \begin{minipage}{0.85\textwidth}
Returns the shortest **distance** between point p and the **
    segment** s->e.

\end{minipage}\hfill \begin{minipage}{0.15\textwidth} \
    includegraphics[height=4\baselineskip]{geometry/
    SegmentDistance} \end{minipage}
BLOCK_DESC_END*/

ld segmentDist(PT& s, PT& e, PT& p){
  if (s==e) return (p-s).len();
  ld d = (e-s)*(e-s);
  ld t = min(d, max<ld>(0, (p-s)*(e-s)));
  return ((p-s)*d - (e-s)*t).len() / d;
}

/*BLOCK_DESC_BEGIN
**Segment intersection** \\
Unique   -> \{p\}         \\
No inter -> \{ \}         \\
Infinity -> \{a, b\}, the endpoints of the common segment.
May be rounded if inter isn't integer; Watch out for overflow
    if long long. BLOCK_DESC_END*/

int sgn(ll x){ return (x>0) - (x<0); }
vector<PT> segInter(PT a, PT b, PT c, PT d){
  auto oa = c.cross(d, a), ob = c.cross(d, b);
  auto oc = a.cross(b, c), od = a.cross(b, d);
  if(sgn(oa)*sgn(ob) < 0 && sgn(oc)*sgn(od) < 0)
    return {(a*ob - b*oa) / (ob-oa)};
  set<PT> s;
  if(onSegment(c, d, a)) s.insert(a);
  if(onSegment(c, d, b)) s.insert(b);
  if(onSegment(a, b, c)) s.insert(c);
  if(onSegment(a, b, d)) s.insert(d);
  return {begin(s), end(s)};
}
```

4 Graphs

4.1 Dijkstra

```
Dijkstra - Shortest Paths from Source
// !!! Change MAXN to N
caminho minimo de um vertice u para todos os
outros vertices de um grafo ponderado

Complexity: O(N Log N)

dijkstra(s)       ->  s : Source, Origem. As distancias serao
    calculadas com base no vertice s
grafo[u] = {v, c};    ->  u : Vertice inicial, v : Vertice
    final, c : Custo da aresta
priority_queue<pii, vector<pii>, greater<pii>> ->  Ordena
    pelo menor custo -> {d, v} -> d : Distancia, v : Vertice
```

```
const int MAXN = 1e6 + 5;
#define INF 0x3f3f3f3f
#define vi vector<int>

vector<pii> grafo [MAXN];

vi dijkstra(int s){
```

```
    vi dist (MAXN, INF);
    priority_queue<pii, vector<pii>, greater<pii>> fila;
  fila.push({0, s});
  dist[s] = 0;

  while(!fila.empty())
  {
        auto [d, u] = fila.top();
    fila.pop();

    if(d > dist[u]) continue;

    for(auto [v, c] : grafo[u])
      if( dist[v] > dist[u] + c )
      {
        dist[v] = dist[u] + c;
        fila.push({dist[v], v});
      }
  }

  return dist;
}
```

4.2 BFS

```
int grid[1000][1000];
int dx[] = {0, 1, -1, 0};
int dy[] = {1, 0, 0, -1};

bool visitados[1000][1000];

void BFS(int x, int y){

    queue<pair<int,int>> q;
    q.push({x,y});

    visitados[x][y] = true;

    while(q.size()){
        auto [x1, y1] = q.front();
        q.pop();

        for(int i = 0; i < 4; i++){
            int ax = x1 + dx[i];
            int ay = y1 + dy[i];

            if(!visitados[ax][ay]){
                visitados[ax][ay] = true;
                q.push({ax, ay});
            }
        }
    }
}
```

4.3 DFS

```
int grid[1000][1000];
int dx[] = {0, 1, -1, 0};
int dy[] = {1, 0, 0, -1};

bool visitados[1000][1000];

void dfs(int x, int y){
    visitados[x][y] = true;

    for(int i = 0; i < 4; i++){
        int ax = x + dx[i];
        int ay = y + dy[i];

        if(!visitados[ax][ay]) dfs(ax,ay);
    }
}
```

4.4 DSU

```
Disjoint Set Union - Union Find
Find: O( a(n) ) -> Inverse Ackermann function
Join: O( a(n) ) -> a(1e6) <= 5
```

```
struct DSU {
  vector<int> pai, sz;
  DSU(int n) : pai(n+1), sz(n+1, 1) {
    for(int i=0; i<=n; i++) pai[i] = i;
  }

  int find(int u){ return pai[u] == u ? u : pai[u] = find(pai[
      u]); }

  void join(int u, int v){
    u = find(u), v = find(v);

    if(u == v) return;
    if(sz[v] > sz[u]) swap(u, v);

    pai[v] = u;
    sz[u] += sz[v];
  }
};
```

4.5 BellManFord

```
Disjoint Set Union - Union Find
Find: O( a(n) ) -> Inverse Ackermann function
Join: O( a(n) ) -> a(1e6) <= 5
```

```
int n, m;
int d[MAX];
vector<pair<int, int>> edges;
vector<int> weights;

bool bellman_ford(int a) {
  for (int i = 0; i < n; i++) d[i] = INF;
  d[a] = 0;

  for (int i = 0; i <= n; i++)
    for (int j = 0; j < m; j++) {
      if (d[edges[j].second] > d[edges[j].first] + weights[j])
          {
        if (i == n) return 1;

        d[edges[j].second] = d[edges[j].first] + weights[j];
      }
    }

  return 0;
}

/*LATEX_DESC_BEGIN***************************

  Bellman-Ford
  Calcula a menor dist ncia entre (a) e todos os outros
      v rtices
  Detecta ciclo negativo
  Retorna 1 se h  ciclo negativo
  N o precisa representar o grafo
  S  armazenar as arestas

  O(n * m)
*****************************LATEX_DESC_END*/
```

4.6 Floyd-Warshall

```
  Floyd-Warshall
  Encontra o menor caminho entre todo par de vertices e
     detecta ciclo negativo
  Returna 1 se ha ciclo negativo
  d[i][i] deve ser 0
  para i != j, d[i][j] deve ser w se ha uma aresta (i, j) de
     w, INF caso contrario
  O(n  )
```

```
int n;
int d[MAX][MAX];

bool floyd_warshall() {
  for (int k = 0; k < n; k++)
  for (int i = 0; i < n; i++)
  for (int j = 0; j < n; j++)
    d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

  for (int i = 0; i < n; i++)
    if (d[i][i] < 0) return 1;

  return 0;
}
```

4.7 EulerPath

```
  Euler Path / Euler Cycle
  Para declarar: euler<true> E(n) se for direcionado com N
     vertices
  As funcoes retornam um par
    - Booleano indica se ha o path/cycle pedido
    - Vetor e formada de {vertice, id aresta para chegar no
     vertice}

  Se for get_path, na primeira posicao o id vai ser -1
  get_path(src) tenta achar um caminho ou ciclo euleriano,
     comecando no src
#warning chamar para o src certo!
  Se achar um ciclo, o primeiro e o ultimo vertice seram src
  Se for um P3, um possivel retorno seria [0, 1, 2, 0]
  get_cycle() acha um ciclo euleriano se o grafo for euleriano
  Se for um P3, um possivel retorno seria [0,1,2]
  (Vertice inicial nao se repete)
```

```
O(n + m)
```

```
template<bool directed=false> struct euler {
  int n;
  vector<vector<pair<int, int>>> g;
  vector<int> used;

  euler(int n_) : n(n_), g(n) {}
  void add(int a, int b) {
    int at = used.size();
    used.push_back(0);
    g[a].emplace_back(b, at);
    if (!directed) g[b].emplace_back(a, at);
  }

  pair<bool, vector<pair<int, int>>> get_path(int src) {
    if (!used.size()) return {true, {}};
    vector<int> beg(n, 0);
    for (int& i : used) i = 0;
    // {{vertice, anterior}, label}
    vector<pair<pair<int, int>, int>> ret, st = {{{src, -1},
        -1}};
    while (st.size()) {
      int at = st.back().first.first;
      int& it = beg[at];
      while (it < g[at].size() and used[g[at][it].second]) it
          ++;
      if (it == g[at].size()) {
        if (ret.size() and ret.back().first.second != at)
          return {false, {}};
        ret.push_back(st.back()), st.pop_back();
      } else {
        st.push_back({{g[at][it].first, at}, g[at][it].second
            });
        used[g[at][it].second] = 1;
      }
    }
    if (ret.size() != used.size()+1) return {false, {}};
    vector<pair<int, int>> ans;
    for (auto i : ret) ans.emplace_back(i.first.first, i.
        second);
    reverse(ans.begin(), ans.end());
    return {true, ans};
  }
  pair<bool, vector<pair<int, int>>> get_cycle() {
    if (!used.size()) return {true, {}};
    int src = 0;
    while (!g[src].size()) src++;
    auto ans = get_path(src);
    if (!ans.first or ans.second[0].first != ans.second.back()
        .first)
      return {false, {}};
    ans.second[0].second = ans.second.back().second;
    ans.second.pop_back();
    return ans;
  }
};
```

4.8 EulerTour

```
Euler Tour
  Lineariza um grafo
  Verificar a ancestralida u e ancestral de v se in[u] <=
   in[v] <= out[u]
```

```
#define MAXN 100

bool visitados[MAXN];

void EulerTour(int u, vector<vector<int>>& adj, vector<int>&
     euler){
    euler.push_back(u);
    visitados[u] = true;

    for(auto v : adj[u]){
        if(!visitados[v]){
            EulerTour(v, adj, euler);
            euler.push_back(u);
        }
    }
}
```

4.9 Kosaraju

```
Kosaraju (Grafos fortemente conexos)

g e o grafo (a vai para b)
gi e o grafo reverso (b vai para a)

comp e o componente conexo de cada vertice

graph_condensed() grafo condensado apenas com as sccs
analisa os graus de entrada e saidas de cadas sccs


O(m + n)
```

```
int n;
vector<int> g[MAXN];
vector<int> gi[MAXN];
bool vis[MAXN];
stack<int> S;
int comp[MAXN];
vector<int> condensed[MAXN];
set<int> sccs;
int num_scc = 0;
map<int, long long> grauEntrada;
map<int, long long> grauSaida;
long long sink, source;

void dfs(int k){
    vis[k] = true;
    for(auto i : g[k]){
        if(!vis[i]){
            dfs(i);
        }
    }
    S.push(k);
}

void scc(int k, int c){
    vis[k] = true;
    comp[k] = c;
    for(auto i : gi[k]){
        if(!vis[i]){
            scc(i, c);
        }
    }
}

void kosaraju(){
    for(int i = 0; i < n; i++) vis[i] = false;
    for(int i = 0; i < n; i++) if(!vis[i]) dfs(i);

    for(int i = 0; i < n; i++) vis[i] = false;
    while (S.size())
    {
        int u = S.top();
        S.pop();
        if(!vis[u]) {
            scc(u, u);
            num_scc++;
        }
    }
}

void graph_condensed(){

    for (int i = 0; i < n; i++)
    {
        sccs.insert(comp[i]);
    }

    for (auto i : sccs)
    {
        grauEntrada[i] = 0;
        grauSaida[i] = 0;
    }

    sink = 0; source = 0;

    for (int u = 0; u < n; u++)
    {
        for(auto v : g[u]){
            if(comp[u] != comp[v]){
                condensed[comp[u]].push_back(comp[v]);
                grauEntrada[comp[v]]++; grauSaida[comp[u]]++;
            }
        }
    }

    for (auto i : grauEntrada)
    {
        if(grauEntrada[i.first] == 0) source++;
    }
    for (auto i : grauSaida)
    {
        if(grauSaida[i.first] == 0) sink++;
    }
}
```

4.10 MinCostMaxFlow

```
MCMF find the maximum possible flow from a source to a sink
     while ensuring the total cost of flow is minimized
Cost per unit of flow
Capacity
O(Total Flow + (Edges + Nodes)logNodes)
```

```
struct Aresta {
  int u, v; ll cap, cost;
  Aresta(int u, int v, ll cap, ll cost) : u(u), v(v), cap(cap)
      , cost(cost) {}
};
```

```cpp
struct MCMF {
  const ll INF = numeric_limits<ll>::max();
  int n, source, sink;
  vector<vector<int>> adj;
  vector<Aresta> edges;
  vector<ll> dist, pot;
  vector<int> from;

  MCMF(int n, int source, int sink) : n(n), source(source),
      sink(sink) { adj.resize(n); pot.resize(n); }

  void addAresta(int u, int v, ll cap, ll cost){
    adj[u].push_back(edges.size());
    edges.emplace_back(u, v, cap, cost);

    adj[v].push_back(edges.size());
    edges.emplace_back(v, u, 0, -cost);
  }

  queue<int> q;
  vector<bool> vis;
  bool SPFA(){
    dist.assign(n, INF);
    from.assign(n, -1);
    vis.assign(n, false);

    q.push(source);
    dist[source] = 0;

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

      vis[u] = false;

      for(auto i : adj[u]){
        if(edges[i].cap == 0) continue;
        int v = edges[i].v;
        ll cost = edges[i].cost;

        if(dist[v] > dist[u] + cost + pot[u] - pot[v]){
          dist[v] = dist[u] + cost + pot[u] - pot[v];
          from[v] = i;
          if(!vis[v]) q.push(v), vis[v] = true;
        }
      }
    }

    for(int u=0; u<n; u++)
      if(dist[u] < INF)
        pot[u] += dist[u];

    return dist[sink] < INF;
  }

  pair<ll, ll> augment(){
    ll flow = edges[from[sink]].cap, cost = 0;
    for(int v=sink; v != source; v = edges[from[v]].u)
      flow = min(flow, edges[from[v]].cap),
      cost += edges[from[v]].cost;

    for(int v=sink; v != source; v = edges[from[v]].u)
      edges[from[v]].cap   -= flow,
      edges[from[v]^1].cap += flow;

    return {flow, cost};
  }

  bool inCut(int u){ return dist[u] < INF; }

  pair<ll, ll> maxFlow(){
    ll flow = 0, cost = 0;

    while( SPFA() ){
      auto [f, c] = augment();
      flow += f;
      cost += f*c;
    }
    return {flow, cost};
  }
};
```

4.11 Kruskal

```
Kruskal - Minimum Spanning Tree
Algoritmo para encontrar a Arvore Geradora Minima (MST)
-> Complexity: O(E log E)
E : Numero de Arestas
```

```cpp
/*Create a DSU*/
void join(int u, int v); int find(int u);

const int MAXN = 1e6 + 5;
struct Aresta{ int u, v, c; };
bool compAresta(Aresta a, Aresta b){ return a.c < b.c; }

vector<Aresta> arestas;

int kruskal(){
  sort(begin(arestas), end(arestas), compAresta);
  int resp = 0;
  for(auto a : arestas)
    if( find(a.u) != find(a.v) )
    {
      resp += a.c;
      join(a.u, a.v);
    }
  return resp;
}
```

5 dp

5.1 LIS

```
LIS - Longest Increasing Subsequence

Complexity: O(N Log N)
* For ICREASING sequence, use lower_bound()
* For NON DECREASING sequence, use upper_bound()
```

```cpp
int LIS(vector<int>& nums){
  vector<int> lis;

  for(auto x : nums)
  {
    auto it = lower_bound(lis.begin(), lis.end(), x);
    if(it == lis.end()) lis.push_back(x);
    else *it = x;
  }

  return (int) lis.size();
}
```

5.2 subsetSum

```
Subset sum
Retorna max(x <= t tal que existe subset de w que soma x)

Complexidade
O(n * max(w))
O(max(w)) de memoria
```

```cpp
int subset_sum(vector<int> w, int t) {
  int pref = 0, k = 0;
  while (k < w.size() and pref + w[k] <= t) pref += w[k++];
  if (k == w.size()) return pref;
  int W = *max_element(w.begin(), w.end());
  vector<int> last, dp(2*W, -1);
  dp[W - (t-pref)] = k;
  for (int i = k; i < w.size(); i++) {
    last = dp;
    for (int x = 0; x < W; x++) dp[x+w[i]] = max(dp[x+w[i]],
        last[x]);
    for (int x = 2*W - 1; x > W; x--)
      for (int j = max(0, last[x]); j < dp[x]; j++)
        dp[x-w[j]] = max(dp[x-w[j]], j);
  }
  int ans = t;
  while (dp[W - (t-ans)] < 0) ans--;
  return ans;
}
```

5.3 LCS

```
LCS - Longest Common Subsequence

Complexity: O(N^2)

* Recursive: memset(memo, -1, sizeof memo); LCS(0, 0);
* Iterative: LCS_It();

* RecoverLCS O(N)
  Recover just one of all the possible LCS
```

```cpp
const int MAXN = 5*1e3 + 5;
int memo[MAXN][MAXN];

string s, t;

inline int LCS(int i, int j){
  if(i == s.size() || j == t.size()) return 0;
  if(memo[i][j] != -1) return memo[i][j];

  if(s[i] == t[j]) return memo[i][j] = 1 + LCS(i+1, j+1);

  return memo[i][j] = max(LCS(i+1, j), LCS(i, j+1));
```

```
}


string RecoverLCS(int i, int j){
  if(i == s.size() || j == t.size()) return "";

  if(s[i] == t[j]) return s[i] + RecoverLCS(i+1, j+1);

  if(memo[i+1][j] > memo[i][j+1]) return RecoverLCS(i+1, j);

  return RecoverLCS(i, j+1);
}
```

5.4 SumOVerSubsetDP

```
    SOS DP [nohash]

    Soma de sub-conjunto e de super-conjunto
    O(n 2^n)
```

```
vector<ll> sos_dp_sub(vector<ll> f) {
  int N = __builtin_ctz(f.size());
  assert((1<<N) == f.size());

  for (int i = 0; i < N; i++) for (int mask = 0; mask < (1<<N)
      ; mask++)
    if (mask>>i&1) f[mask] += f[mask^(1<<i)];
  return f;
}


vector<ll> sos_dp_sup(vector<ll> f) {
  int N = __builtin_ctz(f.size());
  assert((1<<N) == f.size());

  for (int i = 0; i < N; i++) for (int mask = 0; mask < (1<<N)
      ; mask++)
    if (~mask>>i&1) f[mask] += f[mask^(1<<i)];
  return f;
}
```

5.5 knapsack

```

Resolve mochila, recuperando a resposta
DP usando os itens [l, r], com capacidade = cap
v[max] e w[MAX] valor e peso


    Complexidade:
    -> O(n * cap), O(n + cap)
```

```
#define MAX (long long) 1e4
#define MAX_CAP (long long) 1e4
#define INF INT_MAX

int v[MAX], w[MAX];
int dp[2][MAX_CAP];

void get_dp(int x, int l, int r, int cap) {
  memset(dp[x], 0, (cap+1)*sizeof(dp[x][0]));
  for (int i = l; i <= r; i++) for (int j = cap; j >= 0; j--)
    if (j - w[i] >= 0) dp[x][j] = max(dp[x][j], v[i] + dp[x][j
        - w[i]]);
}

void solve(vector<int>& ans, int l, int r, int cap) {
  if (l == r) {
    if (w[l] <= cap) ans.push_back(l);
    return;
  }
  int m = (l+r)/2;
  get_dp(0, l, m, cap), get_dp(1, m+1, r, cap);
  int left_cap = -1, opt = -INF;
  for (int j = 0; j <= cap; j++)
    if (int at = dp[0][j] + dp[1][cap - j]; at > opt)
      opt = at, left_cap = j;
  solve(ans, l, m, left_cap), solve(ans, m+1, r, cap -
       left_cap);
}

vector<int> knapsack(int n, int cap) {
  vector<int> ans;
  solve(ans, 0, n-1, cap);
  return ans;
}
```

6 Strings

6.1 trie

```
Trie - Arvore de Prefixos
insert(P) - O(|P|)
count(P)  - O(|P|)
MAXS  - Soma do tamanho de todas as Strings
sigma - Tamanho do alfabeto
```

```
const int MAXS = 1e5 + 10;
const int sigma = 26;

int trie[MAXS][sigma], terminal[MAXS], z = 1;

void insert(string &p){
  int cur = 0;

  for(int i=0; i<p.size(); i++){
    int id = p[i] - 'a';

    if(trie[cur][id] == -1 ){
      memset(trie[z], -1, sizeof trie[z]);
      trie[cur][id] = z++;
    }

    cur = trie[cur][id];
  }

  terminal[cur]++;
}

int count(string &p){
  int cur = 0;

  for(int i=0; i<p.size(); i++){
    int id = (p[i] - 'a');
    if(trie[cur][id] == -1) return 0;

    cur = trie[cur][id];
  }
  return terminal[cur];
}

void init(){
  memset(trie[0], -1, sizeof trie[0]);
  z = 1;
}
```

6.2 Manacher

```
Manacher Algorithm
Find every palindrome in string
Complexidade: O(N)
```

```
vector<int> manacher(string &st){
  string s = "$_";
  for(char c : st){ s += c; s += "_"; }
  s += "#";

  int n = s.size()-2;

  vector<int> p(n+2, 0);
  int l=1, r=1;

  for(int i=1, j; i<=n; i++)
  {
    p[i] = max(0, min(r-i, p[l+r-i]) ); //atualizo o valor
        atual para o valor do palindromo espelho na string ou
        para o total que esta contido

    while( s[i-p[i]] == s[i+p[i]] ) p[i]++;

    if( i+p[i] > r ) l = i-p[i], r = i+p[i];
  }

  for(auto &x : p) x--; //o valor de p[i] e igual ao tamanho
      do palindromo + 1

  return p;
}
```

6.3 KMP

```
    KMP - Find all occurences of a pattern string inside a
     text string

    matching(s, t) retorna os indices das ocorrencias de s em
     t
    autKMP constroi o automato do KMP

    Complexidades:
    pi - O(n)
    match - O(n + m)
    construir o automato - O(|sigma|*n)
    n = |padrao| e m = |texto|
```

```cpp
template<typename T> vector<int> pi(T s) {
  vector<int> p(s.size());
  for (int i = 1, j = 0; i < s.size(); i++) {
    while (j and s[j] != s[i]) j = p[j-1];
    if (s[j] == s[i]) j++;
    p[i] = j;
  }
  return p;
}

template<typename T> vector<int> matching(T& s, T& t) {
  vector<int> p = pi(s), match;
  for (int i = 0, j = 0; i < t.size(); i++) {
    while (j and s[j] != t[i]) j = p[j-1];
    if (s[j] == t[i]) j++;
    if (j == s.size()) match.push_back(i-j+1), j = p[j-1];
  }
  return match;
}

struct KMPaut : vector<vector<int>> {
  KMPaut(){}
  KMPaut (string& s) : vector<vector<int>>(26, vector<int>(s.
       size()+1)) {
    vector<int> p = pi(s);
    auto& aut = *this;
    aut[s[0]-'a'][0] = 1;
    for (char c = 0; c < 26; c++)
      for (int i = 1; i <= s.size(); i++)
        aut[c][i] = s[i]-'a' == c ? i+1 : aut[c][p[i-1]];
  }
};
```

6.4 hash

```
String Hash
precalc()     -> O(N)
StringHash() -> O(|S|)
gethash()     -> O(1)

StringHash hash(s); -> Cria uma struct de StringHash para a
     string s
hash.gethash(l, r); -> Retorna o hash do intervalo L R da
     string (0-Indexado)

IMPORTANTE! Chamar precalc() no inicio do codigo

const ll MOD  = 131'807'699; -> Big Prime Number
const ll base = 127;          -> Random number larger than the
     Alphabet
```

```cpp
const int MAXN = 1e6 + 5;

const ll MOD = 1e9 + 7; //WA? Muda o MOD e a base
const ll base = 153;

ll expb[MAXN];

void precalc(){
  expb[0] = 1;
  for(int i=1; i<MAXN; i++)
    expb[i] = (expb[i-1]*base)%MOD;
}

struct StringHash{
  vector<ll> hsh;

  StringHash(string &s){
    hsh.assign(s.size()+1, 0);
    for(int i=0; i<s.size(); i++)
      hsh[i+1] = (hsh[i] * base % MOD + s[i]) % MOD;
  }

  ll gethash(int l, int r){
    return (MOD + hsh[r+1] - hsh[l]*expb[r-l+1] % MOD ) % MOD;
  }
};
```

---

7 Math

7.1 totient

```
Totiente de Euler - Conta quantos numeros de 1 ate n sao
 coprimos de n

Complexidade:
O(sqrt(n))
```

```cpp
// Totiente
//
// O(sqrt(n))

int tot(int n) {
  int ret = n;

  for (int i = 2; i*i <= n; i++) if (n % i == 0) {
    while (n % i == 0) n /= i;
    ret -= ret / i;
  }
  if (n > 1) ret -= ret / n;

  return ret;
}
```

7.2 sieve

```
Sieve of Eratosthenes - Encontra o maior divisor primo
Fact -> Fatora um numero <= limite, sai ordenada
Crivo calcula a lista de primos

Crivo_mobius
 - 1 se n=1
 - 0 se n tem algum fator primo ao quadrado
 - (-1)^k se n e produto de k primos distintos

A funcao fact adiciona o numero 1 se vc tentar fatorar o 1.
Complexidade:
crivo - O(n log(logN))
fact - O(log(n))
```

```cpp
#define MAX 1000
int divi[MAX];
vector<int> primes;

void crivo(int lim) {
  divi[1] = 1;
  for (int i = 2; i <= lim; i++) {
    if (divi[i] == 0) divi[i] = i, primes.push_back(i);
    for (int j : primes) {
      if (j > divi[i] or i*j > lim) break;
      divi[i*j] = j;
    }
  }
}

void crivo_mobius(ll lim) {
    mobius[1] = 1;
    for (int i = 2; i <= lim; i++) {
        if (!divi[i]) {
            divi[i] = i;
            primes.push_back(i);
            mobius[i] = -1; // primo        = -1
        }
        for (int p : primes) {
            if (p > divi[i] || 1LL * i * p > lim) break;
            divi[i*p] = p;
            if (i % p == 0) {
                mobius[i*p] = 0; // quadrado        = 0
                break;
            } else {
                mobius[i*p] = -mobius[i];
            }
        }
    }
}

void fact(vector<int>& v, int n) {
  if (n != divi[n]) fact(v, n/divi[n]);
  v.push_back(divi[n]);
}
```

7.3 ModComb

```
Combinacao modular
Inverso modular
Exponenciacao rapida (O (Log P ) - p: potencia)
O(N) fatorial
```

```cpp
#define MOD 9987123

vector<ll> fact(1e6, -1);

void pre(){
    fact[0] = 1;
    for(ll i = 1; i < fact.size(); i++){
        fact[i] = (fact[i-1] * i) % MOD;
    }
}

ll fexp(ll a, ll b){
    ll ans = 1;

    while(b){
        if(b & 1) ans = (ans * a) % MOD;
        a = (a*a) % MOD;
        b >>= 1;
    }

    return ans;
```

```
}

ll inv(ll a, ll p){
    return fexp(a, p -2);
}

ll comb(ll n, ll k, ll p){
    return ((fact[n] * inv(fact[k], p)) % p) * inv(fact[n-k],
        p) % p;
}
```

7.4 Kadane

```
    Algoritmo de Kadane
    Consegue o max subarray sum
    O(N)
```

```
ll kadane(vector<ll> &arr) {
    ll answ = arr[0];
    ll maxEnding = arr[0];

    for (int i = 1; i < arr.size(); i++) {
        maxEnding = max(maxEnding + arr[i], arr[i]);

        answ = max(answ, maxEnding);
    }

    return answ;
}
```

7.5 divisors

```
    get_divisors(n) -- O(sqrt(N))
```

```
vector<int> get_divisors(int n ){
    vector<int> divisors;

    for(int i = 1; i*i <= n; i++){
        if(n % i == 0){
            divisors.push_back(i);
            if(i != n/i) divisors.push_back(n/i);
        }
    }
    return divisors;
}
```

Combinatorics

General

$$\sum_{0\leq k\leq n} n-kk = Fib_{n+1}$$

$$nk = nn-k$$

$$nk + nk+1 = n+1k+1$$

$$knk = nn-1k-1$$

$$nk = \frac{n}{k}n-1k-1$$

$$\sum_{i=0}^{n} ni = 2^n$$

$$\sum_{i\geq 0} n2i = 2^{n-1}$$

$$\sum_{i\geq 0} n2i+1 = 2^{n-1}$$

$$\sum_{i=0}^{k} (-1)^i ni = (-1)^k n-1k$$

$$\sum_{i=0}^{k} n+ii = \sum_{i=0}^{k} n+in = n+k+1k$$

$$1n1 + 2n2 + 3n3 + \ldots + nnn = n2^{n-1}$$

$$1^2n1 + 2^2n2 + 3^2n3 + \ldots + n^2nn = (n+n^2)2^{n-2}$$

0.1 Vandermonde's Identity:

$$\sum_{k=0}^{r} mknr-k = m+nr$$

0.2 Hockey-Stick Identity:

$$n, r \in N,\ n > r, \quad \sum_{i=r}^{n} ir = n+1r+1$$

$$\sum_{i=0}^{k} ki2^i = 2kk$$

$$\sum_{k=0}^{n} nknn-k = 2nn$$

$$\sum_{k=q}^{n} nkkq = 2^{n-q}nq$$

$$\sum_{i=0}^{n} k^i ni = (k+1)^n$$

$$\sum_{i=0}^{n} 2ni = 2^{2n-1} + 122nn$$

$$\sum_{i=1}^{n} nin-1i-1 = 2n-1n-1$$

$$\sum_{i=0}^{n} 2ni^2 = 12\left(4n2n + 2nn^2\right)$$

0.3 Highest Power of 2 that divides $2nn$:

Let x be the number of 1s in the binary representation. Then the number of odd terms will be 2^x. Let it form a sequence. The n-th value in the sequence (starting from $n = 0$) gives the highest power of 2 that divides $2nn$.

Pascal Triangle

In a row p, where p is a prime number, all the terms in that row except the 1s are multiples of p. **Parity:** To count odd terms in row n, convert n to binary. Let x be the number of 1s in the binary representation. Then the number of odd terms will be 2^x. Every entry in row $2^n - 1,\ n \geq 0$, is odd. An integer $n \geq 2$ is prime if and only if all intermediate binomial coefficients are inserted.

$$n1, n2, \ldots, nn-1$$

are divisible by n.

0.4 Kummer's Theorem

For given integers $n \geq m \geq 0$ and a prime number p, the largest power of p dividing nm is equal to the number of carries when m is added to $n - m$ in base p. For implementation, take inspiration from Lucas theorem.

0.5 Counting Problems

Number of different binary sequences of length n such that no two 0's are adjacent:

$$Fib_{n+1}$$

0.6 Combination with repetition

Choosing k elements from an n-element set, order does not matter, repetition allowed:

$$n + k - 1k$$

Number of ways to divide n persons in nk equal groups of size k:

$$\frac{n!}{k!^{n/k}(n/k)!} = \prod_{n \geq k} n - 1k - 1$$

Number of non-negative solutions of equation:

$$x_1 + x_2 + x_3 + \ldots + x_k = n \quad \Rightarrow \quad n + k - 1n$$

Number of ways to choose n ids from 1 to b such that every id has distance at least k:

$$b - (n-1)(k-1)n$$

Restricted Cycle Permutations

Let $T(n, k)$ be the number of permutations of size n for which all cycles have length $\leq k$:

$$T(n,k) = \{ \, n!n \leq kn \cdot T(n-1,k) - F(n-1,k) \cdot T(n-k-1,k)n > k$$

where

$$F(n,k) = n(n-1)\ldots(n-k+1)$$

Lucas Theorem

If p is prime, then

$$pak \equiv 0 \pmod{p}$$

For non-negative integers m and n and a prime p:

$$mn \equiv \prod_{i=0}^{k} m_i n_i \pmod{p}$$

where

$$m = m_k p^k + m_{k-1} p^{k-1} + \ldots + m_1 p + m_0,$$
$$n = n_k p^k + n_{k-1} p^{k-1} + \ldots + n_1 p + n_0$$

are the base-p expansions of m and n. Convention: $mn = 0$ when $m < n$.

0.1 Propriedades Matemáticas

- **Conjectura de Goldbach:** Todo número par $n > 2$ pode ser representado como $n = a + b$, onde a e b são primos.
- **Primos Gêmeos:** Existem infinitos pares de primos $p, p + 2$.
- **Conjectura de Legendre:** Sempre existe um primo entre n^2 e $(n + 1)^2$.
- **Lagrange:** Todo número inteiro pode ser representado como soma de 4 quadrados.
- **Zeckendorf:** Todo número pode ser representado como soma de números de Fibonacci diferentes e não consecutivos.
- **Tripla de Pitágoras (Euclides):** Toda tripla pitagórica primitiva pode ser gerada por $(n^2 - m^2, 2nm, n^2 + m^2)$ onde n e m são coprimos e um deles é par.
- **Wilson:** n é primo se e somente se $(n - 1)! \equiv -1 \pmod{n}$.
- **Problema do McNugget:** Para dois coprimos x e y, o número de inteiros não representáveis como $ax + by$ é $\frac{(x-1)(y-1)}{2}$. O maior inteiro não representável é $xy - x - y$.
- **Fermat:** Se p é primo, então $a^{p-1} \equiv 1 \pmod{p}$. Se x e m são coprimos e m é primo, então $x^k \equiv x^{k \bmod (m-1)} \pmod{m}$. *Euler:* $x^{\varphi(m)} \equiv 1 \pmod{m}$, onde $\varphi(m)$ é o totiente de Euler.
- **Teorema Chinês do Resto:** Dado o sistema:

$$x \equiv a_1 \pmod{m_1}, \quad \dots, \quad x \equiv a_n \pmod{m_n}$$

  com m_i coprimos dois a dois. Seja $M_i = \frac{m_1 m_2 \cdots m_n}{m_i}$ e $N_i \equiv M_i^{-1} \pmod{m_i}$. A solução é:

$$x = \sum_{i=1}^{n} a_i M_i N_i \pmod{m_1 m_2 \cdots m_n}$$

- **Números de Catalan:** Exemplo: expressões de parênteses bem formadas. $C_0 = 1$, e:

$$C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i} = \frac{1}{n+1}\binom{2n}{n}$$

- **Bertrand (Ballot):** Com $p > q$ votos, a probabilidade de sempre haver mais votos do tipo A do que B até o fim é:

$$\frac{p - q}{p + q}$$

  Permitindo empates:

$$\frac{p + 1 - q}{p + 1}$$

  Multiplicando pela combinação total $\binom{p+q}{q}$, obtém-se o número de possibilidades.
- **Linearidade da Esperança:** $E[aX + bY] = aE[X] + bE[Y]$
- **Variância:** $\mathrm{Var}(X) = E[(X - \mu)^2] = E[X^2] - (E[X])^2$
- **Progressão Geométrica:** $S_n = a_1 \cdot \dfrac{q^n - 1}{q - 1}$
- **Soma dos Cubos:** $\displaystyle\sum_{k=1}^{n} k^3 = \left(\sum_{k=1}^{n} k\right)^2$

- **Lindström-Gessel-Viennot:** A quantidade de caminhos disjuntos em um grid pode ser computada como o determinante da matriz do número de caminhos.
- **Lema de Burnside:** Número de colares diferentes (sem rotações), com m cores e comprimento n:

$$\frac{1}{n}\sum_{i=0}^{n-1} m^{\gcd(i,n)}$$

- **Inversão de Möbius:**

$$\sum_{d|n} \mu(d) = \begin{cases} 1, & n = 1 \\ 0, & \text{caso contrário} \end{cases}$$

- **Propriedades de Coeficientes Binomiais:**

$$\binom{N}{N-K} = \frac{N}{K}\binom{N-1}{K-1} = \binom{N}{K}$$

$$\sum_{k=0}^{m}(-1)^k\binom{n}{k} = (-1)^m\binom{n-1}{m}$$

$$\sum_{k=0}^{n}\binom{n}{k} = 2^n, \qquad \sum_{k=0}^{n} k\binom{n}{k} = n \cdot 2^{n-1}$$

$$\sum_{m=0}^{n}\binom{m}{k} = \binom{n+1}{k+1}, \qquad \sum_{k=0}^{n}\binom{n-k}{k} = F_{n+1}$$

$$\sum_{k=0}^{m}\binom{n+k}{k} = \binom{n+m+1}{m}, \qquad \sum_{k=0}^{n}\binom{n}{k}^2 = \binom{2n}{n}$$

- **Triângulo de Pascal**
  (ilustração omitida)
- **Identidades Clássicas:**
  - **Hockey-stick:** $\sum_{i=r}^{n}\binom{i}{r} = \binom{n+1}{r+1}$
  - **Vandermonde:** $\binom{m+n}{r} = \sum_{k=0}^{r}\binom{m}{k}\binom{n}{r-k}$
- **Distribuições de Probabilidade:**
  - **Uniforme:** $X \in \{a, a+1, \dots, b\}, \quad E[X] = \frac{a+b}{2}$
  - **Binomial:** n tentativas com probabilidade p de sucesso:

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}, \quad E[X] = np$$

  - **Geométrica:** Número de tentativas até o primeiro sucesso:

$$P(X = x) = (1-p)^{x-1} p, \quad E[X] = \frac{1}{p}$$