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)
```

```cpp
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)
```

```cpp
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)
```

```cpp
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 1a linha
```

```
  for (int i = 1; i < MAXN; i++) ps[i][0] += ps[i - 1][0];  //
       inicializo a 1a 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]; }
```

---

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);
    }
};
```

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
```

```cpp
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

```cpp
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

```cpp
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
```

```cpp
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 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)
```

```cpp
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) {}
};

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++)  //fix pot
      if(dist[u] < INF)
        pot[u] += dist[u];
```

```
    return dist[sink] < INF;
  }

  pair<ll, ll> augment(){
    ll flow = edges[from[sink]].cap, cost = 0; //fixed flow:
        flow = min(flow, remainder)

    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};
  }
};
```

4.6 Kruskal

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

```
/*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;       //Lista de Arestas

int kruskal(){
  sort(begin(arestas), end(arestas), compAresta); //Ordena
       pelo custo
  int resp = 0;          //Custo total da MST

  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()
```

```
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
```

```
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
```

```
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 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|
```

```
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]];
  }
};
```

---

7 Math

7.1 totient

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


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

```
// 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

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

```
#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 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
```

```
#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;
}
```
