#INCLUDE <BITS/STDC++.H>

0 Defines/Métodos para resolver problemas 1

0.1 Caminhos mínimos disjuntos. 1

1 Cálculo numérico 2

1.1 Zero de funções 2

1.2 Integração 2

1.3 Derivação 2

2 Funções de recorrência 3

2.1 Números eulerianos 3

2.2 Triângulo de Bell 3

2.3 Triângulo de Catalão 3

3 Algoritmos 5

3.1 Algoritmo de Euclides Estendido 5

3.2 Máximo divisor comum 5

3.3 Algoritmo de Pollard Rho 5

3.4 Mínimo múltiplo comum 6

3.5 Algoritmo Húngaro 6

3.6 Fatoração em números primos 8

3.7 Matching máximo em grafo bipartido 9

3.8 ModPow 10

3.9 Máximo e mínimo de funções 10

3.10 Mergesort 11

3.11 Todos divisores de um número 11

3.12 Suffix Tree 12

3.13 Crivo de Eratóstenes segmentado (para gerar primos muito grandes) 14

3.14 Kd-Tree para Queries de pares mais próximos em O(log(n)) 15

3.15 Algoritmo de Mo (Queries offline, muitas queries e impossibilidade de SegTree) 16

3.16 Fluxo de custo mínimo 18

3.17 Teorema da Convolução utilizando Transformada Rápida de Fourrier 19

3.18 Transformada de Fourrier (Código menor) 21

3.19 Gerar matriz de $\boldsymbol{Cnk}$ 22

3.20 Sweep radial – Rotacionar uma reta em um conjunto de pontos. 22

4 Algoritmos em JAVA 24

4.1 2-sat 24

4.2 Simplex 26
4.3 SegTree 2D 28
5 Bibliotecas e estruturas 30
5.1 Geometria computacional reduzido 30
5.2 Geometria computacional completo 32
5.3 Grafos 35
5.4 Segtree com lazy propagation 39
5.5 Matrizes 41
5.6 Union-Find com ranking 42
5.7 Wavelet-Tree 43
6 Bibliotecas STD 45
6.1 Vector 45
6.2 Map 45
6.3 Queue / Priority_queue 45

0 DEFINES/MÉTODOS PARA RESOLVER PROBLEMAS

```
#include<bits/stdc++.h>
using namespace std;
#define D(x) cout << #x " = " << (x) << endl
```

0.1 CAMINHOS MÍNIMOS DISJUNTOS.

Resolver utilizando Fluxo Mínimo.

Passo 1: Dobrar todo vértice, botando o fluxo do $V_{in} \rightarrow V_{out} = 1$.

Passo 2: Cada aresta $a \rightarrow b$ será representada por $a_{out} \rightarrow b_{in}$.

Passo 3: Adicionar dois vértices virtuais, o de origem e o de destino.

Passo 4: Ligar o vértice virtual de origem ao vértice de saída do vértice inicial e ligar o vértice virtual de destino ao vértice de entrada do vértice final.

Passo 5: Obter o fluxo mínimo entre os vértices virtuais.

1 CÁLCULO NUMÉRICO

1.1 ZERO DE FUNÇÕES

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

1.2 INTEGRAÇÃO

$$\int_a^b f(x)dx \cong \frac{h}{3}\left(f(x_1) + 4f(x_2) + 2f(x_3) + 4f(x_4) + \cdots + 4f(x_{n-1}) + f(x_n)\right)$$

$$h = \frac{b-a}{n}, \qquad n\,\%\,2 = 1, \qquad x_1 = a, \qquad x_{i+1} = x_i + h, \qquad x_n = b$$

1.3 DERIVAÇÃO

$$f'(x) \cong \frac{f(x+h) - f(x-h)}{2h}$$

$$onde\ quanto\ menor\ o\ h, maior\ a\ precisão$$

2 FUNÇÕES DE RECORRÊNCIA

2.1 NÚMEROS EULERIANOS

1						
1	1					
1	4	1				
1	11	11	1			
1	26	66	26	1		
1	57	302	302	57	1	
1	120	1191	2416	1191	120	1

Função de recorrência:

$$A_{n,1} = A_{n,n} = 1$$
$$A_{n+1,k} = k\,A_{n,k} + (n+2-k)\,A_{n,k-1}$$

2.2 TRIÂNGULO DE BELL

1						
1	2					
2	3	5				
5	7	10	15			
15	20	27	37	52		
52	67	87	113	151	203	
203	255	322	409	523	674	877

Modo de construção:

$$A_{1,1} = 1$$
$$A_{n,1} = A_{(n-1,n-1)}$$
$$A_{n,k} = A_{n,k-1} + A_{n-1,k-1}$$

2.3 TRIÂNGULO DE CATALÃO

1						
1	1					
1	2	2				
1	3	5	5			
1	4	9	14	14		
1	5	14	28	42	42	
1	6	20	48	90	132	132

Modo de construção:

$$C(n,0) = 1$$
$$C(n,1) = n$$
$$C(n+1,k) = C(n+1,k-1) + C(n,k)$$
$$C(n+1,n+1) = C(n+1,n)$$

3 ALGORITMOS

3.1 ALGORITMO DE EUCLIDES ESTENDIDO

Objetivo: Dados os inteiros a e b, calcular o $mdc(a,b)$ e achar valores para x e y tais que $a * x + b * y = mdc(a,b)$.
Entrada: a, b.
Saída: x, y, $mdc(a,b)$.

```
int xmdc(int a, int b, int &x, int &y) {
	if(b == 0){
		x = 1;
		y = 0;
		return a;
	}

	int x1, y1, mdc = xmdc(b, a % b, x1, y1);
	x = y1;
	y = x1 - (a / b) * y1;
	return mdc;
}
```

3.2 MÁXIMO DIVISOR COMUM

```
int mdc(int a, int b) {
	int remainder;
	while (b != 0) {
		remainder = a % b;
		a = b;
		b = remainder;
	}
	return a;
}
```

3.3 ALGORITMO DE POLLARD RHO

Objetivo: Decompor um número inteiro em seus dois maiores fatores primos.
Entrada: Um número inteiro.
Saída: O menor dos dois fatores do número.

```
int pollard(int number) {
	x_fixed = 2, cycle_size = 2, x = 2, factor = 1;

	while (factor == 1) {
		for (int count=1;count <= cycle_size && factor <= 1;count++) {
			x = (x*x+1)%number;
			factor = mdc(x - x_fixed, number);
		}

		cycle_size *= 2;
		x_fixed = x;
	}
	return factor;
}
```

3.4 MÍNIMO MÚLTIPLO COMUM

```
int mmc(int a, int b) {
    int temp = mdc(a, b);

    return temp ? (a / temp * b) : 0;
}
```

3.5 ALGORITMO HÚNGARO

```
 Objetivo:  Dados que temos n trabalhadores e n tarefas a serem feitos,
            e para cada par (trabalhador, tarefa) teremos um valor
            salário. O resultado será o custo de realizar todas as
            tarefas com um custo mínimo, onde um trabalho será
            realizado por apenas um trabalhador e vice-versa.
 Entrada:   Uma matriz de custos[n][n], onde o valor armazenado em
            [i][j] é o salário do trabalhador i para realizar o
            trabalho j.
 Saída:     Valor mínimo para a realização de todas tarefas.
```

```
#define N 100
#define INF 100000000

int cost[N][N], n, max_match, lx[N], ly[N], xy[N], yx[N], slack[N], slackx[N], prev[N];
bool S[N], T[N];

void init_labels() {
      memset(lx, 0, sizeof(lx));
      memset(ly, 0, sizeof(ly));
      for (int x = 0; x < n; x++)
            for (int y = 0; y < n; y++)
                  lx[x] = max(lx[x], cost[x][y]);
}

void update_labels() {
      int x, y, delta = INF;
      for (y = 0; y < n; y++)
            if (!T[y])
                  delta = min(delta, slack[y]);
            for (x = 0; x < n; x++)
                  if (S[x])
                        lx[x] -= delta;
            for (y = 0; y < n; y++)
                  if (T[y])
                        ly[y] += delta;
            for (y = 0; y < n; y++)
                  if (!T[y])
                        slack[y] -= delta;
}

void add_to_tree(int x, int prevx) {
      S[x] = true;
      prev[x] = prevx;
      for (int y = 0; y < n; y++)
            if (lx[x] + ly[y] - cost[x][y] < slack[y]) {
                  slack[y] = lx[x] + ly[y] - cost[x][y];
                  slackx[y] = x;
            }
}

void augment() {
```

```
if (max_match == n)
	return;
int x, y, root, q[N], wr = 0, rd = 0;
memset(S, false, sizeof(S));
memset(T, false, sizeof(T));
memset(prev, -1, sizeof(prev));

for (x = 0; x < n; x++)
	if (xy[x] == -1) {
		q[wr++] = root = x;
		prev[x] = -2;
		S[x] = true;
		break;
	}

for (y = 0; y < n; y++) {
	slack[y] = lx[root] + ly[y] - cost[root][y];
	slackx[y] = root;
}

while (true) {
	while (rd < wr) {
		x = q[rd++];
		for (y = 0; y < n; y++)
			if (cost[x][y] == lx[x] + ly[y] && !T[y]) {
				if (yx[y] == -1)
					break;
				T[y] = true;
				q[wr++] = yx[y];
				add_to_tree(yx[y], x);
			}

		if (y < n)
			break;
	}

	if (y < n)
		break;

	update_labels();
	wr = rd = 0;
	for (y = 0; y < n; y++)
		if (!T[y] && slack[y] == 0) {
			if (yx[y] == -1) {
				x = slackx[y];
				break;
			}
		else {
			T[y] = true;
			if (!S[yx[y]]) {
				q[wr++] = yx[y];
				add_to_tree(yx[y], slackx[y]);
			}
		}
	}

	if (y < n)
		break;
}
if (y < n) {
```

```
            max_match++;
            for (int cx = x, cy = y, ty; cx != -2; cx = prev[cx], cy = ty) {
                  ty = xy[cx];
                  yx[cy] = cx;
                  xy[cx] = cy;
            }

            augment();
      }
}

int hungaro() {
      int ret = 0;
      max_match = 0;
      memset(xy, -1, sizeof(xy));
      memset(yx, -1, sizeof(yx));
      init_labels();
      augment();

      for (int x = 0; x < n; x++)
            ret += cost[x][xy[x]];

      return ret;
}
```

3.6 FATORAÇÃO EM NÚMEROS PRIMOS

```
Objetivo:  Dado um número n, obter um vetor com todos seus fatores
           primos. Por exemplo: entrada: 4, saída: [2, 2].
Entrada:   Um número n > 0.
Saída:     Um vector contendo todos seus fatores primos, incluindo
           números repetidos.
```

```
vector<int> primeFactors(int n) {
      vector<int> v;
      int sqrtn = sqrt(n);

      while (!(n & 1)) {
            v.push_back(2);
            n >>= 1;
      }

      for (int i = 3; i <= sqrtn; i = i+2) {
            while (n % i == 0) {
                  v.push_back(i);
                  n = n/i;
            }
      }

      if (n > 2)
            v.push_back(n);

      return v;
}
```

3.7 MATCHING MÁXIMO EM GRAFO BIPARTIDO

```
Objetivo:  Obter o matching máximo em um grafo bipartido em O(√v * E).
Entrada:   Um grafo com n1 vértices no seu lado direito e n2 vértices
           no seu lado esquerdo, com todas as arestas já adicionadas.
Saída:     O valor do fluxo máximo.
Uso:       Inicializar:      init(n_Esquerda, n_Direita);
           Adicionar aresta: addAresta(origem, destino);
```

```
const int MAXN1 = 50000, MAXN2 = 50000, MAXM = 150000;
int n1, n2, edges, last[MAXN1], prev[MAXM], head[MAXM], matching[MAXN2], dist[MAXN1],
Q[MAXN1], used[MAXN1], vis[MAXN1];

void init(int _n1, int _n2) {
    n1 = _n1;
    n2 = _n2;
    edges = 0;
    fill(last, last + n1, -1);
}

void addAresta(int u, int v) {
    head[edges] = v;
    prev[edges] = last[u];
    last[u] = edges++;
}


void bfs() {
    fill(dist, dist + n1, -1);
    int sizeQ = 0;
    for (int u = 0; u < n1; ++u) {
        if (!used[u]) {
            Q[sizeQ++] = u;
            dist[u] = 0;
        }
    }
    for (int i = 0; i < sizeQ; i++) {
        int u1 = Q[i];
        for (int e = last[u1]; e >= 0; e = prev[e]) {
            int u2 = matching[head[e]];
            if (u2 >= 0 && dist[u2] < 0) {
                dist[u2] = dist[u1] + 1;
                Q[sizeQ++] = u2;
            }
        }
    }
}

bool dfs(int u1) {
    vis[u1] = true;
    for (int e = last[u1]; e >= 0; e = prev[e]) {
        int v = head[e];
        int u2 = matching[v];
        if (u2 < 0 || !vis[u2] && dist[u2] == dist[u1] + 1 && dfs(u2)) {
            matching[v] = u1;
            used[u1] = true;
            return true;
        }
    }
    return false;
}
```

```
int maxMatching() {
    fill(used, used + n1, false);
    fill(matching, matching + n2, -1);
    for (int res = 0;;) {
        bfs();
        fill(vis, vis + n1, false);
        int f = 0;
        for (int u = 0; u < n1; ++u)
        if (!used[u] && dfs(u))
                ++f;
        if (!f)
            return res;
        res += f;
    }
}
```

3.8 MODPOW

Objetivo: Obter o resultado de $a^b \ (mod\ m)$
Entrada: $a, b, m.$
Saída: $a^b (mod\ m)$

```
int modPow(int a, int b, int m) {
    int res = 1;
    for (; b > 0; b >>= 1) {
        if (b & 1)
            res = (long long) res * a % m;
        a = (long long) a * a % m;
    }
    return res;
}
```

3.9 MÁXIMO E MÍNIMO DE FUNÇÕES

Objetivo: Dada uma função f, que recebe um valor double, achar seu ponto de minímo ou máximo
Entrada: a, b (mínimo e máximo do domínio); f(x); erro máximo.
Saída: $\min f(x)$ ou $\max f(x)$

```
double gss(double a, double b, double (*f)(double), double e = 1e-6) {
    double r = (sqrt(5)-1)/2; //=.618...=golden ratio-1
    double x1 = b-r*(b-a), x2 = a+r*(b-a);
    double f1 = f(x1), f2 = f(x2);
    while (b-a > e) {
        if (f1 < f2) { //change to > to find maximum
            b = x2; x2 = x1; f2 = f1;
            x1 = b-r*(b-a); f1 = f(x1);
        } else {
            a = x1; x1 = x2; f1 = f2;
            x2 = a+r*(b-a); f2 = f(x2);
        }
    }
    return (b + a) / 2;
}
```

3.10 MERGESORT

```
typedef vector<int>::iterator vec_it;

void merge(vec_it left, vec_it left_end, vec_it right, vec_it right_end, vec_it
numbers) {
    while (left != left_end) {
        if (*left < *right || right == right_end) {
            *numbers = *left;
            ++left;
        } else {
            *numbers = *right;
            ++right;
        }

        ++numbers;
    }

    while (right != right_end) {
        *numbers = *right;
        ++right;
        ++numbers;
    }
}

void merge_sort(vector<int>& numbers) {
    if (numbers.size() <= 1)
        return;

    vector<int>::size_type middle = numbers.size() / 2;
    vector<int> left(numbers.begin(), numbers.begin() + middle);
    vector<int> right(numbers.begin() + middle, numbers.end());

    merge_sort(left);
    merge_sort(right);

    merge(left.begin(), left.end(), right.begin(), right.end(), numbers.begin());
}
```

3.11 TODOS DIVISORES DE UM NÚMERO

```
vector<int> divisores(int n) {
      vector<int> div;
      sqrtn = sqrt(n) + 1;

      for(i = 1; i <= sqrtn; i++)
            if(!(n % i))
                  div.push_back(i), div.push_back(n / i);

      return div;
}
```

3.12 Suffix Tree

```
//Usage:
// Fill txt with the characters of the txting.
// Call SuffixSort(n), where n is the length of the txting stored in txt.
// That's it!

//Output:
// SA = The suffix array.
// Contains the n suffixes of txt sorted in lexicographical order.
// Each suffix is represented as a single integer (the SAition of txt where it starts).
// iSA = The inverse of the suffix array. iSA[i] = the index of the suffix txt[i..n)
//   in the SA array. (In other words, SA[i] = k <==> iSA[k] = i)
//   With this array, you can compare two suffixes in O(1): Suffix txt[i..n) is smaller
//   than txt[j..n) if and only if iSA[i] < iSA[j]

const int MAX = 100010;
char txt[MAX]; //input
int iSA[MAX], SA[MAX]; //output
int cnt[MAX], next[MAX]; //internal
bool bh[MAX], b2h[MAX];

// Compares two suffixes according to their first characters
bool smaller_first_char(int a, int b){
	return txt[a] < txt[b];
}

void suffixSort(int n){
	for (int i=0; i<n; ++i)
		SA[i] = i;

	sort(SA, SA + n, smaller_first_char);

	for (int i=0; i<n; ++i) {
		bh[i] = i == 0 || txt[SA[i]] != txt[SA[i-1]];
		b2h[i] = false;
	}

	for (int h = 1; h < n; h <<= 1){
		int buckets = 0;

		for (int i=0, j; i < n; i = j){
			j = i + 1;

			while (j < n && !bh[j]) j++;

			next[i] = j;
			buckets++;
		}

		if (buckets == n) break;

		for (int i = 0; i < n; i = next[i]){
			cnt[i] = 0;

			for (int j = i; j < next[i]; ++j)
				iSA[SA[j]] = i;
		}
```

```
            cnt[iSA[n - h]]++;
            b2h[iSA[n - h]] = true;

            for (int i = 0; i < n; i = next[i]) {
                  for (int j = i; j < next[i]; ++j) {
                        int s = SA[j] - h;

                        if (s >= 0){
                              int head = iSA[s];
                              iSA[s] = head + cnt[head]++;
                              b2h[iSA[s]] = true;
                        }
                  }
                  for (int j = i; j < next[i]; ++j){
                        int s = SA[j] - h;

                        if (s >= 0 && b2h[iSA[s]])
                              for (int k = iSA[s]+1; !bh[k] && b2h[k]; k++)
                                    b2h[k] = false;
                  }
            }
            for (int i=0; i<n; ++i){
                  SA[iSA[i]] = i;
                  bh[i] |= b2h[i];
            }
      }
      for (int i=0; i<n; ++i){
            iSA[SA[i]] = i;
      }
}
// End of suffix array algorithm

// Begin of the O(n) longest common prefix algorithm
int lcp[MAX];
// lcp[i] = length of the longest common prefix of suffix SA[i] and suffix SA[i-1]
// lcp[0] = 0
void getlcp(int n) {
      for (int i=0; i<n; ++i)
            iSA[SA[i]] = i;

      lcp[0] = 0;

      for (int i=0, h=0; i<n; ++i) {
            if (iSA[i] > 0) {
                  int j = SA[iSA[i]-1];

                  while (i + h < n && j + h < n && txt[i+h] == txt[j+h])
                        h++;

                  lcp[iSA[i]] = h;

                  if (h > 0)
                        h--;
            }
      }
}
```

3.13 CRIVO DE ERATÓSTENES SEGMENTADO (PARA GERAR PRIMOS MUITO GRANDES)

```
char nprimo[100001] = {0}; // tamanho = sqrt(maximo)
std::vector<int> primos;

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

	nprimo[1] = 1;
	nprimo[0] = 1;

	for(i = 2; i < 320; i++) // i [2, sqrt(sqrt(maximo))]
		if(!nprimo[i])
			for(j = i * i; j < 100001; j += i) // j [i^2, sqrt(maximo)]
				nprimo[j] = 1;

	for(i = 2; i < 100001; i++)
		if(!nprimo[i])
			primos.push_back(i);


	scanf("%d", &n);

	while(n--) {
		scanf("%d %d", &a, &b);

		if(a > 100000 && b > 100000) { // (a > sqrt(N) && b > sqrt(N))
			for(i = a; i <= b; i++) {
				for(j = 0; j < primos.size(); j++)
					if(i % primos[j] == 0)
						goto ab;

				printf("%d\n", i);
				ab:;
			}
		}
		else if(a < 100001 && b < 100001) { // (a < sqrt(N) && b < sqrt(N))
			for(i = a; i <= b; i++)
				if(!nprimo[i])
					printf("%d\n", i);
		}
		else {
			for(i = 0; i < primos.size(); i++)
				if(primos[i] >= a)
					break;

			for(; i < primos.size(); i++)
				printf("%d\n", primos[i]);

			for(; i <= b; i++) {
				for(j = 0; j < primos.size(); j++)
					if(i % primos[j] == 0)
						goto ac;

				printf("%d\n", i);
				ac:;
			}
		}
	}
}
```

3.14 KD-TREE PARA QUERIES DE PARES MAIS PRÓXIMOS EM O(LOG(N))

```
typedef pair<int, int> pii;
typedef vector<pii> vpii;

const int maxn = 100000;
int tx[maxn];
int ty[maxn];
bool divX[maxn];

bool cmpX(const pii &a, const pii &b) {
	return a.first < b.first;
}

bool cmpY(const pii &a, const pii &b) {
	return a.second < b.second;
}

void buildTree(int left, int right, pii points[]) {
	if (left >= right)
		return;
	int mid = (left + right) >> 1;

	//sort(points + left, points + right + 1, divX ? cmpX : cmpY);
	int minx = INT_MAX;
	int maxx = INT_MIN;
	int miny = INT_MAX;
	int maxy = INT_MIN;
	for (int i = left; i < right; i++) {
		checkmin(minx, points[i].first);
		checkmax(maxx, points[i].first);
		checkmin(miny, points[i].second);
		checkmax(maxy, points[i].second);
	}
	divX[mid] = (maxx - minx) >= (maxy - miny);
	nth_element(points + left, points + mid, points + right, divX[mid] ? cmpX : cmpY);

	tx[mid] = points[mid].first;
	ty[mid] = points[mid].second;

	if (left + 1 == right)
		return;
	buildTree(left, mid, points);
	buildTree(mid + 1, right, points);
}

long long closestDist;
int closestNode;

void findNearestNeighbour(int left, int right, int x, int y) {
	if (left >= right)
		return;
	int mid = (left + right) >> 1;
	int dx = x - tx[mid];
	int dy = y - ty[mid];
	long long d = dx * (long long) dx + dy * (long long) dy;
	if (closestDist > d && d) {
		closestDist = d;
		closestNode = mid;
	}
```

```
	if (left + 1 == right)
		return;

	int delta = divX[mid] ? dx : dy;
	long long delta2 = delta * (long long) delta;
	int l1 = left;
	int r1 = mid;
	int l2 = mid + 1;
	int r2 = right;
	if (delta > 0)
		swap(l1, l2), swap(r1, r2);

	findNearestNeighbour(l1, r1, x, y);
	if (delta2 < closestDist)
		findNearestNeighbour(l2, r2, x, y);
}

int findNearestNeighbour(int n, int x, int y) {
	closestDist = LLONG_MAX;
	findNearestNeighbour(0, n, x, y);
	return closestNode;
}

int main() {
	vpii p;
	p.push_back(make_pair(0, 2));
	p.push_back(make_pair(0, 3));
	p.push_back(make_pair(-1, 0));

	p.resize(unique(p.begin(), p.end()) - p.begin());

	int n = p.size();
	buildTree(1, 0, n - 1, &(vpii(p)[0]));
	int res = findNearestNeighbour(n, 0, 0);

	cout << p[res].first << " " << p[res].second << endl;

	return 0;
}
```

3.15 Algoritmo de Mo (Queries offline, muitas queries e impossibilidade de SegTree)

```
#define N 311111
#define A 1111111
#define BLOCK 555 // ~sqrt(N)

int cnt[A], a[N], ans[N], answer = 0;

struct node {
	int L, R, i;
}q[N];

bool cmp(node x, node y) {
	if(x.L/BLOCK != y.L/BLOCK) {
		// different blocks, so sort by block.
		return x.L/BLOCK < y.L/BLOCK;
	}
	// same block, so sort by R value
```

```
	return x.R < y.R;
}

void add(int position) {
	cnt[a[position]]++;
	if(cnt[a[position]] == 1) {
		answer++; // Verifica se é resposta aqui!!!
	}
}

void remove(int position) {
	cnt[a[position]]--;
	if(cnt[a[position]] == 0) {
		answer--; // Verifica se é resposta aqui!!!
	}
}

int main() {
	int n;
	scanf("%d", &n);
	for(int i=0; i<n; i++)
		scanf("%d", &a[i]);

	int m;
	scanf("%d", &m);
	for(int i=0; i<m; i++) {
		scanf("%d%d", &q[i].L, &q[i].R);
		q[i].L--; q[i].R--;
		q[i].i = i;
	}

	sort(q, q + m, cmp);

	int currentL = 0, currentR = 0;
	for(int i=0; i<m; i++) {
		int L = q[i].L, R = q[i].R;
		while(currentL < L) {
			remove(currentL);
			currentL++;
		}

		while(currentL > L) {
			add(currentL-1);
			currentL--;
		}
		while(currentR <= R) {
			add(currentR);
			currentR++;
		}
		while(currentR > R+1) {
			remove(currentR-1);
			currentR--;
		}
		ans[q[i].i] = answer;
	}

	for(int i=0; i<m; i++)
		printf("%d\n", ans[i]);
}
```

3.16 FLUXO DE CUSTO MÍNIMO

```
int INFINITO = 1<<30, MAXM = 10000, MAXN = 1000;

struct Node {
    int x, y, cap, cost;
    int next;
} edge[MAXM];

class MinCost {
public:
    int e, head[MAXN], dis[MAXN], pre[MAXN], record[MAXN], inq[MAXN];
    void Addedge(int x, int y, int cap, int cost) {
        edge[e].x = x, edge[e].y = y, edge[e].cap = cap, edge[e].cost = cost;
        edge[e].next = head[x], head[x] = e++;
        edge[e].x = y, edge[e].y = x, edge[e].cap = 0, edge[e].cost = -cost;
        edge[e].next = head[y], head[y] = e++;
    }
    int mincost(int s, int t) {
        int mncost = 0, flow, totflow = 0;
        int i, x, y;
        while(1) {
            memset(dis, 63, sizeof(dis));
            int oo = dis[0];
            dis[s] = 0;
            deque<int> Q;
            Q.push_front(s);
            while(!Q.empty()) {
                x = Q.front(), Q.pop_front();
                inq[x] = 0;
                for(i = head[x]; i != -1; i = edge[i].next) {
                    y = edge[i].y;
                    if(edge[i].cap > 0 && dis[y] > dis[x] + edge[i].cost) {
                        dis[y] = dis[x] + edge[i].cost;
                        pre[y] = x, record[y] = i;
                        if(inq[y] == 0) {
                            inq[y] = 1;
                            if(Q.size() && dis[Q.front()] > dis[y])
                                Q.push_front(y);
                            else
                                Q.push_back(y);
                        }
                    }
                }
            }
            if(dis[t] == INFINITO)
                break;
            flow = INFINITO;
            for(x = t; x != s; x = pre[x]) {
                int ri = record[x];
                flow = min(flow, edge[ri].cap);
            }
            for(x = t; x != s; x = pre[x]) {
                int ri = record[x];
                edge[ri].cap -= flow;
                edge[ri^1].cap += flow;
                edge[ri^1].cost = -edge[ri].cost;
            }
            totflow += flow;
            mncost += dis[t] * flow;
```

```
        }
        return mncost;
    }
    void init(int n) {
        e = 0;
        for (int i = 0; i <= n; i++)
            head[i] = -1;
    }
} g;

// Uso:
// Iniciar um grafo com 200 vértices:
// g.init(200);
// Adicionar aresta entre 1 e 2 com custo 10 e capacidade 5
// g.Addedge(1, 2, 10, 5);
// Achar o fluxo de custo mínimo entre 1 e 2
// g.mincost(1, 2);
```

3.17 TEOREMA DA CONVOLUÇÃO UTILIZANDO TRANSFORMADA RÁPIDA DE FOURRIER

```
// TEOREMA DA CONVOLUÇÃO:
// Podemos fazer a convolução de 2 polinomios utilizando a FFT
// Reduzindo a complexidade de n^2 para n log n
// Definimos a convolução h(f . g) como h[k] = sum_j^k f[j] * g[k - j], exemplo:
// h[0] = f[0] * g[0]
// h[1] = f[0] * g[1] + f[1] * g[0]
// h[2] = f[0] * g[2] + f[1] * g[1] + f[2] * g[0]
// Segundo o teorema da convolução
// h(f . g) = transformada inversa de (transformada (f) * transformada (g))
// onde . é o operador de convolução.
// e * é o operador de multiplicação termo a termo.

// we use the nearest upper power of 2 to 200k
#define MAX_DIST (262144*2)

struct cpx
{
	cpx() {}
	cpx(double aa) :a(aa) {}
	cpx(double aa, double bb) :a(aa), b(bb) {}
	double a;
	double b;
	double modsq(void) const
	{
		return a * a + b * b;
	}
	cpx bar(void) const
	{
		return cpx(a, -b);
	}
};

cpx operator +(cpx a, cpx b)
{
	return cpx(a.a + b.a, a.b + b.b);
}

cpx operator *(cpx a, cpx b)
{
```

```
	return cpx(a.a * b.a - a.b * b.b, a.a * b.b + a.b * b.a);
}

cpx operator /(cpx a, cpx b)
{
	cpx r = a * b.bar();
	return cpx(r.a / b.modsq(), r.b / b.modsq());
}

cpx EXP(double theta)
{
	return cpx(cos(theta), sin(theta));
}

const double two_pi = 4 * acos(0);

cpx distances[MAX_DIST];
cpx fftOut[MAX_DIST];
int maxDist;

// from Stanford's notebook: https://web.stanford.edu/~liszt90/acm/notebook.html
// in:     input array
// out:    output array
// step:   {SET TO 1} (used internally)
// size:   Tamanho do input/output {DEVE SER DA ORDEM DE 2}
// dir:    1 = Transformada, -1 = Transformada inversa
// RESULT: out[k] = \sum_{j=0}^{size - 1} in[j] * exp(dir * 2pi * i * j * k / size)
void FFT(cpx *in, cpx *out, int step, int size, int dir)
{
	if (size < 1) return;
	if (size == 1)
	{
		out[0] = in[0];
		return;
	}
	FFT(in, out, step * 2, size / 2, dir);
	FFT(in + step, out + size / 2, step * 2, size / 2, dir);
	for (int i = 0; i < size / 2; i++)
	{
		cpx even = out[i];
		cpx odd = out[i + size / 2];
		out[i] = even + EXP(dir * two_pi * i / size) * odd;
		out[i + size / 2] = even + EXP(dir * two_pi * (i + size / 2) / size) *
odd;
	}
}


// Exemplo:
// Há um robo que pode disparar bolas em N distâncias diferentes.
// Queremos saber se ele alcança uma distância M com 1 ou 2 tacadas.
// Resolução:
// Podemos definir um vetor distances[MAX_DIST],
// onde a distances[i] = 1 se ele pode tacar até a distancia i
// e distances[i] = 0 caso contrario
// Para ver se o robo acerta com 1 tacada, é trivial.
// Para ver se o robo acerta com 2 tacadas, podemos fazer a convolução de distances com
distances.
// E verificar então a posição distances[m]
```

```
int main()
{
	int N, d;
	cpx cpx_1(1);

	while (cin >> N) {
		maxDist = 0;
		memset(distances, 0, MAX_DIST * sizeof(cpx));

		distances[0] = cpx_1; // permit one shots

		for (int i = 0; i < N; i++) {
			cin >> d;
			if (d > maxDist)
				maxDist = d;
			distances[d] = cpx_1;
		}

		// Find (nearest upper power of 2) * 2. O(log 200k)
		int shiftAmount;
		for (shiftAmount = 0; (maxDist >> shiftAmount) != 0; shiftAmount++);
		maxDist = 1 << (shiftAmount + 1);

		// Aplicaremos o teorema da convolução.

		// fftOut <= transformada de distances
		FFT(distances, fftOut, 1, maxDist, 1);

		// Multiplicação termo a termo de f e g, no caso, f = g = fftOut
		// fftOut *= fftOut
		for (int i = 0; i < maxDist; i++)
			fftOut[i] = fftOut[i] * fftOut[i];

		// transformada inversa da multiplcação termo a termo.
		FFT(fftOut, distances, 1, maxDist, -1);


		cin >> N;
		int total = 0;
		for (int i = 0; i < N; i++) {
			cin >> d;

			// Entra a distancia d
			// e verifica se a parte real da distância é != 0
			if (distances[d].a > 0.01)
				total++;
		}
		cout << total << endl;
	}
	return 0;
}
```

3.18 TRANSFORMADA DE FOURRIER (CÓDIGO MENOR)

```
typedef complex<double> num;
const double pi = acos(-1.0);
const int N = ;
int p[N];
```

```
// DFT se type = 1 e IDFT se i = -1
// Se for multiplicar lembre-se de deixar cada vetor com n>=soma dos graus dos pols
// n tem que ser potencia de 2
void FFT(num v[], num ans[], int n, int type) {
	assert(!(n & (n - 1)));
	int i, sz, o;
	p[0] = 0;
	for (i = 1; i < n; i++) p[i] = (p[i >> 1] >> 1) | ((i & 1) ? (n >> 1) : 0);
	for (i = 0; i < n; i++) ans[i] = v[p[i]];
	for (sz = 1; sz < n; sz <<= 1) {
		const num wn(cos(type * pi / sz), sin(type * pi / sz));
		for (o = 0; o < n; o += (sz << 1)) {
			num w = 1;
			for (i = 0; i < sz; i++) {
				const num u = ans[o + i], t = w * ans[o + sz + i];
				ans[o + i] = u + t;
				ans[o + i + sz] = u - t;
				w *= wn;
			}
		}
	}
	if (type == -1) for (i = 0; i < n; i++) ans[i] /= n;
}
```

3.19 GERAR MATRIZ DE $C\binom{n}{k}$

```
long double combs[MAXN][MAXN];

void buildCombs() {
  combs[0][0] = combs[1][1] = combs[1][0] = 1;
  for (int i = 2; i < MAXN; i++) {
    combs[i][0] = combs[i][i] = 1;
    for (int j = 1; j < i; j++)
      combs[i][j] = combs[i-1][j] + combs[i-1][j-1];
  }
}
```

3.20 SWEEP RADIAL – ROTACIONAR UMA RETA EM UM CONJUNTO DE PONTOS.

```
// Descrição do problema de exemplo:
// O "valor" de um celular pode ser dado por V = w1 * x1 + w2 * x2.
// Dado o as notas x1 e x2 do primeiro celular e tambem as notas de N outros celulares,
// qual a melhor posição possível para ele, considerando vitória em empates
// e a pior posição possível, considerando derrota em empates.
// Modo de resolver:
// Podemos descrever todos (x1,x2) como um ponto (x,y), tomando (x1,x2) do primeiro celular
// como origem.
// A partir de então, sabemos que todos os pontos no primeiro quadrante sempre serão
// melhores que o primeiro
// e todos os pontos no terceiro quadrante serão piores que o primeiro.
// Basta então rotacionar uma reta, que começa de (0,1) até (−1,0),
// Levando todo ponto do quarto quadrante para o segundo com valores (−x,−y)
// e tomando os pontos do segundo quadrante como "evento".
// Onde um evento é, nada mais, que um mapa ordenado de pontos, de ordem anti-horária
// em pares, que falam se o ponto vai entrar ou sair como melhor.
//  - Radial sweep in Q2 quadrant in nlogn.
//  - Sorts events using cross product to avoid dealing with
```

```
//    numeric problems.
#include <map>
#include <iostream>
#include <vector>
#include <algorithm>
#include <cassert>

using namespace std;

struct Point {
	Point(int x = 0, int y = 0) : x(x), y(y) {}

	bool operator<(const Point& o) const {
		// Order points in a quadrant by angle with origin:
		// Uses anti-clockwise order by returning true when the
		// cross product between the points is positive.
		return (x*o.y - y*o.x) > 0;
	}

	/*
	bool operator<=(const Point& o) const {
	return (x*o.y - y*o.x) >= 0;
	}
	*/
	int x, y;
};

pair<int, int> solve(const vector<Point>& points) {
	map<Point, pair<int, int> > events;

	Point begin(0, 1);
	Point end(-1, 0);

	// Add events on the borders to guarantee that we consider them.
	events[begin];
	events[end];

	int superior = 0;  // Number of points in Q1 quadrant.
	int same = 0;      // Number of points in origin.
	int active = 0;    // Number of current points in Q2 and Q4 quadrant better
                                // than origin.

	int best_pos = points.size();
	int worst_pos = 0;

	for (const auto& p : points) {
		if (p.x < 0 && p.y < 0) {}
		else if (p.x > 0 && p.y > 0) superior++;
		else if (p.x == 0 && p.y == 0) same++;
		else if (p.x <= 0 && p.y >= 0) {
			// assert(begin <= Point(p.x, p.y));
			//assert(Point(p.x, p.y) <= end);
			events[Point(p.x, p.y)].first++;
		}
		else if (p.x >= 0 && p.y <= 0) {
			//assert(begin <= Point(-p.x, -p.y));
			//assert(Point(-p.x, -p.y) <= end);
			active++;
			events[Point(-p.x, -p.y)].second++;
		}
```

```
            else assert(false);
      }

      for (const auto& e : events) {
            int tie_best_pos = superior + active - e.second.second;
            int tie_worst_pos = superior + active + e.second.first + same;
            active += e.second.first - e.second.second;

            best_pos = min(best_pos, tie_best_pos);
            worst_pos = max(worst_pos, tie_worst_pos);
      }

      return make_pair(best_pos + 1, worst_pos + 1);
}

// Reads the set of points and centers them around Maria's product.
vector<Point> read() {
      int n, cx, cy;
      cin >> n >> cx >> cy;
      vector<Point> points(n - 1);
      for (Point& p : points) {
            cin >> p.x >> p.y;
            p.x -= cx;
            p.y -= cy;
      }
      return points;
}

int main() {
      auto input = read();
      auto solution = solve(input);

      for (auto& i : input)
            swap(i.x, i.y);

      assert(solution == solve(input));

      cout << solution.first << " " << solution.second << endl;
      return 0;
}
```

4 ALGORITMOS EM JAVA

4.1 2-SAT

```
import java.util.*;
import java.util.stream.Stream;
public class Sat2 {
static void dfs1(List<Integer>[] graph, boolean[] used, List<Integer> order, int u) {
            used[u] = true;
            for (int v : graph[u])
                  if (!used[v])
                        dfs1(graph, used, order, v);
            order.add(u);
      }

      static void dfs2(List<Integer>[] reverseGraph, int[] comp, int u, int color) {
```

```
            comp[u] = color;
            for (int v : reverseGraph[u])
                  if (comp[v] == -1)
                        dfs2(reverseGraph, comp, v, color);
      }

      public static boolean[] solve2Sat(List<Integer>[] graph) {
            int n = graph.length;
            boolean[] used = new boolean[n];
            List<Integer> order = new ArrayList<>();
            for (int i = 0; i < n; ++i)
                  if (!used[i])
                        dfs1(graph, used, order, i);

            List<Integer>[] reverseGraph =
Stream.generate(ArrayList::new).limit(n).toArray(List[]::new);
            for (int i = 0; i < n; i++)
                  for (int j : graph[i])
                        reverseGraph[j].add(i);

            int[] comp = new int[n];
            Arrays.fill(comp, -1);
            for (int i = 0, color = 0; i < n; ++i) {
                  int u = order.get(n - i - 1);
                  if (comp[u] == -1)
                        dfs2(reverseGraph, comp, u, color++);
            }

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

            boolean[] res = new boolean[n / 2];
            for (int i = 0; i < n; i += 2)
                  res[i / 2] = comp[i] > comp[i ^ 1];
            return res;
      }

      public static void main(String[] args) {
            int n = 6;
            List<Integer>[] g =
Stream.generate(ArrayList::new).limit(n).toArray(List[]::new);
            // (a || b) && (b || !c)
            // !a => b
            // !b => a
            // !b => !c
            // c => b
            int a = 0, na = 1, b = 2, nb = 3, c = 4, nc = 5;
            g[na].add(b);
            g[nb].add(a);
            g[nb].add(nc);
            g[c].add(b);

            boolean[] solution = solve2Sat(g);
            System.out.println(Arrays.toString(solution));
      }
}
```

4.2 SIMPLEX

```
import java.util.*;

public class Simplex {
	// returns max c*x such that A*x <= b, x >= 0
	public static Rational simplex(Rational[][] A, Rational[] b, Rational[] c,
Rational[] x) {
		int m = A.length;
		int n = A[0].length + 1;
		int[] index = new int[n + m];
		for (int i = 0; i < n + m; i++) {
			index[i] = i;
		}
		Rational[][] a = new Rational[m + 2][n + 1];
		for (Rational[] a1 : a) {
			Arrays.fill(a1, Rational.ZERO);
		}
		int L = m;
		for (int i = 0; i < m; i++) {
			for (int j = 0; j < n - 1; j++) {
				a[i][j] = A[i][j].negate();
			}
			a[i][n - 1] = Rational.ONE;
			a[i][n] = b[i];
			if (a[L][n].compareTo(a[i][n]) > 0) {
				L = i;
			}
		}
		for (int j = 0; j < n - 1; j++) {
			a[m][j] = c[j];
		}
		a[m + 1][n - 1] = Rational.ONE.negate();
		for (int E = n - 1;;) {
			if (L < m) {
				int t = index[E];
				index[E] = index[L + n];
				index[L + n] = t;
				a[L][E] = a[L][E].inverse();
				for (int j = 0; j <= n; j++) {
					if (j != E) {
						a[L][j] = a[L][j].mul(a[L][E].negate());
					}
				}
				for (int i = 0; i <= m + 1; i++) {
					if (i != L) {
						for (int j = 0; j <= n; j++) {
							if (j != E) {
								a[i][j] =
a[i][j].add(a[L][j].mul(a[i][E]));
							}
						}
						a[i][E] = a[i][E].mul(a[L][E]);
					}
				}
			}
			E = -1;
			for (int j = 0; j < n; j++) {
				if (E < 0 || index[E] > index[j]) {
```

```
                                    if (a[m + 1][j].signum() > 0 || a[m + 1][j].signum() ==
0 && a[m][j].signum() > 0) {
                                          E = j;
                                    }
                              }
                        }
                        if (E < 0) {
                              break;
                        }
                        L = -1;
                        for (int i = 0; i < m; i++) {
                              if (a[i][E].signum() < 0) {
                                    Rational d;
                                    if (L < 0 || (d =
a[L][n].div(a[L][E]).sub(a[i][n].div(a[i][E]))).signum() < 0 || d.signum() == 0
                                                && index[L + n] > index[i + n]) {
                                          L = i;
                                    }
                              }
                        }
                        if (L < 0) {
                              return Rational.POSITIVE_INFINITY;
                        }
                  }
                  if (a[m + 1][n].signum() < 0) {
                        return null;
                  }
                  if (x != null) {
                        Arrays.fill(x, Rational.ZERO);
                        for (int i = 0; i < m; i++)
                              if (index[n + i] < n - 1)
                                    x[index[n + i]] = a[i][n];
                  }
                  return a[m][n];
            }

            // Usage example
            public static void main(String[] args) {
                  long[][] a = { { 4, -1 }, { 2, 1 }, { -5, 2 } };
                  long[] b = { 8, 10, 2 };
                  long[] c = { 1, 1 };
                  Rational[] x = new Rational[c.length];
                  Rational res = simplex(cnv(a), cnv(b), cnv(c), x);

                  System.out.println(new Rational(8).equals(res));
                  System.out.println(Arrays.toString(x));

                  a = new long[][] { { 3, 4, -3 }, { 5, -4, -3 }, { 7, 4, 11 } };
                  b = new long[] { 23, 10, 30 };
                  c = new long[] { -1, 1, 2 };
                  x = new Rational[c.length];
                  res = simplex(cnv(a), cnv(b), cnv(c), x);
                  System.out.println(new Rational(57, 8).equals(res));
                  System.out.println(Arrays.toString(x));

                  // no feasible non-negative solutions
                  a = new long[][] { { 4, -1 }, { 2, 1 }, { -5, 2 } };
                  b = new long[] { 8, -10, 2 };
                  c = new long[] { 1, 1 };
                  res = simplex(cnv(a), cnv(b), cnv(c), null);
```

```
		System.out.println(null == res);

		a = new long[][] { { -4, 1 }, { -2, -1 }, { -5, 2 } };
		b = new long[] { -8, -10, 2 };
		c = new long[] { 1, 1 };
		res = simplex(cnv(a), cnv(b), cnv(c), null);
		System.out.println(Rational.POSITIVE_INFINITY == res);

		// no feasible solutions
		a = new long[][] { { 1 }, { -1 } };
		b = new long[] { 1, -2 };
		c = new long[] { 0 };
		res = simplex(cnv(a), cnv(b), cnv(c), null);
		System.out.println(null == res);

		// infinite solutions, but only one is returned
		a = new long[][] { { 1, 1 } };
		b = new long[] { 0 };
		c = new long[] { 1, 1 };
		x = new Rational[c.length];
		res = simplex(cnv(a), cnv(b), cnv(c), x);
		System.out.println(Arrays.toString(x));
	}

	static Rational[] cnv(long[] a) {
		Rational[] res = new Rational[a.length];
		for (int i = 0; i < a.length; i++) {
			res[i] = new Rational(a[i]);
		}
		return res;
	}

	static Rational[][] cnv(long[][] a) {
		Rational[][] res = new Rational[a.length][];
		for (int i = 0; i < a.length; i++) {
			res[i] = cnv(a[i]);
		}
		return res;
	}
}
```

4.3 SegTree 2D

```
import java.util.*;

public class SegmentTree2D {
	public static int max(int[][] t, int x1, int y1, int x2, int y2) {
		int n = t.length >> 1;
		x1 += n;
		x2 += n;
		int m = t[0].length >> 1;
		y1 += m;
		y2 += m;
		int res = Integer.MIN_VALUE;
	for (int lx = x1, rx = x2; lx <= rx; lx = (lx + 1) >> 1, rx = (rx - 1) >> 1)
	   for (int ly = y1, ry = y2; ly <= ry; ly = (ly + 1) >> 1, ry = (ry - 1) >> 1) {
			if ((lx & 1) != 0 && (ly & 1) != 0) res = Math.max(res, t[lx][ly]);
			if ((lx & 1) != 0 && (ry & 1) == 0) res = Math.max(res, t[lx][ry]);
```

```
                if ((rx & 1) == 0 && (ly & 1) != 0) res = Math.max(res, t[rx][ly]);
                if ((rx & 1) == 0 && (ry & 1) == 0) res = Math.max(res, t[rx][ry]);
            }
            return res;
        }

        public static void add(int[][] t, int x, int y, int value) {
            x += t.length >> 1;
            y += t[0].length >> 1;
            t[x][y] += value;
            for (int tx = x; tx > 0; tx >>= 1)
                for (int ty = y; ty > 0; ty >>= 1) {
                    if (tx > 1) t[tx >> 1][ty] = Math.max(t[tx][ty], t[tx ^ 1][ty]);
                    if (ty > 1) t[tx][ty >> 1] = Math.max(t[tx][ty], t[tx][ty ^ 1]);
                }
        }

        public static void main(String[] args) {
            Random rnd = new Random(1);
            for (int step1 = 0; step1 < 1000; step1++) {
                int sx = rnd.nextInt(10) + 1;
                int sy = rnd.nextInt(10) + 1;
                int[][] t = new int[sx * 2][sy * 2];
                int[][] tt = new int[sx][sy];
                for (int step2 = 0; step2 < 1000; step2++) {
                    if (rnd.nextBoolean()) {
                        int x = rnd.nextInt(sx);
                        int y = rnd.nextInt(sy);
                        int v = rnd.nextInt(10) - 5;
                        add(t, x, y, v);
                        tt[x][y] += v;
                    } else {
                        int x2 = rnd.nextInt(sx);
                        int x1 = rnd.nextInt(x2 + 1);
                        int y2 = rnd.nextInt(sy);
                        int y1 = rnd.nextInt(y2 + 1);
                        int resp = max(t, x1, y1, x2, y2);
                    }
                }
            }
        }
    }
```

5 BIBLIOTECAS E ESTRUTURAS

5.1 GEOMETRIA COMPUTACIONAL REDUZIDO

```
typedef pair<double, double> Ponto;

bool cw(Ponto a, Ponto b, Ponto c) {
    return (b.first - a.first) * (c.second - a.second) - (b.second - a.second) *
(c.first - a.first) < 0;
}

// Retorna o casco convexo do conjunto de pontos p
vector<Ponto> convexHull(vector<Ponto> p) {
    int n = p.size();
    if (n <= 1)
        return p;
    int k = 0;
    sort(p.begin(), p.end());
    vector<Ponto> q(n * 2);
    for (int i = 0; i < n; q[k++] = p[i++])
        for (; k >= 2 && !cw(q[k - 2], q[k - 1], p[i]); --k)
            ;
    for (int i = n - 2, t = k; i >= 0; q[k++] = p[i--])
        for (; k > t && !cw(q[k - 2], q[k - 1], p[i]); --k)
            ;
    q.resize(k - 1 - (q[0] == q[1]));
    return q;
}

//O dobro da área definida pelo triangulo de pontos pontos a, b e c (sem sinal).
double uArea2(Ponto a, Ponto b, Ponto c) {
    return abs((b.first - a.first) * (c.second - a.second) - (b.second - a.second) *
(c.first - a.first));
}

//O dobro da área definida pelo triangulo de pontos pontos a, b e c (com sinal).
double area2(Ponto a, Ponto b, Ponto c) {
    return (b.first - a.first) * (c.second - a.second) - (b.second - a.second) *
(c.first - a.first);
}

//Distância entre os pontos a e b
double dist(Ponto a, Ponto b) {
    return hypot(a.first - b.first, a.second - b.second);
}

//Interseção de semi-retas (p1 -> p2), (p3 -> p4)
bool segIntercept(Ponto p1, Ponto p2, Ponto p3, Ponto p4) {
      return cw(p1, p2, p3) != cw(p1, p2, p4) && cw(p3, p4, p1) != cw(p3, p4, p2);
}

//Retorna a área do polígono p
double polygonArea(vector<Ponto> p) {
    double s = 0.0;
    for (int i = 0; i < p.size(); i++ )
        s += area2(Ponto(0,0), p[i], p[( i + 1 ) % p.size()]);
    return fabs(s / 2.0);
}
```

```
//Retorna a área do polígono p definido pelos pontos p[i, f]
double polygonArea2(vector<Ponto> p, int i, int f) {
    double s = 0.0;
    Ponto primeiro = p[i];
    for (; i != f; i++)
        s += area2(Ponto(0,0), p[i], p[( i + 1 )]);
    s += area2(Ponto(0,0), p[i], primeiro);
    return fabs(s / 2.0);
}

//Retorna a menor largura do conjunto de pontos p
double raio(vector<Ponto> p) {
    vector<Ponto> h = convexHull(p);
    int m = h.size();
    if (m == 1)
        return 0;
    if (m == 2)
        return 0;
    int k = 1;
    while (uArea2(h[m - 1], h[0], h[(k + 1) % m]) > uArea2(h[m - 1], h[0], h[k]))
        ++k;
    double res = 10000000;
    for (int i = 0, j = k; i <= k && j < m; i++) {
        res = min(res, dist(h[i], h[j]));
        while (j < m && uArea2(h[i], h[(i + 1) % m], h[(j + 1) % m]) > uArea2(h[i],
h[(i + 1) % m], h[j])) {
            res = min(res, dist(h[i], h[(j + 1) % m]));
            ++j;
        }
    }
    return res;
}

//Retorna a maior largura do conjunto de pontos p
double diametro(vector<Ponto> p) {
    vector<Ponto> h = convexHull(p);
    int m = h.size();
    if (m == 1)
        return 0;
    if (m == 2)
        return dist(h[0], h[1]);
    int k = 1;
    while (uArea2(h[m - 1], h[0], h[(k + 1) % m]) > uArea2(h[m - 1], h[0], h[k]))
        ++k;
    double res = 0;
    for (int i = 0, j = k; i <= k && j < m; i++) {
        res = max(res, dist(h[i], h[j]));
        while (j < m && uArea2(h[i], h[(i + 1) % m], h[(j + 1) % m]) > uArea2(h[i],
h[(i + 1) % m], h[j])) {
            res = max(res, dist(h[i], h[(j + 1) % m]));
            ++j;
        }
    }
    return res;
}
```

5.2 GEOMETRIA COMPUTACIONAL COMPLETO

```
const double EPS = 1e-10;

inline int cmp( double x, double y = 0, double tol = EPS ) {
    return (x <= y + tol ) ? ( x + tol < y ) ? -1 : 0 : 1;
}

struct Point {
    double x, y;

    Point(double x = 0, double y = 0) : x(x), y(y) {}

    Point operator+(Point q) const {
        return Point( x + q.x, y + q.y );
    }

    Point operator-(Point q) const {
        return Point( x - q.x, y - q.y );
    }

    Point operator*(double t) const {
        return Point( x * t, y * t );
    }

    Point operator/(double t) const {
        return Point( x / t, y / t );
    }

    double operator*(Point q)const {
        return x * q.x + y * q.y;
    }

    double operator^( Point q ) const {
        return x * q.y - y * q.x;
    }

    int cmp( Point q ) const {
        if (int t = ::cmp(x, q.x))
                    return t;
        return ::cmp(y, q.y);
    }

    bool operator==(Point q) const {
        return cmp(q) == 0;
    }

    bool operator!=(Point q) const {
        return cmp(q) != 0;
    }

    bool operator<(Point q) const {
        return cmp(q) < 0;
    }

    static Point pivot;
};
```

```
Point Point::pivot;

typedef vector<Point> Polygon;

inline double abs( Point& p ) {
    return hypot( p.x, p.y );
}

inline double arg( Point& p ) {
    return atan2( p.y, p.x );
}

//Verifica o sinal do produto vetorial entre os vetores (p-r) e (q - r)
inline int ccw( Point& p, Point& q, Point& r ) {
    return cmp( ( p - r ) ^ ( q - r ) );
}

//calcula o angulo orientado entre os vetores (p-q) e (r - q)
inline double angle( Point& p, Point &q, Point& r ) {
    Point u = p - q, w = r - q;
    return atan2( u ^ w, u * w );
}

//Decide se o ponto p esta sobre a reta que passa por p1p2.
bool pontoSobreReta( Point& p1, Point &p, Point& p2 ) {
    return ccw( p1, p2, p ) == 0;
}

//Decide de p esta sobre o segmento p1p2
bool between( Point& p1, Point &p, Point& p2 ) {
    return ccw( p1, p2, p ) == 0 && cmp( ( p1 - p ) * ( p2 - p ) ) <= 0;
}

//Calcula a distancia do ponto p a reta que passa por p1p2
double retaDistance( Point& p1, Point& p2, Point &p ) {
    Point A = p1 - p, B = p2 - p1;
    return fabs( A ^ B ) / sqrt( B * B );
}

//Calcula a distancia do ponto p ao segmento de reta que passa por p1p2
double segDistance( Point& p1, Point& p2, Point &p ) {
    Point A = p1 - p, B = p1 - p2, C = p2 - p;
    double a = A * A, b = B * B, c = C * C;
    if ( cmp( a, b + c ) >= 0 ) return sqrt( c );
    if ( cmp( c, a + b ) >= 0 )return sqrt( a );
    return fabs( A ^ C ) / sqrt( b );

}


//Calcula a area orientada do poligono T.
double polygonArea( Polygon& T ) {
    double s = 0.0;
    int n = T.size( );
    for ( int i = 0; i < n; i++ )
    {
        s += T[i] ^ T[( i + 1 ) % n];
    }
    return s / 2.0; //Retorna a area com sinal
}
```

```
//Classifica o ponto p em relacao ao poligono T dependendo se ele está
//na fronteira (-1) no exterior (0) ou no interior (1).
int inpoly( Point& p, Polygon& T ) {
    //-1 sobre, 0 fora, 1 dentro
    double a = 0.0;
    int n = T.size( );
    for ( int i = 0; i < n; i++ )
    {
        if ( between( T[i], p, T[( i + 1 ) % n] ) ) return -1;
        a += angle( T[i], p, T[( i + 1 ) % n] );
    }
    return cmp( a ) != 0;
}

//Ordenacao radial.
bool radialSort( Point p, Point q ) {
    Point P = p - Point::pivot, Q = q - Point::pivot;
    double R = P ^ Q;
    if ( cmp( R ) ) return R > 0;
    return cmp( P * P, Q * Q ) < 0;
}

//Determina o convex hull de T. ATENCAO. A lista de pontos T e destruida.
Polygon convexHull( vector<Point>& T ) {
    int j = 0, k, n = T.size( );
    Polygon U( n );
    Point::pivot = *min_element( T.begin( ), T.end( ) );
    sort( T.begin( ), T.end( ), radialSort );

    for ( k = n - 2; k >= 0 && ccw( T[0], T[n - 1], T[k] ) == 0; k-- );
    reverse( ( k + 1 ) + T.begin( ), T.end( ) );

    for ( int i = 0; i < n; i++ ) {
        // troque o >= por > para manter pontos colineares
        while (j > 1 && ccw( U[j - 1], U[j - 2], T[i] ) >= 0)
             j--;
        U[j++] = T[i];
    }
    U.resize( j );
    return U;
}

//Interseção de semi-retas (p1 -> p2), (p3 -> p4)
bool segIntercept(Point p1, Point p2, Point p3, Point p4) {
      return ccw(p1, p2, p3) != ccw(p1, p2, p4) && ccw(p3, p4, p1) != ccw(p3, p4, p2);
}
```

5.3 GRAFOS

```
struct tVertice {
	int id, pai;
	long long int dist;

	tVertice(int a, long long int b, int p = -1) {
		id = a;
		dist = b;
		pai = p;
	}

	bool operator<(tVertice a) const {
		return a.dist < dist;
	}
};

struct tGrafo {
	vector<tVertice> *Grafo;
	int n, *pais;

	tGrafo(int a) {
		Grafo = new vector<tVertice>[a];
		pais = new int[a];
		n = a;
	}

	~tGrafo() {
		limpa();
		delete [] pais;
		delete [] Grafo;
	}

	void operator=(tGrafo const &a) {
		limpa();

		for(int i = 0; i < n; i++)
			for(int j = 0; j < a.Grafo[i].size(); j++)
				Grafo[i].push_back(a.Grafo[i][j]);
	}

	void limpa() {
		for(int i = 0; i < n; i++)
			Grafo[i].clear();
	}

	void addAresta(int a, int b, long long int v = 0) {
		Grafo[a].push_back(tVertice(b, v));
	}

	void removeAresta(int a, int b) {
		for(int i = 0; i < Grafo[a].size(); i++)
			if(Grafo[a][i].id == b) {
				Grafo[a].erase(Grafo[a].begin() + i);
				return;
			}
	}
```

```
	long long int Dijkstra(int inicio, int fim) {
		priority_queue<tVertice> fila;
		bool visitados[n];
		int i;

		for(i = 0; i < n; i++)
			pais[i] = visitados[i] = 0;

		fila.push(tVertice(inicio, 0));

		while(fila.top().id != fim && !fila.empty()) {
			if(!visitados[fila.top().id]) {
				for(i = 0; i < Grafo[fila.top().id].size(); i++)
					if(!visitados[Grafo[fila.top().id][i].id])
fila.push(tVertice(Grafo[fila.top().id][i].id,
				fila.top().dist + Grafo[fila.top().id][i].dist, fila.top().id));

				visitados[fila.top().id] = 1;
				pais[fila.top().id] = fila.top().pai;
			}

			fila.pop();
		}

		if(fila.empty())
			return -1;

		pais[fila.top().id] = fila.top().pai;

		return fila.top().dist;
	}

// Retorna a MST, caso queira retornar o custo, basta retornar a variável custo.
	tGrafo Prim(int inicio) {
		priority_queue<tVertice> fila;
		tGrafo G2(n);
		bool visitados[n];
		int i, custo = 0;

		fila.push(tVertice(inicio, 0));

		while(!fila.empty()) {
			if(!visitados[fila.top().id]) {
				for(i = 0; i < Grafo[fila.top().id].size(); i++)
					if(!visitados[Grafo[fila.top().id][i].id])
fila.push(tVertice(Grafo[fila.top().id][i].id,
			Grafo[fila.top().id][i].dist,
			fila.top().id));

				custo += fila.top().dist;
				visitados[fila.top().id] = 1;
				G2.addAresta(fila.top().pai, fila.top().id);
			}

			fila.pop();
		}

		return G2;
	}
```

```
	long long int buscaLargura(int inicio, int fim) {
		queue<tVertice> fila;
		bool visitados[n];
		int i;

		for(i = 0; i < n; i++)
			visitados[i] = pais[i] = 0;

		fila.push(tVertice(inicio, 0));

		while(fila.front().id != fim && !fila.empty()) {
			if(!visitados[fila.front().id]) {
				for(i = 0; i < Grafo[fila.front().id].size(); i++)
					if(!visitados[Grafo[fila.front().id][i].id])
fila.push(tVertice(Grafo[fila.front().id][i].id,
				fila.front().dist + 1,
				fila.front().id));

					visitados[fila.front().id] = 1;
					pais[fila.front().id] = fila.front().pai;
				}

				fila.pop();
			}

			if(fila.empty())
				return -1;

			pais[fila.front().id] = fila.front().pai;

			return fila.front().dist;
		}

		long long int fordFulkerson(int s, int t) {
			int u, v;
			long long int max_flow = 0;
			tGrafo G2(n);
			G2 = *this;

			while (G2.buscaLargura(s, t) >= 0) {
				long long int path_flow = 10000000000;

				for (v = t; v != s; v = G2.pais[v]) {
					u = G2.pais[v];
					path_flow = min(path_flow, G2.valAresta(u, v));
				}

				for (v = t; v != s; v = G2.pais[v]) {
					u = G2.pais[v];
					G2.modificaAresta(u, v, -path_flow);
					G2.modificaAresta(v, u, path_flow);
				}

				max_flow += path_flow;
			}

			return max_flow;
		}
```

```
	long long int valAresta(int a, int b) {
		for(int i = 0; i < Grafo[a].size(); i++)
			if(Grafo[a][i].id == b)
				return Grafo[a][i].dist;
		return -1;
	}

	void modificaAresta(int a, int b, long long int v) {
		for(int i = 0; i < Grafo[a].size(); i++)
			if(Grafo[a][i].id == b) {
				Grafo[a][i].dist += v;

				if(Grafo[a][i].dist == 0)
					Grafo[a].erase(Grafo[a].begin() + i);

				return;
			}
		addAresta(a, b, v);
	}

	bool existeAresta(int a, int b) {
		for(int i = 0; i < Grafo[a].size(); i++)
			if(Grafo[a][i].id == b)
				return true;
		return false;
	}
};
```

5.4 SEGTREE COM LAZY PROPAGATION

```
#define MAX 1000000 // O valor aqui tem que ser >= 4 * tamanho do maior n
#define ELEMENTO_NEUTRO 0

int init[MAX], tree[MAX], lazy[MAX];

void build_tree(int node, int a, int b) {
    if(a > b)
        return;

    if(a == b) {
        tree[node] = init[a];
        lazy[node] = ELEMENTO_NEUTRO;
        return;
    }

    build_tree(node*2, a, (a+b)/2);
    build_tree(node*2+1, 1+(a+b)/2, b);

//Atualização do pai - verificar operação
    tree[node] = tree[node*2] + tree[node*2+1];
    lazy[node] = ELEMENTO_NEUTRO;
}

void update_tree(int node, int a, int b, int i, int j, int value) {
//Atualização atrasada - verificar operação
    if(lazy[node] != ELEMENTO_NEUTRO) {
        tree[node] += lazy[node];

        if(a != b) {
            lazy[node*2] += lazy[node];
            lazy[node*2+1] += lazy[node];
        }

        lazy[node] = ELEMENTO_NEUTRO;
    }

    if(a > b || a > j || b < i)
        return;

//Atualização do nó - verificar operação
    if(a >= i && b <= j) {
        tree[node] += value;

        if(a != b) {
            lazy[node*2] += value;
            lazy[node*2+1] += value;
        }

        return;
    }

    update_tree(node*2, a, (a+b)/2, i, j, value);
    update_tree(1+node*2, 1+(a+b)/2, b, i, j, value);

//Atualização do pai - verificar operação
    tree[node] = tree[node*2] + tree[node*2+1];
}
```

```
int query_tree(int node, int a, int b, int i, int j) {
    if(a > b || a > j || b < i) {
        return ELEMENTO_NEUTRO;
    }

//Atualização atrasada - verificar operação
    if(lazy[node] != ELEMENTO_NEUTRO) {
        tree[node] += lazy[node];

        if(a != b) {
            lazy[node*2] += lazy[node];
            lazy[node*2+1] += lazy[node];
        }

        lazy[node] = ELEMENTO_NEUTRO;
    }

    if(a >= i && b <= j)
        return tree[node];

    int q1 = query_tree(node*2, a, (a+b)/2, i, j);
    int q2 = query_tree(1+node*2, 1+(a+b)/2, b, i, j);

//Retorno da arvore - verificar operação
    return q1 + q2;
}
```

Uso:

Assumindo que "n" é o numero de termos que o segmento tem
Inicialize "init" com os valores iniciais, por exemplo:

- `for(i = 0; i < n; scanf("%d", val), init[i] = val, i++);`

E mande construir a arvore:

- `build_tree(1, 0, n-1);`

Para atualizar a arvore:

- `update_tree(1, 0, n-1, inicio, fim, val);`

Onde inicio é a posição inicial do segmento desejado e fim é a posição final do mesmo e val é o quanto você quer alterar os valores desse seguimento

Para fazer queries:

- `query_tree(1, 0, n-1, inicio, fim);`

Onde inicio é a posição inicial do segmento desejado e fim é a posição final do mesmo. O retorno terá o mesmo tipo que os dados guardados na arvore e será o resultado do segmento pesquisado.

5.5 MATRIZES

```
typedef vector<int> vi;
typedef vector<vi> vvi;
const int mod = 1234567891;

// Retorna a matriz I_n
vvi matrixUnit(int n) {
    vvi res(n, vi(n));
    for (int i = 0; i < n; i++)
        res[i][i] = 1;
    return res;
}

// Retorna a + b
vvi matrixAdd(const vvi &a, const vvi &b) {
    int n = a.size();
    int m = a[0].size();
    vvi res(n, vi(m));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            res[i][j] = (a[i][j] + b[i][j]) % mod;
    return res;
}

// Retorna a * b
vvi matrixMul(const vvi &a, const vvi &b) {
    int n = a.size();
    int m = a[0].size();
    int k = b[0].size();
    vvi res(n, vi(k));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < k; j++)
            for (int p = 0; p < m; p++)
               res[i][j] = (res[i][j] + (long long) a[i][p] * b[p][j]) % mod;
    return res;
}

// Retorna a matriz a^p
vvi matrixPow(const vvi &a, int p) {
    if (p == 0)
        return matrixUnit(a.size());
    if (p & 1)
        return matrixMul(a, matrixPow(a, p - 1));
    return matrixPow(matrixMul(a, a), p / 2);
}

// Retorna sum_{i=1}^{p} a^i
vvi matrixPowSum(const vvi &a, int p) {
    int n = a.size();
    if (p == 0)
        return vvi(n, vi(n));
    if (p % 2 == 0)
        return matrixMul(matrixPowSum(a, p / 2), matrixAdd(matrixUnit(n),
matrixPow(a, p / 2)));
    return matrixAdd(a, matrixMul(matrixPowSum(a, p - 1), a));
}

// Criar uma matriz a[3][2] → vvi a(3, vi(2));
```

5.6 UNION-FIND COM RANKING

Obtêm o conjunto disjunto dos pontos, porém, sempre ligando a arvore menor à arvore maior.

```
// Tamanho máximo de n
const int maxn = 200000;
int Rank[maxn], p[maxn], n;

void init(int _n) {
    n = _n;
    fill(Rank, Rank + n, 0);
    for (int i = 0; i < n; i++) p[i] = i;
}

int find(int x) {
    return x == p[x] ? x : (p[x] = find(p[x]));
}

void unir(int a, int b) {
    a = find(a);
    b = find(b);
    if (a == b) return;
    if (Rank[a] < Rank[b]) swap(a, b);
    if (Rank[a] == Rank[b]) ++Rank[a];
    p[b] = a;
}
```

5.7 WAVELET-TREE

```
#include <iostream>
#include <algorithm>
using namespace std;

const int N = 10000;

struct KthSmallest {
	struct Seg {
		int l, r, mid;
		void set(int _l, int _r) {
			l = _l; r = _r;
			mid = l + r >> 1;
		}
	} seg[N << 2];
	int b[25][N], left[25][N], sorted[N];
	void init(int *a, int n) {
		for (int i = 0; i < n; ++i) b[0][i] = sorted[i] = a[i];
		sort(sorted, sorted + n);
		build(0, n, 0, 1);
	}
	void build(int l, int r, int d, int idx) {
		seg[idx].set(l, r);
		if (l + 1 == r) return;
		int mid = seg[idx].mid;
		int lsame = mid - l;
		for (int i = l; i < r; ++i)
			if (b[d][i] < sorted[mid])
				lsame--;
		int lpos = l, rpos = mid, same = 0;

		for (int i = l; i < r; ++i) {
			left[d][i] = (i != l ? left[d][i - 1] : 0);
			if (b[d][i] < sorted[mid]) {
				left[d][i]++;
				b[d + 1][lpos++] = b[d][i];
			}
			else if (b[d][i] > sorted[mid]) {
				b[d + 1][rpos++] = b[d][i];
			}
			else {
				if (same < lsame) {
					same++;
					left[d][i]++;
					b[d + 1][lpos++] = b[d][i];
				}
				else {
					b[d + 1][rpos++] = b[d][i];
				}
			}
		}
		build(l, mid, d + 1, idx << 1);
		build(mid, r, d + 1, idx << 1 | 1);
	}

	//Quando ordernarmos [l, r), qual é o k-ésimo termo?.
	int kth(int l, int r, int k, int d = 0, int idx = 1) { // k : 1-origin!!!
		if (l + 1 == r) return b[d][l];
		int ltl = (l != seg[idx].l ? left[d][l - 1] : 0);
```

```
            int tl = left[d][r - 1] - ltl;

            if (tl >= k) {
                  int newl = seg[idx].l + ltl;
                  int newr = seg[idx].l + ltl + tl;
                  return kth(newl, newr, k, d + 1, idx << 1);
            }
            else {
                  int mid = seg[idx].mid;
                  int tr = r - l - tl;
                  int ltr = l - seg[idx].l - ltl;
                  int newl = mid + ltr;
                  int newr = mid + ltr + tr;
                  return kth(newl, newr, k - tl, d + 1, idx << 1 | 1);
            }
      }

      //When sorting [l, r), what number will x come in?
      //If there are two or more x's, return the rank of the last one.
      //If there is no x, return the rank of the largest but less than x.
      //When there is no less than x, 0 is returned.
      int rank(int l, int r, int x, int d = 0, int idx = 1) {
            if (seg[idx].l + 1 == seg[idx].r) {
                  return l + 1 == r&&sorted[l] <= x;
            }
            int ltl = (l != seg[idx].l ? left[d][l - 1] : 0);
            int tl = left[d][r - 1] - ltl;
            int mid = seg[idx].mid;
            if (x < sorted[mid]) {
                  int newl = seg[idx].l + ltl;
                  int newr = seg[idx].l + ltl + tl;
                  return rank(newl, newr, x, d + 1, idx << 1);
            }
            else {
                  int tr = r - l - tl;
                  int ltr = l - seg[idx].l - ltl;
                  int newl = mid + ltr;
                  int newr = mid + ltr + tr;
                  return tl + rank(newl, newr, x, d + 1, idx << 1 | 1);
            }
      }

      // Quantos x existem entre [l,r)
      int freq(int l, int r, int x) {
            return rank(l, r, x) - rank(l, r, x - 1);
      }
} kth;

int main() {
      int a[8] = { 6,12,5,17,10,2,7,3 };
      kth.init(a, 8);
      cout << kth.kth(2, 7, 3) << endl; // 7
      cout << kth.rank(2, 7, 7) << endl; // 3
}
```

6 BIBLIOTECAS STD

6.1 VECTOR

Funções:

`begin()`	Retorna um iterador para o inicio **do** vector
`end()`	Retorna um iterador para o fim **do** vector
`size()`	Retorna o número de elementos contidos no vector
`push_back(T)`	Insere o termo t no vector
`clear()`	Remove todos os termos **do** vector
`pop_back()`	Remove o último termo **do** vector
`swap(vector)`	Troca todos elementos com o vector recebido
`erase(it)`	Remove o termo apontado pelo iterador passado
`erase(it b, it e)`	Remove os termos entre **[**b, e**)**

6.2 MAP

Funções:

`begin()`	Retorna um iterador para o inicio **do** map
`end()`	Retorna um iterador para o fim **do** map
`size()`	Retorna o número de elementos contidos no map
`insert(make_par(k, v))`	Insere o termo v no map com a chave k
`find(k)`	Procura pelo termo com a chave k e retorna um iterador
`lower_bound(k)`	Retorna um iterador para o menor termo até k, se k existir, retorna k
`upper_bound(k)`	Retorna um iterador para o próximo termo após k, mesmo se k existir

6.3 QUEUE / PRIORITY_QUEUE

Funções:

`front()`	Retorna um iterador para o próximo termo da fila
`back()`	Retorna um iterador para o último termo da fila **(**somente queue**)**
`push(T)`	Adiciona o termo T a fila
`pop()`	Remove o próximo termo da fila
`size()`	Retorna quantos elementos existem na fila
`empty()`	Retorna um booleano informando se a lista está vazia

Problema	Quem leu	Dificuldade	Escrito	Resumo
A				
B				
C				
D				
E				
F				
G				
H				
I				
J				
K				
L				