

Caderno de Maratona de Programação

Fernando Kiotheka

Universidade Federal do Paraná

29 de junho de 2024

Sumário

1 Teoria	2		
1.1 Limites numéricos	2	4.11 Árvore geradora mínima – Prim	9
1.2 Comparações numéricas	2	4.12 Caminho euleriano	9
1.3 Identidades binomiais	2	4.13 Menor no maior	9
1.4 Teorema de Lucas	2	4.14 Decomposição pesado-leve	10
1.5 Teorema do Chicken McNugget	3	4.15 Decomposição pesado-leve – K-ésimo	10
1.6 Triplas pitagóricas	3	4.16 Fluxo máximo	10
1.7 Razões trigonométricas	3	4.17 Fluxo de custo mínimo	10
1.8 Soma de ângulos	3	4.18 Fluxo máximo Dinic	11
1.9 Grafos planares	3	4.19 Stoer-Wagner	11
1.10 Corte mínimo	3	4.20 Ascensão binária	11
1.11 Fatorial e desarranjos	3	4.21 Menor anc. comum – Ancensão binária	11
1.12 Teorema binomial	3	4.22 Empar. máximo bipartido – Kuhn	12
1.13 Círculo inscrito em triângulo	3	4.23 Empar. generalizado – Blossom	12
1.14 Lei dos senos	3	4.24 Emparelhamento estável	12
1.15 Lei dos cossenos	3	4.25 Decomposição centróide	12
1.16 Truque da distância Manhattan	3	4.26 Menor anc. comum – Offline	12
1.17 Números de Catalão	4	4.27 Decomposição raiz quadrada de árvore	13
1.18 Fórmula de Heron	4	4.28 Menor anc. comum – Dec. raiz quad.	13
1.19 Progressões	4		
1.20 Teoremas de Fermat	4	5 Matemática	13
1.21 Módulo no expoente	4	5.1 Operações comuns	13
1.22 Séries	4	5.2 Inteiro modular	13
1.23 Alguns primos	4	5.3 Transformação de Fourier Modular	13
		5.4 Matriz	13
2 C++	4	5.5 Distribuição binomial	13
2.1 Esboço	4	5.6 Integração pelo método de Simpson	13
2.2 Teste de estresse	4	5.7 Sequência de Bruijn	14
2.3 Hash da entrada padrão	4	5.8 PD de dígito	14
2.4 vimrc	4	5.9 Primeiros dígitos de n^k	14
2.5 Gerador	4	5.10 Problema de Josephus	14
		5.11 2-SAT	14
3 Estrutura de Dados	5	5.12 Multiplicação – Karatsuba	14
3.1 União-busca	5	5.13 Truque de divisão	14
3.2 Árvore de Fenwick	5	5.14 Inclusão-Exclusão	15
3.3 Árvore de Fenwick 2D	5	5.15 Compressão de coordenadas	15
3.4 Árvore de Fenwick intervalar	5	5.16 Mínimo excluído com conjunto	15
3.5 Tabela esparsa	5	5.17 Mínimo excluído	15
3.6 Pilha mínima	5	5.18 Número (Números de Grundy)	15
3.7 Próximo maior elemento	5	5.19 Jogo de Nim	15
3.8 Fila mínima	6	5.20 Euclides estendido/inv. multiplicativo	15
3.9 Árvore de segmentos iterativa	6	5.21 Números de Catalão	15
3.10 Árvore de segmentos preguiçosa iter.	6	5.22 Detecção de ciclo	15
3.11 K maiores elementos	6	5.23 Equação diofantina linear	15
3.12 Hash customizado	7	5.24 Fatoração por tentativa	16
3.13 Conjunto de intervalos coloridos	7	5.25 Crivo de Eratóstenes	16
3.14 Árvore de segmentos preguiçosa rec.	7	5.26 Totiente de Euler	16
3.15 Árvore de segmentos recursiva	7	5.27 Eliminação gaussiana	16
3.16 Nó de Kadane	7	5.28 Exponenciação binária	16
3.17 Nó de parênteses	8	5.29 Miller-Rabin	16
		5.30 Pollard Rho	16
4 Grafos	8	5.31 Teorema chinês do resto generalizado	16
4.1 Articulações e pontes	8		
4.2 Componentes fortes – Tarjan	8	6 Strings	17
4.3 Componentes fortes – Kosaraju	8	6.1 KMP	17
4.4 Ordenação topológica	8	6.2 Algoritmo Z	17
4.5 Busca em largura 0-1	8	6.3 String Hashing	17
4.6 Caminho mínimo – Dijkstra	8	6.4 Autômato KMP	17
4.7 Caminho mínimo – Bellman-Ford	9	6.5 Autômato de Sufixo	17
4.8 Caminho mínimo – SPFA	9	6.6 Vetor de sufixos	18
4.9 Caminhos mínimos – Floyd-Warshall	9	6.7 Vetor de sufixos radix	18
4.10 Árvore geradora mínima – Kruskal	9	6.8 Aho-Corasick/Trie	18
		6.9 Árvore de sufixos	18

7	Geometria	19
7.1	Pontos	19
7.2	Fecho convexo de Graham	19
7.3	Fecho convexo com corrente monotônica	19
7.4	Distância máxima	19
7.5	Árvore KD	19
7.6	Ponto dentro do polígono?	20
7.7	Ponto dentro do polígono convexo?	20
7.8	Área de polígono	20
7.9	Pontos inteiros no polígono	20
7.10	Segmento	20
7.11	Ponto e segmento	20
7.12	Manhattan máximo	20
7.13	Voronoi	20
7.14	Varredura angular	22
7.15	Interseção de retângulos	22
7.16	Segmento e segmento	22
8	Algoritmos	22
8.1	Maior subsequência crescente rápido	22
8.2	Subconjuntos de tamanho K	22
8.3	Soma sobre subconjuntos	22
8.4	Ordenação por fusão	22
8.5	Algoritmo de Mo	22
8.6	Meet in the middle	22
8.7	Agendamento ótimo de tarefas	23
8.8	Maior subsequência comum	23
8.9	PD de perfil quebrado	23
8.10	Submáscaras	23
9	Problemas	23
9.1	Frequência de frequências/moda com Mo	23
9.2	Rainhas no tabuleiro	23
9.3	Remoção de valores com deque	23
9.4	Desembarcadouro	23
9.5	Inversões usando algoritmo de Mo	24
9.6	Média e mediana	24
9.7	Distância de edição	24
9.8	Código de Gray	24
9.9	Árvore de segmentos de GCD	24
9.10	Distância em árvore	24

1 Teoria

1.1 Limites numéricos

Sinal	Tipo	Bits	Máximo	Dígitos
+/-	char	8	127	2
+	char	8	255	2
+/-	short	16	32 767	4
+	short	16	65 535	4
+/-	int/long	32	$\approx 2 \cdot 10^9$	9
+	int/long	32	$\approx 4 \cdot 10^9$	9
+/-	long long	64	$\approx 9 \cdot 10^{18}$	18
+	long long	64	$\approx 18 \cdot 10^{18}$	19
+/-	__int128	128	$\approx 17 \cdot 10^{37}$	38
+	__int128	128	$\approx 3 \cdot 10^{38}$	38

1.2 Comparações numéricas

lg 10 (1E1)	2.3
lg 100 (1E1)	4.6
lg 1000 (1E2)	6.9
lg 10000 (1E3)	9.2
lg 100000 (1E4)	11.5
lg 1000000 (1E5)	13.8
lg 10000000 (1E6)	16.1
lg 100000000 (1E7)	18.4
lg 1000000000 (1E8)	20.7
lg 10000000000 (1E9)	23.0
lg 100000000000 (1E10)	25.3
lg 1000000000000 (1E11)	27.6
lg 10000000000000 (1E12)	29.9
2^{10}	$\approx 10^3$
2^{20}	$\approx 10^6$

1.3 Identidades binomiais

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}, \forall n \geq 0 \wedge 0 \leq k \leq n$$

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

1.4 Teorema de Lucas

$$\binom{n}{m} = \prod_{i=0}^k \binom{n_i}{m_i} \pmod{p}$$

Para p primo, n_i e m_i são coeficientes das representações de n e m na base p .

n	Pior algoritmo que passa	Comentário
$\leq [10..11]$	$\mathcal{O}(n!)$, $\mathcal{O}(n^6)$	ex. Enumerar permutações
$\leq [15..18]$	$\mathcal{O}(2^n n^2)$	ex. PD do Caixeiro-Viajante
$\leq [18..22]$	$\mathcal{O}(2^n n)$	ex. PD com técnica de máscara de bits
≤ 100	$\mathcal{O}(n^4)$	ex. PD com 3 dimensões + laço $\mathcal{O}(n)$, ${}_nC_{k=4}$
≤ 400	$\mathcal{O}(n^3)$	ex. Floyd-Warshall
$\leq 2 \cdot 10^3$	$\mathcal{O}(n^2 \lg n)$	ex. 2 laços aninhados + consulta em árvore
$\leq 5 \cdot 10^4$	$\mathcal{O}(n^2)$	ex. Bubble/Selection/Insertion Sort
$\leq 10^5$	$\mathcal{O}(n \lg^2 n) = \mathcal{O}((\lg n)(\lg n))$	ex. Construção de vetor de sufixos padrão
$\leq 10^6$	$\mathcal{O}(n \lg n)$	ex. Merge Sort, construir árvore de segmento
$\leq 10^7$	$\mathcal{O}(n \lg \lg n)$	ex. Crivo de Eratóstenes, função totiente
$\leq 10^8$	$\mathcal{O}(n)$, $\mathcal{O}(\lg n)$, $\mathcal{O}(1)$	ex. Solução matemática. Maioria dos problemas tem $n \leq 10^9$ (gargalo de E/S)

10⁸ operações por segundo.

1.5 Teorema do Chicken McNugget

Dados dois números coprimos n e m , o maior número que não pode ser escrito como uma combinação linear deles é $nm - n - m$.

- Existem $\frac{(n-1)(m-1)}{2}$ inteiros não-negativos que não podem ser escritos como uma combinação linear de n e m .
- Para cada par $(k, nm - n - m - k)$, para $k \geq 0$, exatamente um pode ser escrito.

1.6 Triplas pitagóricas

Para todo $a, b, c \in \mathbb{N}$ satisfazendo $a^2 + b^2 = c^2$, existem $m, n \in \mathbb{N}$ e $m > n$ de tal forma que:

$$a = m^2 - n^2 \quad b = 2mn \quad c = m^2 + n^2$$

1.7 Razões trigonométricas

	30°	45°	60°
$\sin \theta$	$\frac{1}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{3}}{2}$
$\cos \theta$	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{1}{2}$
$\tan \theta$	$\frac{\sqrt{3}}{3}$	1	$\sqrt{3}$

1.8 Soma de ângulos

$$\begin{aligned}\sin(a \pm b) &= \sin a \cos b \pm \cos a \sin b \\ \cos(a \pm b) &= \cos a \cos b \mp \sin a \sin b \\ \tan(a \pm b) &= \frac{\tan a \pm \tan b}{1 \mp \tan a \tan b}\end{aligned}$$

1.9 Grafos planares

- Se G tem k componentes conectadas, então $n - m + f = k + 1$.
- $m \leq 3n - 6$. Se G não tem triângulos, $m \leq 2n - 4$.
- O grau mínimo é ≤ 5 . E pode ser 6-colorido em $\mathcal{O}(n + m)$.

1.10 Corte mínimo

O algoritmo de Stoer-Wagner acha o corte mínimo em grafos não-direcionados. Entre dois vértices, o Teorema do Fluxo Máximo e Corte Mínimo afirma que o fluxo máximo é igual ao peso total das arestas do custo mínimo. Para achar o corte, faça uma busca em profundidade partindo da fonte, percorrendo apenas as arestas com capacidade residual. Todos os vértices alcançados fazem parte de uma das partes do corte mínimo.

1.11 Fatorial e desarranjos

$$\begin{aligned}n! &\approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \\ !n &= n! \sum_{i=0}^n \frac{(-1)^i}{i!} \text{ para } n \geq 0 \\ !n &= \left\lfloor \frac{n! + 1}{e} \right\rfloor \text{ para } n \geq 1 \\ !n &= n(!n - 1) + (-1)^n \text{ para } n > 0\end{aligned}$$

1.12 Teorema binomial

$$\begin{aligned}(x + y)^n &= \sum_{k=0}^n \binom{n}{k} x^{n-k} y^k = \sum_{k=0}^n \binom{n}{k} x^k y^{n-k} \\ (x + 1)^n &= \sum_{k=0}^n \binom{n}{k} x^k\end{aligned}$$

1.13 Círculo inscrito em triângulo

O centro do círculo forma três triângulos. A área do triângulo ABC é igual a $\frac{rp}{2}$ onde p é o perímetro do triângulo.

1.14 Lei dos senos

Dado um triângulo inscrito em um círculo de raio r , vale que

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2r$$

1.15 Lei dos cossenos

Triângulo ABC com lados a, b e c , a lei dos cossenos:

$$a^2 = b^2 + c^2 - 2bc \cdot \cos A$$

E podemos derivar a desigualdade triangular:

$$c^2 \leq a^2 + b^2 + 2ab = (a + b)^2$$

1.16 Truque da distância Manhattan

Seja $\mathcal{L}((x, y)) = (x + y, x - y)$. Temos as seguintes funções de distância:

- Manhattan: $M(p, q) = |p.x - q.x| + |p.y - q.y|$.
- Chebyshev: $C(p, q) = \max(|p.x - q.x|, |p.y - q.y|)$.

Então:

- $\mathcal{L}(\mathbb{Z}^2)$ escala \mathbb{Z}^2 em $\sqrt{2}$ e rotaciona em $\frac{\pi}{4}$ em sentido horário.
- Para algum $p \in \mathbb{Z}^2$, $\mathcal{L}(\{q : M(q, p) \leq d\})$ (círculo) forma um quadrado alinhado nos eixos em $\mathcal{L}(\mathbb{Z})^2$, com canto inferior-esquerdo em $\mathcal{L}(p) - (d, d)$ e canto superior-direito em $\mathcal{L}(p) + (d, d)$.
- $M(p, q) = C(\mathcal{L}(p), \mathcal{L}(q))$, e $C(p, q) = M(\mathcal{L}^{-1}(p), \mathcal{L}^{-1}(q))$.

1.17 Números de Catalão

1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452, 18367353072152, 69533550916004, 263747951750360, 1002242216651368.

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} = \prod_{k=2}^n \frac{n+k}{k}, n \geq 0.$$

Aplicações:

- C_n é o número de expressões contendo n parênteses corretamente pareados.
- C_n é o número de árvores binárias completas com $n+1$ folhas.
- C_n são as vezes que um polígono convexo com $n+2$ lados pode ser cortado em triângulos conectando os vértices e com linhas retas.

1.18 Fórmula de Heron

A área de um triângulo pode ser escrita como

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

onde $s = \frac{a+b+c}{2}$.

1.19 Progressões

$$a_n = a_k + r(n-k)$$
$$a_n = a_k q^{(n-k)}$$

- r, q : Razão
- k : Termo conhecido
- n : Termo que se quer obter

$$S_n = \frac{n(a_1 + a_n)}{2}$$
$$S_n = \frac{a_1(q^n - 1)}{q - 1}$$

1.20 Teoremas de Fermat

$$a^p = a \pmod{p}$$
$$a^{p-1} = 1 \pmod{p}$$
$$(a+b)^p = a^p + b^p \pmod{p}$$
$$a^{-1} = a^{p-2} \pmod{p}$$

onde p é primo.

1.21 Módulo no expoente

$$a^n = a^{n \pmod{\varphi(m)}} \pmod{m}$$

para a e m coprimos.

1.22 Séries

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$
$$\sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4} = \left(\sum_{i=1}^n i \right)^2$$

1.23 Alguns primos

- $10^6 + 69$
- $10^9 + 7$
- $10^9 + 9$
- $10^9 + 21$
- $10^{18} - 11$
- $10^{18} + 3$
- $2^{61} - 1$
- 1000696969
- 998244353 ((119<<23)+1)
- 880803841 ((105<<23)+1)
- 754974721 ((45<<24)+1)
- 469762049 ((7<<26)+1)
- 167772161 ((5<<25)+1)

2 C++

2.1 Esboço

```
2b7 #include <bits/stdc++.h>
58b using namespace std; using ll = long long;
5d2 int main() { cin.tie(0)->sync_with_stdio(0); }
```

2.2 Teste de estresse

```
8c6 for (( I=0; I < 5; I++ )); do
3bf ./gen $I >t.in; ./brute <t.in >t.exp; ./a.out <t.in >t.out
35d if diff -u t.exp t.out; then : ; else
a3e echo "--> entrada:"; cat t.in
ced echo "--> saída esperada"; cat t.exp
12e echo "--> saída obtida"; cat t.out; break; fi
132 echo -n .
6b2 done
```

2.3 Hash da entrada padrão

```
766 import subprocess as sp, hashlib as hl, sys
713 p = lambda a: ''.join(
324 sp.run(['cpp', '-d', '-P', '-fpreprocessed'],
5de input=a, text=True, capture_output=True).stdout.split())
e91 l, s = [], []
e5e for i, line in enumerate(sys.stdin):
11c l.append(p(line))
6c4 for c in l[i]:
dab if c == '{': s.append(i)
753 elif c == '}': lp = s[-1]; s.pop()
653 print(hl.md5((p(' '.join(l[lp:])).encode()
```

2.4 vimrc

```
9bd set ts=2 sw=2 cin ai si et sta udf
076 set is tm=50 nu noeb bg=dark ru cul sm
bcc sy on
```

2.5 Gerador

```
b32 mt19937 rnd; ll gen(ll a, ll b) {
000 uniform_int_distribution<ll> dist(a, b); return dist(rnd); }
f3b int main(int argc, char* argv[]) {
7d4 if (argc < 2) {
95e cerr << "usage: " << argv[0] << " <seed>\n"; return 1; }
a7f rnd = mt19937(atoi(argv[1]));
518 }
```

3 Estrutura de Dados

3.1 União-busca

```
bfb6 vector<int> rep (N), rnk (N), siz (N);

e51 int ds_find(int u) { // O(α(n))
71a   if (rep[u] ≠ u) { rep[u] = ds_find(rep[u]); }
5d2   return rep[u];
786 }

7b0 void ds_unite(int u, int v) { // O(α(n))
453   u = ds_find(u); v = ds_find(v);
2f3   assert(u ≠ v);
4c0   if (!(rnk[u] > rnk[v])) { swap(u, v); }
d24   if (rnk[u] == rnk[v]) { rnk[u]++; }
9e3   rep[v] = u;
2dd   siz[u] += siz[v];
7fa }
```

```
e12 void ds_init(int n) { // O(n)
19e   for (int u = 0; u < n; u++)
fbf     rep[u] = u, rnk[u] = 0, siz[u] = 1;
b6b }
```

3.2 Árvore de Fenwick

```
b36 vector<ll> bit (N, 0);

447 void add(int i, ll d) { // O(lg n)
40e   for (; i < N; i += i & -i) { bit[i] += d; }
62b }

462 ll pref(int i) { // O(lg n)
5ff   ll sum = 0;
344   for (; i > 0; i -= i & -i) { sum += bit[i]; }
e66   return sum;
7f9 }

3c0 int sum_inclusive(int i, int j) {
cff   return pref(j) - pref(i-1);
ca7 }

182 void add_inclusive(int a, int b, ll d) {
048   add(a, d); add(b + 1, -d);
fa9 }

727 int lower_bound(ll t) { // O(lg n)
8a6   ll sum = 0; int pos = 0;
eff   for (int p = log2(N); p ≥ 0; p--) if (
ee5     pos + (1<<p) < N && sum + bit[pos + (1<<p)] < t) {
8a0     sum += bit[pos + (1 << p)];
7ff     pos += (1 << p);
a17   }
d75   return pos;
796 }
```

3.3 Árvore de Fenwick 2D

```
1ad vector<vector<ll>> bit (N, vector<ll>(M));

c6f void add(int x, int y, ll d) { // O(lg2 n)
11f   for (; x < N; x += x & -x)
88e   for (int j = y; j < M; j += j & -j)
cee     bit[x][j] += d;
a62 }

cbc ll pref(int x, int y) { // O(lg2 n)
5ff   ll sum = 0;
c72   for (; x > 0; x -= x & -x)
f9f   for (int j = y; j > 0; j -= j & -j)
611     sum += bit[x][j];
e66   return sum;
ceb }

a20 ll sum_inclusive(int x1, int y1, int x2, int y2) {
ef7   return pref(x2, y2) - pref(x2, y1 - 1)
e37     - pref(x1 - 1, y2) + pref(x1 - 1, y1 - 1);
```

531 }

3.4 Árvore de Fenwick intervalar

```
739 vector<ll> bit1 (N), bit2 (N);

4fb void add(vector<ll>& bit, int i, ll d) { // O(lg n)
40e   for (; i < N; i += i & -i) { bit[i] += d; }
f8f }

f39 ll pref(vector<ll>& bit, int i) { // O(lg n)
5ff   ll sum = 0;
344   for (; i > 0; i -= i & -i) { sum += bit[i]; }
e66   return sum;
c47 }

462 ll pref(int i) {
4ed   return pref(bit1, i) * i - pref(bit2, i);
a57 }

906 ll sum_inclusive(int i, int j) {
cff   return pref(j) - pref(i - 1);
054 }

182 void add_inclusive(int a, int b, ll d) {
897   add(bit1, a, d);
e6b   add(bit2, a, d * (a - 1));
164   add(bit1, b + 1, -d);
89f   add(bit2, b + 1, -d * b);
59e }
```

```
d14 const int K = 25;
bf1 vector<vector<int>> st (N, vector<int>(K+1));

129 void build(vector<int>& src) { // O(n lg n)
3d8   for (int i = 0; i < src.size(); i++)
696     st[i][0] = src[i];
2ce   for (int j = 1; j ≤ K; j++)
285     for (int i = 0; i + (1 << j) ≤ N; i++)
e62       st[i][j] = min(st[i][j-1], st[i+(1 << (j-1))][j-1]);
ddb }

83f int min_inclusive(int l, int r) { // O(1)
037   int j = __lg(r-l+1);
21f   return min(st[l][j], st[r-(1 << j)+1][j]);
4e5 }
```

3.5 Tabela esparsa

3.6 Pilha mínima

```
46e stack<pair<int, int>> st;

975 void push_el(int x) { // O(1)
3af   int m = st.empty() ? x : min(x, st.top().second);
357   st.push({x, m});
66e }

883 int pop_el() { // O(1)
25f   int r = st.top().first; st.pop(); return r;
4a5 }

414 int min_el() { // O(1)
520   return st.top().second;
c7c }
```

3.7 Próximo maior elemento

```
de7 vector<int> next_greater_element(vector<int> v) { // O(n)
d46   vector<int> ans (v.size()); stack<int> nge;
// start from the end for previous
603   for (int i = 0; i < n; i++) {
// < is greater next, > is lesser next
ad5     while (!nge.empty() && v[nge.top()] < v[i])
7fe       ans[nge.top()] = i, nge.pop();
7db     nge.push(i);
957   }
ba7   return ans;
b74 }
```

3.8 Fila mínima

```
f23 stack<pair<int, int>> s1, s2;

556 void enqueue_el(int x) { // O(1)
973     int m = min(x, s1.size() ? s1.top().second : oo);
8d0     s1.push({ x, m });
51c }
f0a int dequeue_el() { // O(1)
43a     if (s2.empty()) while (!s1.empty()) {
af8         int el = s1.top().first; s1.pop();
577         int m = min(el, s2.size() ? s2.top().second : oo);
7dd         s2.push({ el, m });
8c9     }
f37     int r = s2.top().first; s2.pop(); return r;
4e1 }
414 int min_el() { // O(1)
03f     return min(
269         s1.size() ? s1.top().second : oo,
9a3         s2.size() ? s2.top().second : oo);
588 }
```

3.9 Árvore de segmentos iterativa

Macro	+	mín	máx
OP(X, Y)	((X) + (Y))	min(X, Y)	max(X, Y)
NEUTRAL	0	oo	-oo
FACTOR	(tb - ta + 1)	1	1

```
265 vector<int> t (2*N);

d9b void build(vector<int>& src, int n) { // O(n)
535     for (int i = 1; i ≤ n; i++)
920         t[n+i] = src[i];
79e     for (int ti = n-1; ti > 0; ti--)
0c6         t[ti] = OP(t[2*ti], t[2*ti+1]);
435 }

5f1 int op_inclusive(int l, int r, int n) { // O(lg n)
668     r++; int left = NEUTRAL, right = NEUTRAL;
4f7     for (l += n, r += n; l < r; l /= 2, r /= 2) {
7ba         if (l & 1) left = OP(left, t[l++]);
1d9         if (r & 1) right = OP(t[--r], right);
ae9     }
6b9     return OP(left, right);
a88 }

849 void set_value(int i, int v, int n) { // O(lg n)
4b8     t[i += n] = v;
3a9     for (i /= 2; i > 0; i /= 2)
5ee         t[i] = OP(t[i*2], t[i*2+1]);
41b }
```

3.10 Árvore de segmentos preguiçosa iter.

```
3d4 const int L = ceil(log2(N));
e03 struct dlta { int add = 0, set = -1; };
c16 vector<ll> t (2*N); vector<dlta> delta (2*N);

d9b void build(vector<int>& src, int n) { // O(n)
535     for (int i = 1; i ≤ n; i++)
920         t[n+i] = src[i];
79e     for (int ti = n-1; ti > 0; ti--)
0c6         t[ti] = OP(t[2*ti], t[2*ti+1]);
435 }

515 ll apply(int ti, dlta d, int sz) {
df1     if (d.set ≠ -1) {
f78         t[ti] = ll(d.set) * FACTOR(sz);
cec         delta[ti] = { 0, d.set };
e09     }
f96     if (d.add ≠ 0) {
e3d         t[ti] += ll(d.add) * FACTOR(sz);
297         delta[ti].add += d.add;
460     }
ac4     return t[ti];
```

```
5d4 }
```

```
6f4 void pull(int i) {
c86     for (int s = __builtin_ctz(i)+1; s < L; s++) {
d2b         int ti = i >> s;
0c6         t[ti] = OP(t[2*ti], t[2*ti+1]);
f56     }
367 }
```

```
51a void push(int i) {
808     int sz = 1 << (L-1);
950     for (int s = L; s > 0; s--, sz /= 2) {
d2b         int ti = i >> s;
61a         apply(2*ti, delta[ti], sz);
e28         apply(2*ti+1, delta[ti], sz);
2c7         delta[ti] = {};
e9c     }
74b }
```

```
61e void apply_inclusive(int l, int r, int n,
595     char op = '\0', ll x = 0) { // O(lg n)
de0     r++; dlta d;
34c     if (op == '+') { d.add = x; }
979     if (op == '=') { d.set = x; }
957     int tl = l += n, tr = r += n, sz = 1;
22f     push(tl); push(tr);
2d0     for (; l < r; l /= 2, r /= 2, sz *= 2) {
8fc         if (l & 1) { apply(l++, d, sz); }
ba0         if (r & 1) { apply(--r, d, sz); }
591     }
b5c     pull(tl); pull(tr);
230 }
```

```
dde void add_inclusive(int l, int r, int n, ll d) {
531     apply_inclusive(l, r, n, '+', d);
63b }
```

```
4ea ll op_inclusive(int l, int r, int n) { // O(lg n)
e5f     r++;
957     int tl = l += n, tr = r += n, sz = 1;
22f     push(tl); push(tr);
906     ll left = NEUTRAL, right = NEUTRAL;
2d0     for (; l < r; l /= 2, r /= 2, sz *= 2) {
a90         if (l & 1) left = OP(left, apply(l++, dlta(), sz));
b01         if (r & 1) right = OP(apply(--r, dlta(), sz), right);
0dd     }
6b9     return OP(left, right);
5a3 }
```

3.11 K maiores elementos

```
774 #include <ext/pb_ds/assoc_container.hpp>
30f #include <ext/pb_ds/tree_policy.hpp>
0d7 using namespace __gnu_pbds;
0ca template <class T, class U = less<T>>
f84 using ord_set = tree<T, null_type, U, rb_tree_tag,
3a1     tree_order_statistics_node_update>;
4e1 template <class C = less<>>
53a struct k_elements {
234     int k; ord_set<pair<ll, int>, C> s; ll sum;
7ac     k_elements(int k) : k(k), sum(0) {};
a1a     void insert(int key, ll value) { // O(lg n)
918         int l = s.order_of_key(make_pair(value, key));
0e6         s.insert(make_pair(value, key));
3c5         if (l ≥ k) { return; }
6c0         sum += value;
f70         if (s.size() > k)
06f             sum -= s.find_by_order(k)->first;
e0d     }
e26     void remove(int key, ll value) { // O(lg n)
a23         if (s.order_of_key(make_pair(value, key)) < k) {
fc1             sum -= value;
f70             if (s.size() > k)
4c8                 sum += s.find_by_order(k)->first;
562         }
991         s.erase(s.find(make_pair(value, key)));
cbe     }
3c9 };
```

3.12 Hash customizado

```
ef6 struct custom_hash {
dea  size_t operator()(uint64_t x) const { // O(1)
a70  static const uint64_t FIXED_RANDOM =
e71  chrono::steady_clock::now()
31f  .time_since_epoch().count();
c64  x += FIXED_RANDOM + 0x9e3779b97f4a7c15;
3e0  x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
31b  x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
d03  return x ^ (x >> 31);
17a }
188 };
```

3.13 Conjunto de intervalos coloridos

```
39c struct itvl {
70f  int a, b, c;
2a5  itvl(int _a, int _b, int _c)
bc7  : a(_a), b(_b), c(_c) {}
5a5  bool operator<(const itvl& o) const {
1dd  return ii(b, a) < ii(o.b, o.a); }
a01  bool operator<(const int& o) const { return b < o; }
b8c };
```

```
e87 set<itvl, less<>> si;
```

```
bc4 void ins_in(itvl in) { // O(lg n)
e9c  auto it = si.lower_bound(in);
003  int a = in.a, b = in.b, c = in.c;
270  if (si.size() > 0 && it != begin(si)
ddb  && prev(it)->b+1 == a && prev(it)->c == c) {
702  auto prv = prev(it); int prva = prv->a;
2db  si.erase(prv); a = prva;
122  }
ac0  if (it != end(si) && c == it->c && b+1 == it->a) {
255  int nxb = it->b; si.erase(it); b = nxb;
0d8  }
a42  si.insert(itvl(a, b, c));
10f };
```

```
26f void upd_color(int a, int b, int c) { // O(lg n)
80b  auto it = si.lower_bound(b);
e16  if (it != end(si)) {
8d5  itvl ori = *it; si.erase(it);
140  if (ori.a < a) { ins_in(itvl(ori.a, a-1, v)); }
b3b  if (ori.b > b) { ins_in(itvl(b+1, ori.b, v)); }
872  }
a21  ins_in(itvl(a, b, c));
7f4 };
```

3.14 Árvore de segmentos preguiçosa rec.

```
f7b vector<ll> t (4*N), lazy (4*N), sety (4*N, -1);
```

```
4bc void build(vector<int>& src,
51c  int ti=1, int tl=1, int tr=N) { // O(n)
cc2  if (tl == tr) {
d96  if (tl < src.size()) { t[ti] = src[tl]; }
505  return;
288  }
43e  int tm = (tl + tr) / 2;
80c  build(src, ti*2, tl, tm);
080  build(src, ti*2+1, tm+1, tr);
d72  t[ti] = OP(t[ti*2], t[ti*2+1]);
cc8 };
```

```
442 void push(int ti, int tl, int tm, int tr) {
5b6  if (sety[ti] != -1) {
9a5  t[ti*2] = sety[ti] * FACTOR(tm - tl + 1);
021  lazy[ti*2] = 0; sety[ti*2] = sety[ti];
02b  t[ti*2+1] = sety[ti] * FACTOR(tr - (tm+1) + 1);
060  lazy[ti*2+1] = 0; sety[ti*2+1] = sety[ti];
13e  sety[ti] = -1;
bb8  }
dda  t[ti*2] += lazy[ti] * FACTOR(tm - tl + 1);
ffa  lazy[ti*2] += lazy[ti];
cff  t[ti*2+1] += lazy[ti] * FACTOR(tr - (tm+1) + 1);
dad  lazy[ti*2+1] += lazy[ti];
```

```
ee0  lazy[ti] = 0;
83f };
```

```
a69 void update_inclusive(int l, int r, char op, int d,
51c  int ti=1, int tl=1, int tr=N) { // O(lg n)
472  if (l > r) { return; }
46c  if (l == tl && tr == r) {
e2c  if (op == '=') {
7ad  t[ti] = 0; sety[ti] = d; lazy[ti] = 0;
15f  } else if (op == '+') {
783  lazy[ti] += d;
ead  }
05e  t[ti] += ll(d) * FACTOR(tr - tl + 1);
505  return;
353  }
892  int tm = (tl + tr) / 2; push(ti, tl, tm, tr);
ea0  update_inclusive(l, min(r, tm), d, ti*2, tl, tm);
57a  update_inclusive(max(l, tm+1), r, d, ti*2+1, tm+1, tr);
d72  t[ti] = OP(t[ti*2], t[ti*2+1]);
3a3 };
```

```
83e ll op_inclusive(int l, int r,
51c  int ti=1, int tl=1, int tr=N) { // O(lg n)
6f2  if (l > r) { return NEUTRAL; }
534  if (l <= tl && tr <= r) { return t[ti]; }
892  int tm = (tl + tr) / 2; push(ti, tl, tm, tr);
1dc  return OP(op_inclusive(l, min(r, tm), ti*2, tl, tm),
a27  op_inclusive(max(l, tm+1), r, ti*2+1, tm+1, tr));
bab };
```

3.15 Árvore de segmentos recursiva

```
30b vector<ll> t (4*N);
```

```
e11 void build(vector<ll>& src,
51c  int ti=1, int tl=1, int tr=N) { // O(n)
cc2  if (tl == tr) {
d96  if (tl < src.size()) { t[ti] = src[tl]; }
505  return;
288  }
43e  int tm = (tl + tr) / 2;
80c  build(src, ti*2, tl, tm);
080  build(src, ti*2+1, tm+1, tr);
d72  t[ti] = OP(t[ti*2], t[ti*2+1]);
cc8 };
```

```
83e ll op_inclusive(int l, int r,
51c  int ti=1, int tl=1, int tr=N) { // O(lg n)
6f2  if (l > r) { return NEUTRAL; }
d85  if (l == tl && r == tr) { return t[ti]; }
43e  int tm = (tl + tr) / 2;
1dc  return OP(op_inclusive(l, min(r, tm), ti*2, tl, tm),
a27  op_inclusive(max(l, tm+1), r, ti*2+1, tm+1, tr));
143 };
```

```
00c void set_value(int i, ll v,
51c  int ti=1, int tl=1, int tr=N) { // O(lg n)
430  if (tl == tr) { t[ti] = v; return; }
43e  int tm = (tl + tr) / 2;
49f  if (i <= tm) { set_value(i, v, ti*2, tl, tm); }
685  else { set_value(i, v, ti*2+1, tm+1, tr); }
d72  t[ti] = OP(t[ti*2], t[ti*2+1]);
d57 };
```

3.16 Nó de Kadane

```
a22 struct kadane {
721  int sum, pre, suf, ans;
739  kadane(int x) : sum(x) { pre = suf = ans = max(x, 0); }
3d1  kadane() : kadane(-oo) {}
fd6  kadane operator+(kadane const& r) {
f03  auto l = *this; kadane a;
bef  a.sum = max(l.sum + r.sum, -oo);
0d5  a.pre = max(l.pre, l.sum + r.pre);
dd6  a.suf = max(r.suf, r.sum + l.suf);
99b  a.ans = max({ l.ans, r.ans, l.suf + r.pre });
3f5  return a;
306  }
de2 };
```


3.17 Nó de parênteses

```
3c9 struct node {
27a   int open; int closed; int ans;
a03   explicit node(char c) : open(0), closed(0), ans(0) {
01e     if (c == '(') { open = 1; }
5b9     if (c == ')') { closed = 1; }
891   }
d89   node operator+(node r) {
750     node a (0);
c8f     int mn = min(open, r.closed);
887     a.ans = ans + r.ans + mn;
98d     a.open = open - mn + r.open;
0f9     a.closed = r.closed - mn + closed;
3f5     return a;
cfa   }
e57 };
```

4 Grafos

4.1 Articulações e pontes

```
845 int tk = 1; vector<int> tin (N, -1), low (N);
07c vector<ii> brid; set<int> arti;

fb6 void dfs(int u, int p) { // O(n+m)
921   tin[u] = low[u] = tk++; int ch = 0;
7b9   for (auto v : g[u]) {
730     if (v == p) continue;
4c8     else if (tin[v] == -1) {
4a2       dfs(v, u); ch++;
1c1       if ((low[v] ≥ tin[u] && p ≠ u) ||
1ee         (ch ≥ 2 && p == u))
7fc         arti.insert(u);
2ce       if (low[v] > tin[u])
ea6         brid.push_back(ii(u, v));
ab6       low[u] = min(low[u], low[v]);
a99     } else { low[u] = min(low[u], tin[v]); }
8ce   }
d41 }
```

4.2 Componentes fortes – Tarjan

```
c7d vector<int> tin (N, -1), lowlnk (N, -1), rep (N);
2c6 stack<int> st, topo; int cts = 1;

79b void dfs_tarjan(int u) {
b9a   if (tin[u] ≠ -1) { return; }
694   lowlnk[u] = tin[u] = vis[u] = cts++;
4a6   st.push(u);
4d5   for (int v : g[u]) {
f2f     dfs_tarjan(v);
2ce     if (vis[v]) lowlnk[u] = min(lowlnk[u], lowlnk[v]);
392   }
f58   if (lowlnk[u] == tin[u]) {
299     int v; do {
cc8       v = st.top(); st.pop(); vis[v] = 0;
9e3       rep[v] = u;
578     } while (u ≠ v);
cc5     topo.push(u);
7bb   }
5c8 }

99b stack<int> tarjan(int n) { // O(n+m)
a9f   for (int u = 0; u < n; u++) { dfs_tarjan(u); }
7b6   return topo;
164 }
```

4.3 Componentes fortes – Kosaraju

```
cfc int cts = 1; vector<int> vis (N), rep (N);
81e stack<int> sinks;

11d void fill_stack(int u) {
454   if (vis[u] == cts) { return; } vis[u] = cts;
789   for (int v : g_t[u]) { fill_stack(v); }
f9d   sinks.push(u);
```

```
cf8 }
```

```
744 void mark_component(int u, int r) {
78f   if (vis[u] == cts) { return; }
3a5   vis[u] = cts; rep[u] = r;
b00   for (int v : g[u]) { mark_component(v, r); }
3c1 }

ba2 stack<int> kosaraju(int n) { // O(n+m)
989   for (int u = 0; u < n; u++) { fill_stack(u); }
d52   cts++; stack<int> topo;
aca   while (!sinks.empty()) {
dd1     int u = sinks.top(); sinks.pop();
9e6     mark_component(u, u);
eba     if (rep[u] == u) topo.push(u);
20f   }
7b6   return topo;
689 }
```

4.4 Ordenação topológica

```
291 vector<bool> vis (N); stack<int> st;

f28 void dfs_topo(int u) {
7bd   if (vis[u]) { return; }
150   vis[u] = 1;
7f9   for (int v : g[u]) { dfs_topo(v); }
4a6   st.push(u);
bc5 }

3ff vector<int> toposort(int n) { // O(n+m)
cf8   fill(begin(vis), begin(vis)+n, 0);
e32   vector<int> topo;
654   for (int u = 0; u < n; u++) { dfs_topo(u); }
6f2   while (!st.empty()) {
672     int v = st.top(); st.pop();
529     topo.push_back(v);
2ec   }
7b6   return topo;
1ac }
```

4.5 Busca em largura 0-1

```
52a vector<int> d (N, oo);

6c8 void bfs01(int s) { // O(n+m)
799   d[s] = 0; deque<int> q; q.push_front(s);
14d   while (!q.empty()) {
97a     int u = q.front(); q.pop_front();
f4a     for (auto [v, w] : g[u]) if (d[u] + w < d[v]) {
aa9       d[v] = d[u] + w;
721       if (w == 1) { q.push_back(v); }
f0b       else { q.push_front(v); }
931     }
8d0   }
7f1 }
```

4.6 Caminho mínimo – Dijkstra

```
144 void dijkstra(int s, int t, int n) { // O((m+n)lg n)
f92   vector<ll> d (n, oo); vector<bool> vis (n);
116   vector<vector<int>> dag (n);
77a   priority_queue<ii, vector<ii>, greater<ii>> q;
4fd   d[s] = 0; q.push({0, s});
14d   while (!q.empty()) {
a75     auto [c, u] = q.top(); q.pop();
0d1     if (vis[u]) { continue; } vis[u] = 1;
bc2     for (auto [v, w] : gt[u]) if (d[u] == d[v] + w)
d39       dag[v].push_back(u);
bd1     if (u == t) { break; }
505     for (auto [v, w] : g[u])
e17       if (d[v] > d[u] + w) {
aa9         d[v] = d[u] + w;
33e         q.push({d[v], v});
c80       }
32c   }
ffa }
```

4.7 Caminho mínimo – Bellman-Ford

```
abc struct edge { int u, v, w; };
ec9 vector<edge> edges;
52a vector<int> d (N, oo);

99d int bellman_ford(int src, int dest, int n) { //  $\mathcal{O}(nm)$ 
455     d[src] = 0;
891     for (int i = 0; i < n - 1; i++)
b33         for (auto e : edges)
301             if (d[e.u] != oo && d[e.v] > d[e.u] + e.w)
bed                 d[e.v] = d[e.u] + e.w;
// Verificação de ciclos negativos
b33     for (auto e : edges)
301         if (d[e.u] != oo && d[e.v] > d[e.u] + e.w)
ad5             return -oo;
367     return d[dest];
9d8 }
```

4.8 Caminho mínimo – SPFA

```
848 bool spfa(int src, int n) { //  $\mathcal{O}(nm)$ 
750     vector<int> cnt (n); vector<bool> queu (n);
41e     vector<ll> d (n, oo);
d77     queue<int> q; q.push(src); d[src] = 0;
14d     while (!q.empty()) {
be1         int u = q.front(); q.pop();
6c7         queu[u] = 0;
b59         for (auto [v, w] : g[u]) if (d[v] > d[u] + w) {
aa9             d[v] = d[u] + w;
626             if (!queu[v]) {
65d                 q.push(v); queu[v] = 1; cnt[v]++;
// negative cycle
850                 if (cnt[v] > n) { return 0; }
da6             }
51e         }
00c     }
6a5     return 1;
572 }
```

4.9 Caminhos mínimos – Floyd-Warshall

```
f9b vector<vector<ll>> d (N, vector<ll>(N, oo));

4c8 void floydwarshall(int n) { //  $\mathcal{O}(n^3)$ 
19e     for (int u = 0; u < n; u++)
b61         d[u][u] = 0;
007     for (int m = 0; m < n; m++)
19e         for (int u = 0; u < n; u++)
84d             for (int v = 0; v < n; v++)
337                 d[u][v] = min(d[u][v], d[u][m] + d[m][v]);
f28 }
```

4.10 Árvore geradora mínima – Kruskal

```
670 struct edge {
28f     int u, v, w;
f5a     bool operator<(struct edge &o) { return w < o.w; }
600 };

d1e int kruskal(int n) { //  $\mathcal{O}(m\alpha(n))$ 
5c1     sort(all(edges));
0bc     ds_init(n); int components = n; ll sum = 0;
f59     for (auto [u, v, w] : edges) {
d89         if (components == 1) { break; }
58f         if (ds_find(u) != ds_find(v)) {
ee7             ds_union(u, v);
59e             components--;
70b             sum += w;
6c9         }
112     }
e66     return sum;
6e5 }
```

4.11 Árvore geradora mínima – Prim

```
bac ll prim(int src, int n) { //  $\mathcal{O}((m+n)\lg n)$ 
7e6     vector<bool> vis (n); vector<int> par (n, -1);
```

```
324     vector<ll> d (n, oo); vector<vector<ii>> pt (n);
5ff     ll sum = 0;
77a     priority_queue<ii, vector<ii>, greater<ii>> q;
696     q.push(ii(d[src] = 0, src));
14d     while (!q.empty()) {
a75         auto [c, u] = q.top(); q.pop();
b65         if (vis[u]) { continue; }
150         vis[u] = 1;
d00         if (par[u] != -1) {
c60             int v = par[u]; ll w = d[u];
9c5             pt[u].push_back(ii(v, w));
e49             pt[v].push_back(ii(u, w));
d06         }
7ee         sum += c;
505         for (auto [v, w] : g[u])
d83             if (!vis[v] && w < d[v]) {
76d                 q.push(ii(d[v] = w, v));
1a4                 par[v] = u;
ea9             }
1f4         }
e66     return sum;
83a }
```

4.12 Caminho euleriano

```
f5c vector<int> eulerian_path(int src, int n) { //  $\mathcal{O}(n+m)$ 
19e     for (int u = 0; u < n; u++)
69b         if (deg[u] % 2) { return {}; }
235     stack<int> st; st.push(src);
dab     vector<int> ans;
6f2     while (!st.empty()) {
f1a         int u = st.top();
0de         if (g[u].empty()) {
361             ans.push_back(u); st.pop();
9d9         } else {
4a0             int v = *begin(g[u]);
8fc             g[u].erase(g[u].find(v));
a64             g[v].erase(g[v].find(u));
789             st.push(v);
7bd         }
1af     }
2d8     for (int u = 0; u < n; u++) if (g[u].size())
21d         return {};
1dc     reverse(all(ans));
ba7     return ans;
415 }
```

4.13 Menor no maior

```
a94 vector<int> hvy (N), sz (N, 1);

4ca int calc_size(int u, int p) {
535     for (int v : g[u]) if (v != p)
bae         sz[u] += calc_size(v, u);
f93     return sz[u];
199 }

db7 void dfs_add(int u, int p, int x) {
// implement add
337     for (int v : g[u]) if (v != p && !hvy[v])
21e         dfs_add(v, u, x);
1a3 }

aeb void small_large(int u, int p, bool keep) { //  $\mathcal{O}(n\lg n)$ 
cd3     int mx = -1, h = -1;
8c6     for (int v : g[u]) if (v != p && sz[v] > mx) {
b57         mx = sz[v]; h = v;
d66     }
08a     for (int v : g[u]) if (v != p && v != h)
627         small_large(v, u, 0);
34e     if (h != -1) { small_large(h, u, 1); hvy[h] = 1; }
722     dfs_add(u, p, +1);
// solve queries for u
e16     if (h != -1) { hvy[h] = 0; }
ba7     if (keep == 0) { dfs_add(u, p, -1); }
a20 }
```

4.14 Decomposição pesado-leve

```
8f5 vector<int> hvy (N), par (N), dep (N), hds (N), ixs (N);
d6c vector<ll> wei (N), ori (N);
e3f int cix = 1;

53f int hld_fill(int u, int p, int d = 0, ll w = 0) {
9ff int s = 1, maxs = 0;
978 wei[u] = w; par[u] = p; dep[u] = d; hvy[u] = -1;
4c3 for (auto [v, w] : g[u]) if (v != p) {
ab9 int cs = hld_fill(v, u, d+1, w);
f6c s += cs;
832 if (cs > maxs) { maxs = cs; hvy[u] = v; }
0c4 }
047 return s;
4ac }

f2b void hld(int u, int h) {
54c hds[u] = h; ori[cix] = wei[u]; ixs[u] = cix++;
fa4 if (hvy[u] != -1)
127 hld(hvy[u], h); // continue chain
ff8 for (auto [v, w] : g[u]) if (v != par[u] && v != hvy[u])
973 hld(v, v); // new chain
feb }

d4a ll hld_op(int u, int v, bool edge_wei = 1) { // O(lg q)
04b ll ans = 0;
a90 for (; hds[u] != hds[v]; v = par[hds[v]]) {
fa0 if (dep[hds[u]] > dep[hds[v]]) { swap(u, v); }
1d7 ans = OP(ans, op_inclusive(ixs[hds[v]], ixs[v]));
b62 }
443 if (dep[u] > dep[v]) { swap(u, v); }
13c return OP(ans, op_inclusive(ixs[u] + edge_wei, ixs[v]));
c33 }

a81 void hld_init(int u, int n) { // O(n+m+b)
595 cix = 1; hld_fill(u, u); hld(u, u); build(ori, n);
98d }
```

4.15 Decomposição pesado-leve – K-ésimo

```
dc7 int hld_len(int u, int v) {
1a4 int ans = 0;
a90 for (; hds[u] != hds[v]; v = par[hds[v]]) {
fa0 if (dep[hds[u]] > dep[hds[v]]) { swap(u, v); }
c89 ans += ixs[v] - ixs[hds[v]] + 1;
75d }
443 if (dep[u] > dep[v]) { swap(u, v); }
d50 return ans + ixs[v] - ixs[u] + 1;
88d }

82f int hld_kth(int u, int v, int k) {
1de bool sw = 0; int l = 0, r = hld_len(u, v)-1;
a90 for (; hds[u] != hds[v]; v = par[hds[v]]) {
311 if (dep[hds[u]] > dep[hds[v]])
573 { swap(u, v); sw ^= 1; }
5ce int sz = ixs[v]-ixs[hds[v]]+1;
c86 int i = sw ? k-l : r-k;
ae9 if (0 ≤ i && i < sz) { return rixs[ixs[v]-i]; }
fe6 if (sw) { l += sz; } else { r -= sz; }
8e4 }
119 if (dep[u] > dep[v]) { swap(u, v); sw ^= 1; }
c86 int i = sw ? k-l : r-k;
6f8 return rixs[ixs[v]-i];
5c0 }

1aa vector<int> hld_path(int u, int v, int k) {
b66 int len = hld_len(u, v); vector<int> path (len);
92b bool sw = 0; int l = 0, r = len-1;
a90 for (; hds[u] != hds[v]; v = par[hds[v]]) {
311 if (dep[hds[u]] > dep[hds[v]])
573 { swap(u, v); sw ^= 1; }
522 for (int i = ixs[v]; i ≥ ixs[hds[v]]; i--)
8d7 path[sw ? l++ : r--] = rixs[i];
662 }
119 if (dep[u] > dep[v]) { swap(u, v); sw ^= 1; }
583 for (int i = ixs[v]; i ≥ ixs[u]; i--) {
8d7 path[sw ? l++ : r--] = rixs[i];
fffb }
```

```
535 return path;
30a }
```

4.16 Fluxo máximo

```
2d4 vector<vector<int>> res (N);
475 vector<int> ix (N), dist (N), par (N);
4de struct edge { int u, v, cap; };
ec9 vector<edge> edges;

569 void link(int u, int v, int c) {
499 res[u].pb(edges.size()); edges.pb({ u, v, c });
1a2 res[v].pb(edges.size()); edges.pb({ v, u, 0 });
340 }

fc8 bool minimum_path(int s, int t) {
a9e fill(all(dist), oo);
90c queue<int> q; q.push(s); dist[s] = 0;
14d while (!q.empty()) {
be1 int u = q.front(); q.pop();
bd1 if (u == t) { break; }
158 for (int i : res[u]) {
fdc edge e = edges[i]; int v = e.v;
a3e if (e.cap && dist[v] == oo) {
8a0 dist[v] = dist[u] + 1;
331 par[v] = u; ix[v] = i;
2a1 q.push(v);
434 }
6cd }
14a }
5ab return dist[t] < oo;
8d3 }

364 pair<int, int> ffek(int s, int t) { // O(nm2)
186 int min_cost = 0, max_flow = 0;
792 while (minimum_path(s, t)) {
9c9 int flow = oo;
5d6 for (int u = t; u != s; u = par[u])
19d flow = min(flow, edges[ix[u]].cap);
043 for (int u = t; u != s; u = par[u]) {
bd0 edges[ix[u]].cap -= flow;
fa6 edges[ix[u]^1].cap += flow;
523 }
e26 min_cost += flow * dist[t];
3d7 max_flow += flow;
970 }
993 return { min_cost, max_flow };
950 }
```

4.17 Fluxo de custo mínimo

```
58b struct edge { int u, v, cap, cost; };
ec9 vector<edge> edges;
6b8 vector<int> queu (N), ix (N), dist (N), par (N);

782 void link(int u, int v, int cap, int cost) {
b0f res[u].push_back(edges.size());
7fd edges.push_back({ u, v, cap, cost });
8d9 res[v].push_back(edges.size());
e8a edges.push_back({ v, u, 0, -cost });
701 }

fc8 bool minimum_path(int s, int t) {
a9e fill(all(dist), oo);
90c queue<int> q; q.push(s); dist[s] = 0;
14d while (!q.empty()) {
be1 int u = q.front(); q.pop();
6c7 queu[u] = 0;
158 for (int i : res[u]) {
fdc edge e = edges[i]; int v = e.v;
033 if (e.cap && dist[v] > dist[u] + e.cost) {
0af dist[v] = dist[u] + e.cost;
331 par[v] = u; ix[v] = i;
385 if (!queu[v]) { q.push(v); queu[v] = 1; }
ca7 }
56a }
750 }
5ab return dist[t] < oo;
4cb }
```

4.18 Fluxo máximo Dinic

```
2d4 vector<vector<int>> res (N);
626 vector<int> level (N), ptr (N);
5c7 struct edge { int u, v; ll cap, flow = 0; };
e27 vector<edge> edges; queue<int> q;

fc8 bool minimum_path(int s, int t) {
14d while (!q.empty()) {
be1 int u = q.front(); q.pop();
158 for (int i : res[u]) {
468 if (edges[i].cap - edges[i].flow < 1) { continue; }
535 if (level[edges[i].v] != -1) { continue; }
b52 level[edges[i].v] = level[u] + 1;
2eb q.push(edges[i].v);
794 }
06a }
0f4 return level[t] != -1;
06e }

570 ll dfs(int u, int t, ll pushed) {
5db if (pushed == 0) { return 0; }
d2b if (u == t) { return pushed; }
246 for (int& cid = ptr[u]; cid < res[u].size(); cid++) {
a64 int i = res[u][cid], v = edges[i].v;
6aa if (level[u] + 1 != level[v]) { continue; }
468 if (edges[i].cap - edges[i].flow < 1) { continue; }
027 ll tr = dfs(v, t,
0b9 min(pushed, edges[i].cap - edges[i].flow));
cad if (tr == 0) { continue; }
125 edges[i].flow += tr; edges[i^1].flow -= tr;
a30 return tr;
1de }
bb3 return 0;
274 }

a68 ll dinic(int s, int t) { //  $O(mn^2)$ 
207 ll max_flow = 0;
31e while (1) {
e8e fill(all(level), -1); level[s] = 0; q.push(s);
86d if (!minimum_path(s, t)) { break; }
94e fill(all(ptr), 0);
3fb while (ll pushed = dfs(s, t, 1e18))
f9a max_flow += pushed;
d2c }
483 return max_flow;
2e4 }
```

4.19 Stoer-Wagner

```
17a vector<vector<int>> res (N, vector<int>(N));
70d vector<bool> bin (N);

21d int contract(int n, int &s, int &t) {
a70 vector<int> dist (n); vector<bool> vis (n);
3a0 int mincut, maxc;
603 for (int i = 0; i < n; i++) {
681 int k = -1; maxc = -1;
f90 for (int j = 0; j < n; j++)
58e if (!bin[j] && !vis[j] && dist[j] > maxc) {
1ba k = j;
d92 maxc = dist[j];
1ca }
c75 if (k == -1) { return mincut; }
466 s = t; t = k;
63c mincut = maxc; vis[k] = 1;
aeb for (int j = 1; j < n; j++)
296 if (!bin[j] && !vis[j]) {
64f dist[j] += res[k][j];
b20 }
697 }
302 return mincut;
eb0 }

2ae int stoer_wagner(int n) { //  $O(nm + n^2 \lg n)$ 
f05 int mincut = oo, s, t;
f50 for (int i = 0; i < n-1; i++) {
648 mincut = min(mincut, contract(n, s, t));
fa8 if (mincut == 0) { return 0; }
```

```
727 bin[t] = 1;
13a for (int j = 0; j < n; j++) if (!bin[j]) {
fe0 res[s][j] = (res[j][s] += res[j][t]);
54a }
695 }
302 return mincut;
385 }
```

4.20 Ascensão binária

```
894 const int L = log2(N);
be9 vector<int> dep (N, 0); vector<ll> wsum (N);
76c vector<vector<ll>> weiop (N, vector<ll>(L+1));
fa4 vector<vector<int>> up (N, vector<int>(L+1));

8b8 void bl_euler_tour(int u, int p, int w) {
eaf up[u][0] = p; weiop[u][0] = w;
664 dep[u] = dep[p] + 1; wsum[u] = wsum[p] + w;
431 for (auto [v, w] : g[u]) if (v != p)
ff2 bl_euler_tour(v, u, w);
d95 }

e96 void bl_init(int u, int n) { //  $O(m + n \lg n)$ 
7e3 dep[u] = -1; bl_euler_tour(u, u, 0);
// can be put into bl_euler_tour but it will not
// work with cycles (could be useful for online)
c00 for (int l = 0; l < L; l++)
687 for (int u = 0; u < n; u++) {
6ce int a = up[u][l];
be5 up[u][l+1] = up[a][l];
6dc weiop[u][l+1] = OP(weiop[u][l], weiop[a][l]);
78e }
98a }

bd8 ll bl_op(int a, int b) { //  $O(\lg n)$ 
cd7 if (!dep[a] > dep[b]) { swap(a, b); }
b77 ll res = NEUTRAL;
fa0 int diff = dep[a] - dep[b];
bc7 for (int l = L; l >= 0; l--) if (diff & (1 << l)) {
cee res = OP(res, weiop[a][l]); a = up[a][l];
908 }
518 if (a == b) { return res; }
bfc for (int l = L; l >= 0; l--)
424 if (up[a][l] != up[b][l]) {
9d7 res = OP(res, OP(weiop[a][l], weiop[b][l]));
162 a = up[a][l], b = up[b][l];
dbf }
a1b return OP(res, OP(weiop[a][0], weiop[b][0]));
30e }
```

```
1fe int bl_query_sum(int a, int b) {
e5f int lca = bl_lca(a, b);
2f5 return (wsum[a] - wsum[lca]) + (wsum[b] - wsum[lca]);
6c6 }
```

4.21 Menor anc. comum – Ancensão binária

```
a41 int bl_lca(int a, int b) {
980 if (!dep[b] < dep[a]) { swap(a, b); }
fa0 int diff = dep[a] - dep[b];
425 for (int l = L; l >= 0; l--) if (diff & (1 << l))
c42 a = up[a][l];
06e if (a == b) { return a; }
d31 for (int l = L; l >= 0; l--) if (up[a][l] != up[b][l])
162 a = up[a][l], b = up[b][l];
e6f return up[a][0];
31d }

c1f int bl_get_one_down(int a, int b) {
980 if (!dep[b] < dep[a]) { swap(a, b); }
7e3 int diff = max(dep[a] - dep[b] - 1, 0);
697 for (int l = L; l >= 0; l--) if (diff & (1 << l))
c42 a = up[a][l];
3f5 return a;
de1 }
```

4.22 Empar. máximo bipartido – Kuhn

```
// distinct enumeration
532 vector<vector<int>> g (L);
c87 vector<int> matchr (R, -1);

f2d int dfs(int u) {
a8f   if (vis[u] == cts) { return 0; } vis[u] = cts;
4fb   for (int v : g[u])
57c     if (matchr[v] == -1 || dfs(matchr[v])) {
f97       matchr[v] = u; return 1;
2e3     }
bb3   return 0;
73f }

79d int kuhn(int l) { // O(nm)
1a4   int ans = 0;
405   for (int u = 0; u < l; u++) { ans += dfs(u); cts++; }
ba7   return ans;
4a3 }
```

4.23 Empar. generalizado – Blossom

```
468 vector<int> match (N), par (N), base (N), vis (N);
3b9 int n; queue<int> q;

107 void contract(int u, int v, bool first = 1) {
e97   static vector<bool> bloss; static int l;
418   if (first) {
a47     bloss = vector<bool>(n, 0);
f09     vector<bool> ok (n, 0); int k = u; l = v;
31e     while (1) {
9b7       ok[k = base[k]] = 1;
de3       if (match[k] == -1) { break; }
c93       k = par[match[k]];
739     }
64e     while (!ok[l = base[l]]) l = par[match[l]];
beb   }
2e9   while (base[u] != l) {
e29     bloss[base[u]] = bloss[base[match[u]]] = 1;
5b4     par[u] = v; v = match[u]; u = par[match[u]];
087   }
5ee   if (!first) { return; }
95e   contract(v, u, 0);
6f9   for (int u = 0; u < n; u++) if (bloss[base[u]]) {
48c     base[u] = l;
764     if (!vis[u]) { q.push(u); } vis[u] = 1;
2f4   }
5f3 }

f10 int getpath(int s) {
830   for (int i = 0; i < n; i++)
e38     base[i] = i, par[i] = -1, vis[i] = 0;
ded   vis[s] = 1; q = queue<int>(); q.push(s);
402   while (q.size()) {
be1     int u = q.front(); q.pop();
bdc     for (int i : g[u]) {
7ce       if (base[i] == base[u] || match[u] == i)
5e2         continue;
08e       if (i == s || (~match[i] && ~par[match[i]]))
4f2         contract(u, i);
c54       else if (par[i] == -1) {
741         par[i] = u;
e8e         if (match[i] == -1) { return i; }
818         i = match[i];
29d         vis[i] = 1; q.push(i);
016       }
885     }
98d   }
daa   return -1;
b2b }
```

```
83f int blossom() { // O(n^3)
e80   int ans = 0; fill(all(match), -1);
2e3   for (int i = 0; i < n; i++) if (match[i] == -1)
f76     for (int j : g[i]) if (match[j] == -1) {
3bc       match[i] = j; match[j] = i; ans++; break; }
da8   for (int i = 0; i < n; i++) if (match[i] == -1) {
476     int j = getpath(i); if (j == -1) { continue; }
```

```
0df     ans++;
3a0     while (j != -1) {
baf       int p = par[j], pp = match[p];
978       match[p] = j; match[j] = p; j = pp;
437     }
287   }
ba7   return ans;
240 }
```

4.24 Emparelhamento estável

```
9e1 vector<vi> pm (N, vi(N)); // [m][j] = w
6b2 vector<vi> pwix (N, vi(N+1)); // [w][m] = j
bec vector<bool> single (N, 1);
dd8 vector<int> match (N, -1), watch (N, N);

4f8 vector<ii> galeshapley(int n) { // O(n^2)
3be   for (int w = 0; w < n; w++) { pwix[w][n] = n; }
8fb   bool done = 0;
947   while (!done) { done = 1;
238     for (int m = 0; m < n; m++) if (single[m]) {
df9       done = 0; match[m] += 1;
031       int w = pm[m][match[m]];
d95       if (pwix[w][m] < pwix[w][watch[w]]) {
38f         single[watch[w]] = 1;
3c6         watch[w] = m; single[m] = 0;
8ea       }
3b5     }
e7d   }
96e   vector<ii> ans;
007   for (int m = 0; m < n; m++)
6f7     ans.push_back(ii(m, pm[m][match[m]]));
ba7   return ans;
79c }
```

4.25 Decomposição centróide

```
fe2 vector<int> rep (N), parc (N), vis (N), sz (N);

4ca int calc_size(int u, int p) {
267   sz[u] = 1;
464   for (int v : g[u]) if (v != p && !vis[v])
bae     sz[u] += calc_size(v, u);
f93   return sz[u];
745 }

208 int find_centroid(int u, int p, int n) {
4fb   for (int v : g[u])
0f0     if (v != p && !vis[v] && sz[v] > n/2)
836       return find_centroid(v, u, n);
03f   return u;
b2f }
```

```
691 void centroid_decomp(int u, int p, int d=0) { // O(n+m)
716   calc_size(u, u);
b3f   int c = find_centroid(u, u, sz[u]);
dbc   vis[c] = 1; parc[c] = p;
bf4   for (int v : g[c]) if (!vis[v])
3af     centroid_decomp(v, c, d+1);
6ea }
```

4.26 Menor anc. comum – Offline

```
032 vector<vector<int>> queries (N);
852 vector<int> anc (N); vector<bool> vis (N);

315 void dfs(int u) { // O(na(n) + m + qa(n))
f9e   vis[u] = 1; anc[u] = u;
8db   for (int v : adj[u]) if (!vis[v]) {
951     dfs(v); ds_unite(u, v); anc[ds_find(u)] = u;
0dd   }
d7f   for (int v : queries[u]) if (vis[v])
7ee     lca[u][v] = anc[ds_find(v)];
389 }
```

4.27 Decomposição raiz quadrada de árvore

```
964 vector<int> up (N), bup (N), depth (N);
7e2 vector<ll> weiop (N), bweiop (N);

271 void stdt_decompose(int u, int p, int w) { // O(n+m)
2f7   up[u] = p; weiop[u] = w;
69a   depth[u] = depth[p] + 1;
f25   bup[u] = depth[u] % B ? bup[p] : p;
554   bweiop[u] = OP(depth[u] % B ? bweiop[p] : NEUTRAL, w);
431   for (auto [v, w] : g[u]) if (v != p)
80f     stdt_decompose(v, u, w);
05e }

4d5 int stdt_op(int a, int b) { // O(√n)
b35   int ans = NEUTRAL;
eb4   if (!(depth[a]/B > depth[b]/B)) { swap(a, b); }
5ac   while (depth[a]/B > depth[b]/B) {
585     ans = OP(ans, bweiop[a]); a = bup[a]; }
54a   if (!(depth[a] > depth[b])) { swap(a, b); }
65e   while (depth[a] > depth[b]) {
203     ans = OP(ans, weiop[a]); a = up[a]; }
984   while (a != b) {
b56     ans = OP(ans, OP(weiop[a], weiop[b]));
c41     a = up[a]; b = up[b];
0b8   }
ba7   return ans;
09b }
```

4.28 Menor anc. comum – Dec. raiz quad.

```
146 int std_lca(int a, int b) {
eb4   if (!(depth[a]/B > depth[b]/B)) { swap(a, b); }
24c   while (depth[a]/B > depth[b]/B) { a = bup[a]; }
54a   if (!(depth[a] > depth[b])) { swap(a, b); }
ebe   while (depth[a] > depth[b]) { a = up[a]; }
4ab   while (a != b) { a = up[a]; b = up[b]; }
3f5   return a;
195 }
```

5 Matemática

5.1 Operações comuns

```
17b ll clamp(ll x, ll lo, ll hi) {
00b   return min(hi, max(x, lo)); }
3bb int msb(ll i) { return i ? __lg(i)+1 : 0; }
5d4 int lsb(ll i) { return msb(i&-i); }
ad1 ll equals_zero_mod_m_inclusive(ll a, ll b, ll m) {
3bf   return b/m - a/m + bool(a%m == 0); }
a7c ll multiples_inclusive(ll a, ll b, ll x) {
ddb   return b/x - (a-1)/x; }
469 ll ceil_div(ll x, ll y) { return (x/y) + bool(x%y); }
b3d ll round_div(ll x, ll y) { return (x+y/2)/y; }
b1a ll mulll(ll a, ll b, ll m) { // a * b % m
e7a   ll ret = a*b - ll((long double)1/m*a*b + 0.5) * m;
074   return ret < 0 ? ret + m : ret;
a0e }
```

5.2 Inteiro modular

```
ae8 template<int p> struct modint {
fed   int x; modint() : x(0) {}
71d   modint(ll a) : x(a) {
1c2     if (!(-p < x && x < p)) x %= p;
9cc     if (x < 0) x += p; }
4da   using m = modint;
3d4 #define DEF(n, b) m& n(const m& a) { b; return *this; }
161   DEF(operator+=, { x += a.x; if (x ≥ p) { x -= p; } })
8dd   DEF(operator-=, { x -= a.x; if (x < 0) { x += p; } })
47f   DEF(operator*=, { x = x * ll(a.x) % p; })
64a   bool operator==(const m& a) { return x == a.x; }
b28   bool operator!=(const m& a) { return x != a.x; }
399   friend m operator+(m a, m b) { return a + b; }
f9e   friend m operator-(m a, m b) { return a - b; }
9c1   friend m operator*(m a, m b) { return a * b; }
162 };
```

5.3 Transformação de Fourier Modular

```
c4b void ntt(vector<mint>& a, bool rev) { // O(n lg n)
6f1   int n = a.size(); auto b = a;
513   mint g = 1;
d39   while (binary_pow(g, P/2) == 1) g += 1;
059   if (rev) { g = binary_pow(g.x, P-2); }
3c4   for (int st = n/2; st; st /= 2) {
c07     mint w = binary_pow(g, P/(n/st)), wn = 1;
eb5     for (int i = 0; i < n/2; i += st) {
424       for (int j = 0; j < st; j++) {
fe4         auto u = a[2*i+j], v = wn * a[2*i+j+st];
69e         b[i+j] = u + v; b[i+n/2+j] = u - v; }
b42     wn *= w; }
257   swap(a, b);
224   }
249   mint n1 = binary_pow(n, P-2);
2fb   if (rev) for (mint& x : a) { x *= n1; }
ec5 }

8f2 vector<mint> convolution (
82e   vector<mint>& a, vector<mint>& b) {
a03   vector<mint> fa (all(a)), fb (all(b));
43e   int n = 1;
58a   while (n < a.size() + b.size()) { n *= 2; }
8e6   fa.resize(n); fb.resize(n);
db6   ntt(fa, 0); ntt(fb, 0);
9e9   for (int i = 0; i < n; i++) { fa[i] *= fb[i]; }
a50   ntt(fa, 1);
83d   return fa;
37d }
```

5.4 Matriz

```
283 using matrix = vector<vector<ll>>;
6f9 matrix identity_matrix() {
485   matrix m (D, vector<ll>(D, 0));
d87   for (int i = 0; i < D; i++) { m[i][i] = 1; }
bd2   return m;
152 }
ec1 matrix multiply(matrix a, matrix b) {
906   matrix ans (D, vector<ll>(D));
f84   for (int i = 0; i < D; i++)
ee9     for (int j = 0; j < D; j++)
26c       for (int k = 0; k < D; k++)
b57         ans[i][j] += a[i][k] * b[k][j];
ba7   return ans;
259 }
```

5.5 Distribuição binomial

A probabilidade do evento X acontecer k vezes é dada por

$$Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

```
94a vector<double> logfact (N);
9e4 void calc() {
042   for (int i = 1; i < N; i++)
f28     logfact[i] = logfact[i-1] + log(i);
540 }
```

```
94c double binom (int n, int k, double p) {
40e   return exp(logfact[n] - logfact[k] - logfact[n - k]
211     + k * log(p) + (n - k) * log (1 - p));
587 }
```

5.6 Integração pelo método de Simpson

```
676 const int N = 3*100; // multiplo de 3
fe3 double integrate(double a, double b,
915   function<double(double)> f) {
621   double s = 0, h = (b - a) / N;
042   for (int i = 1; i < N; i++)
815     s += f(a + i * h) * (i % 3 ? 3 : 2);
0da   return (f(a) + s + f(b)) * 3*h/8;
be5 }
```


5.7 Sequência de Bruijn

Número de sequências distintas:

$$\frac{(k!)^{k^{n-1}}}{k^n}$$

```
042 vector<int> a (K*N, 0), seq;

d5d void de_bruijn(int k, int n, int t=1, int p=1) {
8f7   if (t > n && n % p == 0)
17e     seq.insert(end(seq), begin(a)+1, begin(a)+p+1);
992   if (t > n) { return; }
1cf   a[t] = a[t - p];
5b5   de_bruijn(k, n, t+1, p);
db7   for (int j = a[t-p]+1; j < k; j++) {
7e3     a[t] = j;
99c     de_bruijn(k, n, t+1, t);
ac0   }
677 }
```

5.8 PD de dígito

```
bc4 string num;
20c vector<vector<ll>> mem (20, vector<ll>(12, -1));
1bc ll digit_dp(int i, int l, bool lo, bool nz) {
16b   if (i == -1) { return 1; }
a16   if (lo && nz && mem[i][l] != -1) { return mem[i][l]; }
93b   int lim = lo ? 9 : num[i] - '0';
04b   ll ans = 0;
d5e   for (int d = 0; d <= lim; d++)
47f     if (d != l || (ld && !l && !nz)) {
9da       ans += digit_dp(i-1, d, lo || d < lim, nz || d);
68c     }
696   return (lo && nz ? mem[i][l] = ans : ans);
18c }

1da ll solve(ll b) {
ecb   num = to_string(b); reverse(all(num));
343   return digit_dp(num.size()-1, 10, 0, 0);
319 }
```

5.9 Primeiros dígitos de n^k

```
6f8 int first_d_digits(int n, int k, int d) {
5a1   double logged = k*log10(n);
902   logged -= int(logged) - (d-1);
4bb   return int(pow(10, logged));
be0 }
```

5.10 Problema de Josephus

```
4b2 int josephusrec(int n, int k) { // O(n)
01c   return n ? (josephusrec(n-1, k) + k) % n : 0;
25c }
30b int josephus(int n, int k) { // O(n)
11e   int res = 0;
68a   for (int i = 1; i <= n; ++i)
ae8     res = (res + k) % i;
ee9   return res+1;
8eb }
1c9 int josephuslgn(int n, int k) { // O(k lg n)
447   if (n == 1) { return 0; }
0e2   if (k == 1) { return n-1; }
7cf   if (k > n) { return (josephuslgn(n-1, k) + k) % n; }
fd5   int cnt = n / k;
302   int res = josephuslgn(n - cnt, k);
042   res -= n % k;
994   if (res < 0) { res += n; }
196   else { res += res / (k - 1); }
b50   return res;
8ad }
```

5.11 2-SAT

```
5fa vector<vector<int>> g (2*N), g_t (2*N);
440 vector<int> rep (2*N), vis (2*N);
a2a vector<bool> val (N); stack<int> sinks;
13a int cts = 0;

11d void fill_stack(int u) {
454   if (vis[u] == cts) { return; } vis[u] = cts;
64a   for (int v : g[u]) { fill_stack(v); }
f9d   sinks.push(u);
b56 }

744 void mark_component(int u, int r) {
78f   if (vis[u] == cts) { return; }
3a5   vis[u] = cts; rep[u] = r;
281   for (int v : g_t[u])
e5e     mark_component(v, r);
da7 }

b18 int solve2sat(int n) {
83f   cts++; for (int u = 0; u < 2*n; u++) { fill_stack(u); }
19b   cts++; int c = 0;
aca   while (!sinks.empty()) {
dd1     int u = sinks.top(); sinks.pop();
5c7     mark_component(u, c++);
005   }
687   for (int u = 0; u < n; u++) {
ef8     if (rep[2*u] == rep[2*u+1]) { return 0; }
bdc     val[u] = rep[2*u] > rep[2*u+1];
195   }
6a5   return 1;
8a7 }

be1 void add_implies(int a, bool na, int b, bool nb) {
bc5   a = 2*a+na; b = 2*b+nb;
7d9   g[a].push_back(b); g_t[b].push_back(a);
d17 }

128 void add_disj(int a, bool na, int b, bool nb) {
640   add_implies(a, !na, b, nb);
7c0   add_implies(b, !nb, a, na);
490 }
```

5.12 Multiplicação – Karatsuba

```
2d6 vector<ll> karatsuba /* O(n^{log 3 n}) */ (
b01   const vector<ll> &a, const vector<ll> &b) {
94d   int n = a.size();
1c1   vector<ll> res(n + n);
44d   if (n <= 32) {
830     for (int i = 0; i < n; i++)
f90       for (int j = 0; j < n; j++)
8dd         res[i + j] += a[i] * b[j];
b50     return res;
57f   }
af0   int k = n >> 1;
152   vector<ll> a1(begin(a), begin(a)+k);
9fc   vector<ll> a2(begin(a)+k, end(a));
354   vector<ll> b1(begin(b), begin(b)+k);
3bb   vector<ll> b2(begin(b)+k, end(b));
201   vector<ll> a1b1 = karatsuba(a1, b1);
ddf   vector<ll> a2b2 = karatsuba(a2, b2);
9ef   for (int i=0; i < k; i++) a2[i] += a1[i];
d97   for (int i=0; i < k; i++) b2[i] += b1[i];
f00   vector<ll> r = karatsuba(a2, b2);
b1f   for (int i=0; i < a1b1.size(); i++) r[i] -= a1b1[i];
b21   for (int i=0; i < a2b2.size(); i++) r[i] -= a2b2[i];
21d   for (int i=0; i < r.size(); i++) res[i+k] += r[i];
3c4   for (int i=0; i < a1b1.size(); i++) res[i] += a1b1[i];
9c6   for (int i=0; i < a2b2.size(); i++) res[i+n] += a2b2[i];
b50   return res;
e35 }
```

5.13 Truque de divisão

Gera um conjunto n/i para todo i em $\mathcal{O}(\sqrt{n})$.

```
79c for (int l = 1, r; l <= n; l = r+1) {
746   r = n/(n/l);
// n/i has the same value for l <= i <= r
5bf }
```

5.14 Inclusão-Exclusão

```
748 ll inclusion_exclusion(int n, int m, ll a, ll d) {
e2e   vector<ll> exc { a, a+d, a+2*d, a+3*d, a+4*d };
7af   ll total = 0;
37a   for (int b = 1; b < (1<<5); b++) {
3eb     ll x = 1;
1c3     for (int i = 0; i < 5; i++) if (b & (1<<i)) {
b94       if (x > ceil_div(m, exc[i]) * gcd(x, exc[i])) {
87b         x = b+1; break; }
590       x = lcm(x, exc[i]);
514     }
dea     ll s = (__builtin_popcount(b) % 2) ? +1 : -1;
6a6     total += s * multiples_inclusive(n, m, x);
4fb   }
192   cout << (m-n+1) - total << "\n";
c3e }
```

5.15 Compressão de coordenadas

```
da8 void compress(vector<int>& v, vector<int>& cv) {
fb9   set<int> vs (all(v));
542   map<int, int> to, fm;
529   int ix = 0;
dd1   for (int i : vs) { to[i] = ix; fm[ix] = i; ix++; }
c6e   for (int i = 0; i < v.size(); i++)
bcb     cv[i] = to[v[i]];
825 }
```

5.16 Mínimo excluído com conjunto

```
0c0 struct mex {
5eb   set<int> tomex;
d0e   void insert_mex(int x) { tomex.insert(x); }
928   int get_reset_mex() {
9b4     int m = 0; for (auto &j : tomex) {
a85       if (j > m) { tomex.clear(); return m; }
7b2       m++;
48a     }
70c     tomex.clear(); return m;
4c6   }
d57 };
```

5.17 Mínimo excluído

```
0c0 struct mex {
841   vector<bool> tomex; int gt = 0;
b07   mex(int n) : tomex(n) {};
f97   void insert_mex(int x) {
b39     tomex[x] = 1; gt = max(gt, x);
695   }
928   int get_reset_mex() {
074     int m; for (m = 0; tomex[m]; m++);
425     fill(begin(tomex), begin(tomex)+gt+1, 0);
bd2     return m;
fa4   }
03c };
```

5.18 Número (Números de Grundy)

```
5ac void Grundy() {
0d8   vector<int> g (N, -1);
423   mex m (N);
749   for (int n = 0; n ≤ 1226; n++) {
d3e     if (n == 1 || n == 2) { g[n] = 0; continue; }
41d     for (int i = 1; i ≤ n/2; i++) if (i ≠ n-i)
afa       m.insert_mex(g[i] ^ g[n-i]);
517     g[n] = m.get_reset_mex();
356   }
3fc }
```

5.19 Jogo de Nim

```
5d4 pair<int, int> solve_nim(vector<int> s) {
b6f   int n = s.size(), g = 0;
830   for (int i = 0; i < n; i++)
a32     g ^= s[i];
```

```
70e   if (g == 0) { return {-1, -1}; }
a2a   int msb = 1<<__lg(g);
19e   for (int i = 0; i < n; i++) if (s[i]&msb) {
736     int t = s[i]^g;
e7f     return { s[i]-t, i+1 };
cf5   }
172   assert(0);
057 }
```

5.20 Euclides estendido/inv. multiplicativo

$$ax + by = \gcd(a, b)$$

$$ax = \gcd(a, b) \pmod{b}$$

$$(\gcd(a, b) = 1) \implies (ax = 1 \pmod{b})$$

```
3b2 ll extgcd(ll a, ll b, ll& x, ll& y) { // O(lg min(a, b))
433   if (b == 0) { x = 1; y = 0; return a; }
a59   ll g = extgcd(b, a%b, x, y);
e8f   tie(x, y) = make_tuple(y, x - (a/b)*y);
96b   return g;
52b }
```

```
2da ll inv(ll a, ll m) {
306   ll x, y; extgcd(a, m, x, y);
1b2   return ((x % m) + m) % m;
ae2 }
```

```
2ba vector<ll> invall (N);
9e4 void calc() {
2ff   invall[1] = 1;
ffa   for (int i = 2; i < N; i++)
2df     invall[i] = P - P / i * invall[P % i] % P;
dab }
```

5.21 Números de Catalão

```
9bd ll catalan(int n) {
364   if (n == 0) { return 1; }
df5   ll a = (2*(2*(n-1)+1)) % P;
8cd   ll b = n+1;
ddd   ll b_inv = (extgcd(b, P).x + P) % P;
e15   ll res = (((a*b_inv) % P) * catalan(n-1)) % P;
b50   return res;
b23 }
```

5.22 Detecção de ciclo

```
885 struct cyc { ll prev, first, len; };
906 cyc find_cycle(ll start) {
e92   ll a = start, b = start;
ef6   do { a = succ(a); b = succ(succ(b)); } while (a ≠ b);
e9b   a = start;
283   ll prev = -1;
eda   while (a ≠ b) { prev = a; a = succ(a); b = succ(b); }
004   ll len = 0;
fb0   do { b = succ(b); len++; } while (a ≠ b);
dfe   return { .prev = prev, .first = a, .len = len };
f06 }
```

5.23 Equação diofantina linear

$$ax + by = c$$

```
99a vector<ii> solve(ll a, ll b, ll c, int i) {
ebf   vector<ii> sol;
d73   ll x, y, g = extgcd(a, b, x, y);
fe3   if (c % g) { return; }
a33   x *= c/g; y *= c/g;
b12   while (i--) {
d3a     sol.push_back(ii(x, y));
f38     x += b/g; y -= a/g;
642   }
6c7   return sol;
d05 }
```


5.24 Fatoração por tentativa

```
39e map<int, int> factorize(ll n) { //  $O(\sqrt{n})$ 
ece map<int, int> v;
b6d for (ll i = 2; i*i ≤ n; i++)
49e while (n % i == 0)
c9a n /= i, v[i]++;
bd1 if (n > 1) { v[n]++; }
6dc return v;
f9b }
```

5.25 Crivo de Eratóstenes

```
da9 vector<bool> sieve (1e7+15, 1);
232 void eratosthenes(int n) { //  $O(n \lg \lg n)$ 
494 for (int i = 2; i * i ≤ n; i++) if (sieve[i])
c0a for (int j = i * i; j ≤ n; j += i)
116 sieve[j] = 0;
2a7 }
```

5.26 Totiente de Euler

```
5b7 vector<ll> phi (N);
ef5 void totient(int n) { //  $O(n \lg n)$ 
178 for (int i = 1; i ≤ n; i++) { phi[i] = i; }
f03 for (int i = 2; i ≤ n; i++) if (phi[i] == i) {
c6c for (int j = i; j ≤ n; j += i) {
a9b phi[j] -= phi[j] / i;
5b7 }
7d1 }
596 }
```

5.27 Eliminação gaussiana

```
67a template<typename T>
316 pair<int, vector<T>> gauss( //  $O(mn^2)$ 
017 vector<vector<T>> a, vector<T> b) {
6ca const double eps = 1e-6;
f92 int n = a.size(), m = a[0].size();
2f0 for (int i = 0; i < n; i++) a[i].push_back(b[i]);

3cb vector<int> where(m, -1);
237 for (int col = 0, row = 0; col < m and row < n; col++) {
f05 int sel = row;
b95 for (int i = row; i < n; ++i)
e55 if (abs(a[i][col]) > abs(a[sel][col])) sel = i;
2c4 if (abs(a[sel][col]) < eps) continue;
1ae for (int i = col; i ≤ m; i++)
dd2 swap(a[sel][i], a[row][i]);
2c3 where[col] = row;
0c0 for (int i = 0; i < n; i++) if (i ≠ row) {
96c T c = a[i][col] / a[row][col];
d5c for (int j = col; j ≤ m; j++)
c8f a[i][j] -= a[row][j] * c;
490 }
b70 row++;
3d8 }

b1d vector<T> ans(m, 0);
e1a for (int i = 0; i < m; i++) if (where[i] ≠ -1)
12a ans[i] = a[where[i]][m] / a[where[i]][i];
603 for (int i = 0; i < n; i++) {
501 T sum = 0;
a75 for (int j = 0; j < m; j++)
5a9 sum += ans[j] * a[i][j];
b1f if (abs(sum - a[i][m]) > eps)
6cd return pair(0, vector<T>());
ec9 }

12e for (int i = 0; i < m; i++) if (where[i] == -1)
018 return pair(INF, ans);
280 return pair(1, ans);
18a }
```

5.28 Exponenciação binária

```
b2f mint binary_pow(mint a, ll e) {
2ce if (e == 0) { return 1; }
```

```
fb8 mint res = binary_pow(a, e/2);
b7c res *= res;
bd7 if (e % 2) { res *= a; }
b50 return res;
4c7 }
```

5.29 Miller-Rabin

```
f17 using i128 = __int128_t;
706 ll binary_pow(ll a, ll e, ll m) {
2ce if (e == 0) { return 1; }
08c if (e == 1) { return a; }
0b7 ll res = binary_pow(a, e/2, m);
342 res = (i128)res * res % m;
d2e if (e % 2) res = (i128)res * a % m;
b50 return res;
ab5 }

234 vector<int> witnesses = {
9a9 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 };
0a0 bool is_prime(ll n) {
992 if (n < 2) { return 0; }
3b4 int s = __lg((n-1)&-(n-1));
70f ll d = n >> s;
3a6 for (int a : witnesses) {
8bd if (n == a) { return 1; }
d2b ll p = binary_pow(a, d, n), i = s;
274 while (p ≠ 1 && p ≠ n - 1 && a % n && i--)
516 p = (i128)p * p % n;
25c if (p ≠ n - 1 && i ≠ s) { return 0; }
23d }
6a5 return 1;
4f9 }
```

5.30 Pollard Rho

```
79f ll pollard(ll n) { //  $O(n^{\frac{1}{4}})$ 
a55 auto f = [n](ll x) {
622 return (fast_pow(x, x, n) + 1) % n;
3cf };
17d if (n % 2 == 0) { return 2; }
4a7 for (ll i = 2; ; i++) {
6a0 ll x = i, y = f(x), p;
2d5 while ((p = gcd(n + y - x, n)) == 1)
b78 x = f(x), y = f(f(y));
e95 if (p ≠ n) { return p; }
9bc }
d08 }
```

```
fb5 void factorize(vector<ll>& f, ll n) {
4c1 if (n == 1) { return; }
3df if (is_prime(n)) { f.push_back(n); return; }
ac6 ll x = pollard(n);
6e7 factorize(f, x); factorize(f, n/x);
5d7 }
```

5.31 Teorema chinês do resto generalizado

```
060 ll norm(ll a, ll b) {
ceb a %= b; return (a < 0) ? a + b : a;
318 }

b49 pair<ll, ll> crt_single(ll a, ll n, ll b, ll m) {
d0a ll x, y; ll g = extgcd(n, m, x, y);
9bc if ((a - b) % g) { return {-1, -1}; }
26f ll lcm = (m/g) * n;
9f9 return {norm(a + n*(x*(b-a)/g % (m/g)), lcm), lcm};
1c6 }

// infinite solutions ± lcm( $m_1 m_2 \dots m_t$ )
//  $O(t \lg m_1 m_2 \dots m_t)$ 
2f0 ll crt(vector<ll> a, vector<ll> m, int t) {
565 ll ans = a[0], lcm = m[0];
aac for (int i = 1; i < t; ++i) {
e5b tie(ans, lcm) = crt_single(ans, lcm, a[i], m[i]);
650 if (ans == -1) { return -1; }
404 }
ba7 return ans;
184 }
```

6 Strings

6.1 KMP

```
730 vector<int> pre(string ne) {
954     int n = ne.size();
a9b     vector<int> pi (n, 0);
5f9     for (int i = 1, j = 0; i < n; i++) {
5df         while (j > 0 && ne[i] != ne[j]) { j = pi[j-1]; }
b54         if (ne[i] == ne[j]) { j++; }
d2e         pi[i] = j;
d89     }
81d     return pi;
4a4 }
```

```
df4 int search(string hay, string ne) {
52d     vector<int> pi = pre(ne);
aff     int c = 0;
cfe     for (int i = 0, j = 0; i < hay.size(); i++) {
75f         while (j > 0 && hay[i] != ne[j]) { j = pi[j-1]; }
965         if (hay[i] == ne[j]) { j++; }
981         if (j == ne.size()) {
6a4             c++; /* match at (i-j+1) */
25c             j = pi[j-1];
127         }
1fb     }
807     return c;
5ed }
```

6.2 Algoritmo Z

$z[i]$ é o tamanho da maior string que é ao mesmo tempo, um prefixo de s e um prefixo do sufixo de s começando em i .

```
5cb vector<int> z_function(string s) {
737     int n = (int)s.length();
2e8     vector<int> z (n);
89f     for (int i = 1, l = 0, r = 0; i < n; ++i) {
c2f         if (i ≤ r)
933             z[i] = min (r - i + 1, z[i - l]);
8e0         while (i + z[i] < n && s[z[i]] == s[i + z[i]])
424             ++z[i];
881         if (i + z[i] - 1 > r)
ff7             l = i, r = i + z[i] - 1;
fc0     }
070     return z;
85d }
```

6.3 String Hashing

```
d7d struct str_hash {
dcf     vector<ll> h, p;
ea8     str_hash(string s) : h(s.size()), p(s.size()) {
7a2         p[0] = 1, h[0] = s[0];
ad7         for (int i = 1; i < s.size(); i++)
99b             p[i] = p[i-1]*P%M, h[i] = (h[i-1]*P + s[i])%M;
4f0     }
af7     ll operator()(int l, int r) { // retorna hash s[l...r]
1e2         ll hash = h[r] - (l ? h[l-1]*p[r-l+1]%M : 0);
990         return hash < 0 ? hash + M : hash;
ebf     }
840 };
```

```
3c9 struct node {
820     char c; ll h; int sz;
601     node() : h(1), sz(0) {}
448     explicit node(char _c) : sz(1), c(_c), h(_c-'a'+1) {}
5e0     node operator+ (node o) {
f62         node a; a.h = (h + (o.h*p[sz])%M) % M;
7b1         a.sz = sz + o.sz;
3f5         return a;
399     }
8ca };
```

6.4 Autômato KMP

```
e2b vector<vector<int>> kmp_automaton(string s) {
aa0     s += '#'; int n = s.size();
e88     vector<int> pi = pre(s);
dc5     vector<vector<int>> aut (n, vector<int>(26));
830     for (int i = 0; i < n; i++)
1e1         for (int c = 0; c < 26; c++) {
a5b             int j = i;
3a6             while (j > 0 && 'a' + c != s[j])
25c                 j = pi[j-1];
dc1             if ('a' + c == s[j]) { j++; }
679             aut[i][c] = j;
dbd         }
8a7     return aut;
c57 }
```

6.5 Autômato de Sufixo

```
637 int last = 0, sz = 1;
b10 vector<int> len (2*N), lnk (2*N), acc (2*N);
2d6 vector<vector<int>> nxt (2*N, vector<int>(S));
```

```
e6a void add(int c) {
715     int cur = sz++; len[cur] = len[last]+1;
8bb     int p = last; last = cur;
40c     for (; ~p && !nxt[p][c]; p = lnk[p])
3de         nxt[p][c] = cur;
f11     if (!p) { lnk[cur] = 0; return; }
244     int q = nxt[p][c];
96f     if (len[q] == len[p]+1) { lnk[cur] = q; return; }
c09     int qq = sz++; len[qq] = len[p]+1; lnk[qq] = lnk[q];
49b     copy(all(nxt[q]), begin(nxt[qq]));
5cf     for (; ~p && nxt[p][c] == q; p = lnk[p])
753         nxt[p][c] = qq;
3be     lnk[cur] = lnk[q] = qq;
b81 }
```

```
94e void build(string& s) {
f8a     len[0] = 0, lnk[0] = -1;
d8b     for (char ch : s) { add(to_i(ch)); }
0b8     for (int u = last; u; u = lnk[u]) { acc[u] = 1; }
24d }
```

```
7b6 vector<int> cnt (2*N);
6c1 vector<int> ord;
a75 void reverse_topo(int u) {
558     cnt[u] = 1;
c3a     for (int c = 0; c < S; c++) if (nxt[u][c])
362         if (!cnt[nxt[u][c]]) { reverse_topo(nxt[u][c]); }
153     ord.push_back(u);
d3d }
```

```
815 void process_count() {
289     for (int u : ord) {
c88         cnt[u] = 0;
c3a         for (int c = 0; c < S; c++) if (nxt[u][c])
8a8             cnt[u] += cnt[nxt[u][c]];
d38         if (acc[u]) { cnt[u]++; }
df7     }
080 }
```

```
d8e int longest_common_substring(string& t) {
771     int u = 0, l = 0, ans = 0, pos = -1;
c9d     for (int i = 0; i < t.size(); i++) {
191         int c = t[i];
6ed         while (u && !nxt[u][c]) {
223             u = lnk[u]; l = len[u]; }
327         if (nxt[u][c]) { u = nxt[u][c]; l++; }
032         else { u = 0; l = 0; }
2d6         if (l > ans) { ans = l; pos = i; }
c50     }
ba7     return ans;
dd3 }
```

```
e90 int search(string& t) {
42a     int u = 0; for (auto c : t) {
158         u = nxt[u][to_i(c)]; if (!u) { return 0; }
47f     }
206     return cnt[u];
e62 }
```

6.6 Vetor de sufixos

```
9af #define kk first
f89 #define ii second

08b pair<vector<int>, vector<int>>> build_sa(string s) {
0a3   int n = s.size(); vector<int> sk (all(s));
db9   vector<pair<pair<int, int>, int>>> a(n);
d3f   for (int k = 1; k < n; k *= 2) {
830     for (int i = 0; i < n; i++)
be6       a[i] = { { sk[i], sk[(i+k)%n] }, i };
a6d     sort(begin(a), end(a)); sk[a[0].ii] = 0;
c0d     for (int i = 1, r = 0; i < n; i++)
df1       sk[a[i].ii] = a[i-1].kk == a[i].kk ? r : ++r;
3ff   }
4bf   vector<int> sa (n);
96a   for (int i = 0; i < n; i++) { sa[i] = a[i].ii; }
853   return make_pair(sa, sk);
a77 }

b5d vector<int> make_lcp(vector<int> sa, vector<int> sk) {
795   vector<int> lcp (n); int k = 0;
f50   for (int i = 0; i < n-1; i++) {
607     int pi = sk[i]; int j = sa[pi-1];
fe7     while (s[i+k] == s[j+k]) k++;
435     lcp[pi] = k; k = max(k - 1, 0);
b34   }
5ed   return lcp;
a23 }
```

6.7 Vetor de sufixos radix

```
151 pair<vector<int>, vector<int>>> build_sa(string s, int n) {
b59   vector<int> sk (n), sa (n);
f96   vector<pair<int, int>> a (n);
9f5   for (int i = 0; i < n; i++) { a[i] = { s[i], i }; }
0b5   sort(all(a));
87d   for (int i = 0; i < n; i++) { tie(sk[i], sa[i]) = a[i]; }
4b9   vector<int> nsk(n);
c0d   for (int i = 1, r = 0; i < n; i++)
34b     nsk[sa[i]] = (sk[i-1] == sk[i] ? r : ++r);
d3c   sk.swap(nsk);
d3f   for (int k = 1; k < n; k *= 2) {
e12     for (int i = 0; i < n; i++) { sa[i] = (sa[i]-k+n)%n; }
d40     vector<int> nsa(n), cnt(n+1);
c4f     for (int x : sk) { cnt[x+1]++; }
265     for (int i = 1; i < n; i++) { cnt[i] += cnt[i-1]; }
320     for (int x : sa) { nsa[cnt[sk[x]]++] = x; }
8e0     sa.swap(nsa);
4b9     vector<int> nsk(n);
c0d     for (int i = 1, r = 0; i < n; i++)
355       nsk[sa[i]] =
4f8         make_pair(sk[sa[i-1]], sk[(sa[i-1]+k)%n]) ==
554         make_pair(sk[sa[i]], sk[(sa[i]+k)%n]) ? r : ++r;
d3c     sk.swap(nsk);
f42   }
853   return make_pair(sa, sk);
86d }
```

6.8 Aho-Corasick/Trie

```
3c9 struct node {
682   int p, pc, depth, lnk, out, occ; bool leaf;
613   vector<int> nxt, go, ix;
3cc   node() : p (0), pc (0), depth (-1),
b70     lnk (-1), out (-1), occ (0),
799     leaf (0), nxt (S, -1), go (S, -1) {}
3e8 };

2d3 vector<node> aca (1);
e45 void ins(string ne, int ix) {
ae5   int u = 0; for (int i = 0; i < ne.size(); i++) {
49c     int ch = to_i(ne[i]);
754     if (aca[u].nxt[ch] == -1) {
847       aca[u].nxt[ch] = aca.size();
2c5       node n; n.p = u; n.pc = ch; n.depth = i;
845       aca.push_back(n);
043     }
96f     u = aca[u].nxt[ch];
```

```
ed4   }
683   aca[u].leaf = 1; aca[u].ix.push_back(ix);
aa6 }

9b7 int go(int u, int c);
fd6 int link(int u) {
78d   if (aca[u].lnk != -1) { return aca[u].lnk; }
7f5   if (u == 0 || aca[u].p == 0) { return aca[u].lnk = 0; }
06b   return aca[u].lnk = go(link(aca[u].p), aca[u].pc);
cf4 }

8b3 int go(int u, int c) {
efd   if (aca[u].go[c] != -1) { return aca[u].go[c]; }
3de   if (aca[u].nxt[c] != -1)
3da     return aca[u].go[c] = aca[u].nxt[c];
2c9   if (u == 0) { return aca[u].go[c] = 0; }
7c4   return aca[u].go[c] = go(link(u), c);
bdc }

a8f int out(int u) {
665   if (aca[u].out != -1) { return aca[u].out; }
cfd   int v = link(u);
d98   if (v == 0 || aca[v].leaf) { return aca[u].out = v; }
3eb   return aca[u].out = out(v);
2b2 }

02f vector<int> occ (N); vector<vector<int>>> occix (N);
9af void process(string hay) {
a7d   int u = 0; for (int i = 0; i < hay.size(); i++) {
3f5     int ch = to_i(hay[i]); u = go(u, ch);
7e9     for (int v = u; v != 0; v = out(v)) {
58b       for (auto ix : aca[v].ix)
// match ix O(len + ans)
b4f         occix[ix].push_back(i - aca[v].depth);
380         aca[v].occ++;
141       }
005     }
d06   for (int u = 0; u < aca.size(); u++)
acb     for (auto ix : aca[u].ix)
1a7       occ[ix] += aca[u].occ; // # of matches O(len)
0ce }
```

6.9 Árvore de sufixos

```
182 vector<vector<int>>> nxt (N, vector<int>(5));
973 vector<int> l (N), r (N), parent (N), suf (N);
64e int len(int i) { return r[i] - l[i] + 1; }
6f2 int at(string& s, int i, int j) { return id(s[l[i] + j]); }
941 int cur = 1;
a7f int mknnode(int a, int b, int p) {
86a   l[cur] = a, r[cur] = b; parent[cur] = p;
daa   return cur++;
cd1 }

fec void build(string s) { s += '$';
a4d   int root = mknnode(0, -1, 0);
c9e   int u = root, i = 0, ui = 0, ns = 0;
303   for (int j = 0; j < s.size(); j++) for (; i ≤ j; i++) {
7c5     if (ui == len(u) && nxt[u][id(s[j])])
dd6       { u = nxt[u][id(s[j])]; ui = 0; }
d5c     if (ui < len(u) && at(s, u, ui) == id(s[j]))
828       { ui++; break; }
fde     if (ui == len(u)) {
5a1       nxt[u][id(s[j])] = mknnode(j, s.size()-1, u);
914       if (u != root) { u = suf[u]; ui = len(u); }
9d9     } else {
d5f       int mi = mknode(l[u], l[u] + ui - 1, parent[u]);
3ce       nxt[parent[u]][at(s, mi, 0)] = mi;
847       nxt[mi][at(s, u, ui)] = u;
7d7       parent[u] = mi; l[u] += ui;
834       nxt[mi][id(s[j])] = mknnode(j, s.size()-1, mi);
7b5       if (ns) { suf[ns] = mi; }
f79       u = parent[mi]; int g;
5c4       if (u != root) { u = suf[u]; g = j - ui; }
902       else { g = i + 1; }
042       while (g < j && g + len(nxt[u][id(s[g])]) ≤ j) {
f9b         u = nxt[u][id(s[g])]; g += len(u); }
dec       if (g == j) { ns = 0; suf[mi] = u; ui = len(u); }
22d       else { ns = mi; u = nxt[u][id(s[g])]; ui = j - g; }
538     }
0e8   }
d39 }
```

7 Geometria

7.1 Pontos

```
375 using pt = complex<double>;
88b #define px real()
943 #define py imag()
0c5 double dot(pt a, pt b) { return (conj(a) * b).px; }
a51 double cross(pt a, pt b) { return (conj(a) * b).py; }
b77 pt vec(pt a, pt b) { return b - a; }
dc5 int sgn(double v) { return (v > -EPS) - (v < EPS); }
// -1 (cw), 0 (colinear), +1 (ccw)
223 int seg_ornt(pt p, pt a, pt b) {
939 return sgn(cross(vec(p, a), vec(p, b)));
f41 }
274 int ccw(pt a, pt b, pt c, bool col) {
126 int o = seg_ornt(a, b, c);
4fb return (o == 1) || (o == 0 && col);
6ae }
1d5 const double PI = acos(-1);
5cb double angle(pt a, pt b, pt c) {
a1f return abs(remainder(arg(a-b) - arg(c-b), 2.0 * PI));
7fd }
```

7.2 Fecho convexo de Graham

```
1ff vector<pt> convex_hull(vector<pt>& ps, bool col = 0) {
024 pt p0 = *min_element(all(ps), [](pt a, pt b) {
5f6 return make_pair(a.py, a.px) < make_pair(b.py, b.px);
c0c });
ec6 sort(all(ps), [&p0](pt a, pt b) {
481 int o = seg_ornt(p0, a, b);
cf6 return o < 0 || (o == 0 && norm(p0-a) < norm(p0-b));
c0c });
bf5 if (col) {
1f5 int i = ps.size(); i--;
047 while (i >= 0 && seg_ornt(p0, ps[i], ps.back()) == 0)
169 i--;
a07 reverse(begin(ps)+i+1, end(ps));
36d }
63e vector<pt> ans; for (int i = 0; i < ps.size(); i++) {
28e while (ans.size() > 1 && !ccw(
347 ps[i], ans.back(), ans[ans.size()-2], col))
bba ans.pop_back();
702 ans.push_back(ps[i]);
710 }
ba7 return ans;
f6f }
```

7.3 Fecho convexo com corrente monotônica

```
1ff vector<pt> convex_hull(vector<pt>& ps, bool col = 0) {
4a3 int k = 0, n = ps.size(); vector<pt> ans(2*n);
4d9 sort(all(ps), [](pt a, pt b) {
395 return make_pair(a.px, a.py) < make_pair(b.px, b.py);
c0c });
603 for (int i = 0; i < n; i++) {
e02 while (k >= 2 && !ccw( /* lower hull */
f50 ans[k-2], ans[k-1], ps[i], col)) { k--; }
e98 ans[k++] = ps[i];
1ae }
bbb if (k == n) { ans.resize(n); return ans; }
95e for (int i = n-2, t = k+1; i >= 0; i--) {
971 while (k >= t && !ccw( /* upper hull */
f50 ans[k-2], ans[k-1], ps[i], col)) { k--; }
e98 ans[k++] = ps[i];
1d8 }
f56 ans.resize(k-1); return ans;
f81 }
```

7.4 Distância máxima

```
555 ld abs_area(pt p, pt q, pt r) {
1ae return abs((p.px*q.py + q.px*r.py + r.px*p.py) -
a2e (p.py*q.px + q.py*r.px + r.py*p.px));
481 }
```

```
981 ld rotating_calipers(vector<pt>& ps) {
```

```
b21 vector<pt> h = convex_hull(ps); int n = h.size();
447 if (n == 1) { return 0; }
1fe if (n == 2) { return abs(h[0] - h[1]); }
bfe int k = 1;
0ed while (abs_area(h[n-1], h[0], h[(k+1)%n]) >
215 abs_area(h[n-1], h[0], h[k])) k++;
b0c ld ans = 0;
770 for (int i = 0, j = k; i <= k; i++) {
ea5 while (abs_area(h[i], h[(i+1)%n], h[(j+1)%n]) >
4de abs_area(h[i], h[(i+1)%n], h[j])) {
f7d ans = max(ans, abs(h[i] - h[(j+1)%n]));
600 j = (j+1) % n;
f4f }
3ee ans = max(ans, abs(h[i] - h[j]));
88e }
ba7 return ans;
a60 }
```

7.5 Árvore KD

```
dd1 int get_ii(pt pair, int ix) {
9c9 return ix == 0 ? pair.px : pair.py; }

933 struct kdtree {
3c9 struct node {
242 node(pt& p) : p(p), left(0), right(0) {}
e77 double dist_sq(const pt& o) { return norm(o - p); }
54f pt p; node* left, *right;
3e3 };
847 node* root = 0; vector<double> best; vector<node> t;

d96 struct cmp {
04c size_t ix; cmp(size_t _index) : ix(_index) {}
a10 bool operator()(const node& n1, const node& n2) {
e65 return get_ii(n1.p, ix) < get_ii(n2.p, ix);
afc }
562 };

964 node* make_tree(size_t begin, size_t end, size_t ix) {
c6a if (end <= begin) { return 0; }
c0e size_t n = begin + (end - begin)/2;
3ed nth_element(&t[begin], &t[n], &t[0] + end, cmp(ix));
0f5 ix = (ix + 1) % 2;
e66 t[n].left = make_tree(begin, n, ix);
a02 t[n].right = make_tree(n + 1, end, ix);
8b4 return &t[n];
c20 }

5de void nearest_k(node* r, const pt& p, size_t ix, int k) {
c9e if (r == nullptr) { return; }
40e double d = r->dist_sq(p);
1f0 if (best.size() < k || d < best.back()) {
923 best.push_back(d); sort(all(best));
8a9 if (best.size() > k)
30d best.erase(begin(best)+k, end(best));
712 }
eeb if (best.size() == 0) { return; }
b64 double dx = get_ii(r->p, ix) - get_ii(p, ix);
0f5 ix = (ix + 1) % 2;
a46 nearest_k(dx > 0 ? r->left : r->right, p, ix, k);
24d if (dx * dx >= best[best.size()-1]) { return; }
886 nearest_k(dx > 0 ? r->right : r->left, p, ix, k);
766 }

9d0 template<typename iterator>
058 kdtree(iterator begin, iterator end) : t(begin, end) {
6af root = make_tree(0, t.size(), 0);
5f4 }

488 double nearest_k(const pt& p, int k) {
7be if (root == 0) { throw logic_error("empty tree"); }
927 best.clear(); nearest_k(root, p, 0, k);
531 double acc = 0;
5a0 for (int i = 0; i < best.size(); i++)
43c acc += sqrt(best[i]);
b8b return acc;
86f }
07a };
```

7.6 Ponto dentro do polígono?

```
c88 int is_inside_poly(vector<pt>& ps, pt p, int strict) {
bc4  int n = ps.size();
0bd  if (n < 3) { return 0; }
b52  int count = 0;
603  for (int i = 0; i < n; i++) {
4d0    int j = (i+1)%n;
18b    if (on_segment(seg(ps[i], ps[j]), p))
db6      return strict;
cc2    count ^= (((p.py < ps[i].py) - (p.py < ps[j].py))
89c      * ord_ornt(seg(ps[i], ps[j]), p)) > 0;
02c  }
308  return count;
415 }
```

7.7 Ponto dentro do polígono convexo?

```
62e bool is_inside_poly(vector<pt>& ps, pt p) {
bc4  int n = ps.size();
db8  double theta = 0;
603  for (int i = 0; i < n; i++) {
27b    theta += (seg_ornt(ps[i], ps[(i+1)%n], p)
683      ≥ 0 ? +1 : -1)
399      * angle(ps[i], p, ps[(i+1)%n]);
024  }
969  return abs(theta - 2*PI) < EPS;
f29 }
```

7.8 Área de polígono

```
6a1 double polygon_area(vector<pt>& ps) {
f5d  double area = cross(ps[ps.size()-1], ps[0]);
01b  for (int i = 1; i < ps.size(); i++) {
7e8    area += cross(ps[i-1], ps[i]);
0fd  }
6c1  return area / 2.0;
b99 }
```

7.9 Pontos inteiros no polígono

$$A = i + b/2 - 1$$

```
076 ll lattice_points(vector<pt>& ps) {
c4e  ll b = 0;
603  for (int i = 0; i < n; i++) {
0e6    pt v = vec(ps[i], ps[(i+1)%n]);
b3f    if (v.py == 0) { b += abs(v.px); }
80d    else if (v.px == 0) { b += abs(v.py); }
a38    else { b += gcd(abs(v.px), abs(v.py)); }
da4  }
a7c  return poly_area(ps) - b/2 + 1;
529 }
```

7.10 Segmento

```
8b0 #define aa first
3d1 #define bb second
8e4 using seg = pair<pt, pt>;
b26 pt vec(seg s) { return vec(s.aa, s.bb); }
325 int ord_ornt(seg s, pt c) {
5c6  return seg_ornt(s.aa, s.bb, c);
65b }
```

7.11 Ponto e segmento

```
70b pt point_in_seg(pt p, seg s) {
963  if (s.aa == s.bb) { return s.aa; }
aec  double l = dot(vec(s.aa, p), vec(s)) / norm(vec(s));
d50  if (l < 0.0) { return s.aa; }
ad7  if (l > 1.0) { return s.bb; }
857  return s.aa + l*vec(s);
151 }

773 bool on_segment(seg s, pt p) {
```

```
1af  return abs(cross(vec(p, s.aa), vec(p, s.bb))) < EPS
8bc  && dot(vec(p, s.aa), vec(p, s.bb)) < EPS;
16b }:
```

7.12 Manhattan máximo

```
297 ll max_dist_manhattan(vector<pair<ll, ll>> v) {
99e  ll min_sum, max_sum, min_dif, max_dif;
4f5  min_sum = max_sum = v[0].first + v[0].second;
271  min_dif = max_dif = v[0].first - v[0].second;
c25  for (auto [x, y] : v) {
1cb    min_sum = min(min_sum, x+y);
683    max_sum = max(max_sum, x+y);
782    min_dif = min(min_dif, x-y);
af7    max_dif = max(max_dif, x-y);
e3a  }
9f0  return max(max_sum - min_sum, max_dif - min_dif);
dc0 }
```

7.13 Voronoi

```
bf7 using ii = pair<int, int>; using ld = long double;
d39 const ld EPS = 1e-9;
```

```
f05 pt rot(pt a) { return pt(-a.py, a.px); }
ce0 ld sq(ld x) { return x*x; }
```

```
// precondition is that they aren't collinear
bcd pt point_in_line_line(seg a, seg b) {
8a4  return a.aa + vec(a) * (cross(vec(a.aa, b.aa), vec(b))
7f6    / cross(vec(a), vec(b)));
b55 }
117 pt line_intersect(pt a, pt b, pt u, pt v) {
1f3  return point_in_line_line(seg(u, u + v),
c67    seg(a, a + b));
2fe }
```

```
// precondition is that abc is a non-degenerate triangle
bc2 pt circumcenter(pt a, pt b, pt c) {
230  b = (a + b) * ld(.5);
406  c = (a + c) * ld(.5);
0b8  return point_in_line_line(seg(b, b + rot(vec(a, b))),
5eb    seg(c, c + rot(vec(a, c))));
427 }
```

```
2e7 ld parab_intersect(pt l, pt r, ld sw) {
1a0  if (abs(l.py - r.py) < abs(l.px - r.px) * EPS) {
82f    int sign = l.px < r.px ? 1 : -1;
964    pt m = 0.5l * (l+r);
ae0    pt v = line_intersect(m, rot(r-l), pt(0,sw), pt(1,0));
b50    pt w = line_intersect(m, rot(l-v), v, l-v);
017    ld l1 = abs(v-w);
e47    ld l2 = sqrt(sq(sw-m.py) - norm(m-w));
3be    ld l3 = abs(l-v);
c4d    return v.px + (m.px - v.px) * l3 / (l1 + sign * l2);
23e  }
61d  int sign = l.py < r.py ? -1 : 1;
28a  pt v = line_intersect(l, r-l, pt(0, sw), pt(1, 0));
c36  ld d1 = norm(0.5l * (l+r) - v);
691  ld d2 = norm(0.5l * (l-r));
8f7  return v.px + sign * sqrt(max(0.0l, d1 - d2));
2b4 }
```

```
c99 struct beach {
8ef  struct arc {
73a    arc() {}
8f0    arc(pt p, int ix) : p(p), ix(ix), end(0),
074      link{0, 0}, par(0), prv(0), nxt(0) {}
5f8    pt p; int ix; int end;
015    arc *link[2], *par, *prv, *nxt;
544  };
760  arc *root; ld sw;
aac  beach() : sw(-1e20), root(0) {}
63d  inline int dir(arc *x) { return x->par->link[0] != x; }
95c  void rotate(arc *n) {
92d    arc *p = n->par; int d = dir(n);
ca7    p->link[d] = n->link[!d];
d10    if (n->link[!d]) { n->link[!d]->par = p; }
4d7    n->par = p->par;
```



```

9b8     if (p->par) { p->par->link[dir(p)] = n; }
320     n->link[id] = p; p->par = n;
70b }
5d3 void splay(arc *x, arc *f = 0) {
255     while (x->par != f) {
70d         if (x->par->par == f);
069         else if (dir(x) == dir(x->par)) rotate(x->par);
f66         else { rotate(x); }
64f         rotate(x);
827     }
604     if (f == 0) { root = x; }
29b }
911 void insert(arc *n, arc *p, int d) {
b2d     splay(p); arc* c = p->link[d];
198     n->link[d] = c; if (c) { c->par = n; }
156     p->link[d] = n; n->par = p;
a81     arc* prv = !d ? p->prv : p;
d9d     arc* nxt = !d ? p : p->nxt;
bdd     n->prv = prv; if (prv) { prv->nxt = n; }
8d3     n->nxt = nxt; if (nxt) { nxt->prv = n; }
f6e }
7c7 void erase(arc* n) {
ed5     arc *prv = n->prv, *nxt = n->nxt;
cdf     if (!prv && !nxt) {
e92         if (n == root) { root = 0; } return;
640     }
b86     n->prv = 0; if (prv) { prv->nxt = nxt; }
02a     n->nxt = 0; if (nxt) { nxt->prv = prv; }
054     splay(n);
dd7     if (!nxt) {
321         root->par = n->link[0] = 0; root = prv;
9d9     } else {
a35         splay(nxt, n);
dc8         arc* c = n->link[0]; nxt->link[0] = c; c->par = nxt;
349         n->link[0] = n->link[1] = nxt->par = 0; root = nxt;
71b     }
63e }
8f7 bool get(arc* cur, ld &next_sweep) {
28f     if (!cur->prv || !cur->nxt) { return 0; }
014     pt u = rot(cur->p - cur->prv->p);
ab7     pt v = rot(cur->nxt->p - cur->p);
83d     if (sgn(cross(u, v)) != 1) { return 0; }
8c1     pt p = circumcenter(cur->p, cur->prv->p, cur->nxt->p);
405     next_sweep = p.py + abs(p - cur->p);
6a5     return 1;
274 }
2e3 arc* find_beachline(ld x) {
ea3     arc* cur = root;
314     while (cur) {
052         ld l = cur->prv ?
785             parab_intersect(cur->prv->p, cur->p, sw) : -1e30;
c01         ld r = cur->nxt ?
831             parab_intersect(cur->p, cur->nxt->p, sw) : 1e30;
079         if (l <= x && x <= r) { splay(cur); return cur; }
572         cur = cur->link[x > r];
c02     }
bb3     return 0;
30a }
010 }; using arc = beach::arc;

016 struct ev {
580     ev(ld sw, int ix)
15b         : type(0), sw(sw), ix(ix) {}
229     ev(ld sw, arc* c) : type(1), sw(sw),
b43         prv(c->prv->ix), cur(c), nxt(c->nxt->ix) {}
3ba     int type, ix, prv, nxt; arc* cur; ld sw;
7de     bool operator<(const ev& l) const { return sw > l.sw; }
a05 };

463 struct voronoi_t {
c2e     vector<pt> v; vector<vector<pt>> poly;
19c     vector<i> f; vector<i> dy;
f1b };

c3a voronoi_t fortune(vector<pt> &ps) {
901     voronoi_t ans; beach line;
c75     priority_queue<ev, vector<ev>, greater<ev>> e;

448     auto add_edge = [&](int u, int v, int a, int b,

```

```

b32         arc* c1, arc* c2) {
923         if (c1) { c1->end = ans.dy.size()*2; }
7f2         if (c2) { c2->end = ans.dy.size()*2 + 1; }
a1e         ans.dy.emplace_back(u, v);
1ce         ans.f.emplace_back(a, b);
fb8     };
226     auto write_edge = [&](int ix, int v) {
bda         if (ix % 2 == 0) { ans.dy[ix/2].first = v; }
aaa         else { ans.dy[ix/2].second = v; }
cae     };
c8f     vector<pair<pt, int>> psi(n);
259     if (line.get(cur, nxt)) { e.push({nxt, cur}); }
f9d };

3f4     int n = ps.size(), cnt = 0;
a57     vector<pair<pt, int>> psi(n);
830     for (int i = 0; i < n; i++)
cb0         psi[i] = make_pair(ps[i], i);

ca0     vector<arc> nodes;
85c     nodes.reserve(n*4);
bca     auto mknode = [&](pt p, int ix) {
a9d         nodes.emplace_back(p, ix);
bfd         return &nodes.back();
6dd     };
11c     sort(psi.begin(), psi.end(), [](auto& a, auto& b) {
4c9         return make_pair(a.first.py, a.first.px)
164             < make_pair(b.first.py, b.first.px);
c0c     });
f42     vector<int> bix(n);
620     for (int i = 0; i < n; i++) { bix[i] = psi[i].second; }

9fa     arc* prv = line.root = mknode(psi[0].first, 0), *no;
6f5     for (int i = 1; i < n; i++) {
0e7         if (sgn(psi[i].first.py - psi[0].first.py) == 0) {
e18             add_edge(-1, -1, i-1, i, 0, prv);
9c9             line.insert(no = mknode(psi[i].first, i), prv, 1);
914             prv = no;
586             else { e.emplace(psi[i].first.py, i); }
9d9         }
4c1     while (e.size()) {
3b1         ev q = e.top(); e.pop();
5eb         arc *prv, *cur, *nxt, *si;
61a         int v = ans.v.size(), ix = q.ix;
292         line.sw = q.sw;
4c5         if (q.type == 0) {
dbe             pt p = psi[ix].first;
e4f             cur = line.find_beachline(p.px);
357             line.insert(si = mknode(p, ix), cur, 0);
7a0             line.insert(prv = mknode(cur->p, cur->ix), si, 0);
859             add_edge(-1, -1, cur->ix, ix, si, prv);
a8b             add_event(prv); add_event(cur);
5e2             continue;
966         }

802         cur = q.cur, prv = cur->prv, nxt = cur->nxt;
c47         if (!prv || !nxt || prv->ix != q.prv ||
05f             nxt->ix != q.nxt) { continue; }
cac         ans.v.push_back(circumcenter(prv->p, nxt->p, cur->p));
101         write_edge(prv->end, v); write_edge(cur->end, v);
20f         add_edge(v, -1, prv->ix, nxt->ix, 0, prv);
a9b         line.erase(cur);
01f         add_event(prv); add_event(nxt);
cbb     }

355     ans.poly.assign(n, {});
29b     for (int i = 0; i < ans.dy.size(); i++) {
aad         auto [x, y] = ans.dy[i];
535         if (x == -1 || y == -1) { continue; }
6d6         ans.poly[bix[ans.f[i].first]].push_back(ans.v[x]);
a72         ans.poly[bix[ans.f[i].second]].push_back(ans.v[x]);
efa         ans.poly[bix[ans.f[i].first]].push_back(ans.v[y]);
3b4         ans.poly[bix[ans.f[i].second]].push_back(ans.v[y]);
329     }
38e     for (int i = 0; i < ans.poly.size(); i++)
1f8         ans.poly[i] = convex_hull(ans.poly[i]);
ba7     return ans;
715 }

```

7.14 Varredura angular

```
1aa int get_points_inside(int i, double r, int n) {
19b   vector<double> ang;
f90   for (int j = 0; j < n; j++)
ecd     if (i != j && d[i][j] ≤ 2*r) {
5bc       double a = atan2(p[j].py-p[i].py, p[j].px-p[i].px);
ee7       double b = acos(d[i][j]/(2*r));
18f       ang.push_back(dbi(a - b, j));
0c1       ang.push_back(dbi(a + b, j));
782     }
f24   sort(ang);
0f5   int count = 1, res = 1;
1ed   for (auto angle : ang) {
9eb     count += angle.second ? 1 : -1;
130     res = max(res, count);
864   }
b50   return res;
777 }
```

7.15 Interseção de retângulos

```
6e1 bool has_intersection(pt tl0, pt br0, pt tl1, pt br1) {
855   pt tl = pt(max(tl0.px, tl1.px), max(tl0.py, tl1.py));
13e   pt br = pt(min(br0.px, br1.px), min(br0.py, br1.py));
5c3   return make_pair(tl.px, tl.py)
aa1     ≤ make_pair(br.px, br.py);
af7 }
```

7.16 Segmento e segmento

```
75b bool intersects_seg(seg s, seg t) {
993   int o1 = ord_ornt(s, t.aa), o2 = ord_ornt(s, t.bb);
6be   int o3 = ord_ornt(t, s.aa), o4 = ord_ornt(t, s.bb);
11f   return (
892     (o1 ≠ o2 && o3 ≠ o4) ||
f03     (o1 = 0 && on_segment(s, t.aa)) ||
961     (o2 = 0 && on_segment(s, t.bb)) ||
30f     (o3 = 0 && on_segment(t, s.aa)) ||
c80     (o4 = 0 && on_segment(t, s.bb));
d9e }
```

```
44a double distance_seg_seg(seg s, seg t) {
35e   if (intersects_seg(s, t)) { return 0; }
595   double v1 = distance_seg_point(s.aa, t);
e14   double v2 = distance_seg_point(s.bb, t);
a75   double v3 = distance_seg_point(t.aa, s);
662   double v4 = distance_seg_point(t.bb, s);
545   return min({ v1, v2, v3, v4 });
52d }
```

8 Algoritmos

8.1 Maior subsequência crescente rápido

```
b01 int lis(vector<int>& v) {
3d2   int n = v.size();
dab   vector<int> ans;
e81   ans.push_back(v[0]);
d44   for (int i = 1; i < n; i++) if (v[i] > ans.back()) {
a9b     ans.push_back(v[i]);
9d9   } else {
8b9     ans[lower_bound(all(ans), v[i]) - begin(ans)] = v[i];
cdd   }
1eb   return ans.size();
290 }
```

8.2 Subconjuntos de tamanho K

```
ae5 bool next_combination(vector<int>& a, int n) {
dfb   int k = a.size();
f2f   for (int i = k-1; i ≥ 0; i--) if (a[i] ≤ n-k+i) {
9a9     a[i]++;
48b     for (int j = i+1; j < k; j++)
cc6       a[j] = a[j-1] + 1;
6a5     return 1;
35a   }
```

```
bb3   return 0;
fdd }
```

8.3 Soma sobre subconjuntos

$$f[m] = \sum_{i \subseteq m} a[i]$$

Para mudar para superconjunto, use `~mask` no if.

```
e03 vector<ll> sos_dp(vector<ll> f) {
07b   int n = __builtin_ctz(f.size());
cef   assert((1<<n) == f.size());
830   for (int i = 0; i < n; i++)
547     for (int mask = 0; mask < (1<<n); mask++)
796       if (mask>>i&1) f[mask] += f[mask^(1<<i)];
abe   return f;
efd }
```

8.4 Ordenação por fusão

```
54a int swaps = 0;
75c void merge_sort(int l, int r) {
8b4   if (r - l == 1) { return; }
a81   int mi = l + (r - l) / 2;
b54   merge_sort(l, mi); merge_sort(mi, r);
9c5   vector<int> aux(r - l);
60c   int i = l, j = mi;
d75   for (int k = 0; k < r - l; k++) {
eda     if (i < mi && j < r) {
9a6       if (!a[i] < a[j]) { swaps += mi - i; }
7ed       if (a[i] < a[j]) { aux[k] = a[i++]; }
8bb       else { aux[k] = a[j++]; }
420     } else if (i < mi) { aux[k] = a[i++]; }
8bb     else { aux[k] = a[j++]; }
ef3   }
152   copy(all(aux), begin(a)+l);
a25 }
```

8.5 Algoritmo de Mo

```
ab8 vector<int> mo(vector<qry> qs) {
d5c   vector<int> ans(qs.size());
a0f   sort(all(qs), [](qry a, qry b) {
04d     if (a.l/B ≠ b.l/B) { return a.l < b.l; }
d34     return bool(((a.l/B) % 2) ^ (a.r < b.r));
c0c   });
3d9   int l = 0, r = -1;
3f3   for (qry q : qs) {
619     while (l > q.l) { l--; ins(l, 'l'); }
515     while (r < q.r) { r++; ins(r, 'r'); }
3be     while (l < q.l) { rem(l, 'l'); l++; }
945     while (r > q.r) { rem(r, 'r'); r--; }
ccd     ans[q.ix] = get_ans();
c81   }
ba7   return ans;
90e }
```

8.6 Meet in the middle

```
adf int mitm(vector<int>& v, int x) {
3d2   int n = v.size();
d40   map<ll, int> sums; ll c = 0;
9c4   for (int ss = 0; ss < 1<<(n/2); ss++) {
058     ll s = 0;
9b8     for (int i = 0; i < n/2; i++) if (1<<i & ss)
df1       s += v[i];
391     if (s ≤ x) { sums[s]++; }
705   }
3bb   for (int ss = 0; ss < 1<<(n/2+n%2); ss++) {
058     ll s = 0;
4a9     for (int i = n/2; i < n; i++) if (1<<(i-n/2) & ss)
df1       s += v[i];
af0     if (sums.count(x - s)) { c += sums[x - s]; }
415   }
807   return c;
7ef }
```

8.7 Agendamento ótimo de tarefas

```
888 struct job {
a7b     int dl, duration, idx;
1e2     bool operator<(job o) const { return dl < o.dl; }
5dc };

b83 vector<int> compute_schedule(vector<job> jobs) {
df0     sort(all(jobs));
921     set<pair<int,int>> s; vector<int> schedule;
8fa     for (int i = jobs.size()-1; i ≥ 0; i--) {
0f9         int t = jobs[i].dl - (i ? jobs[i-1].dl : 0);
cac         s.insert(make_pair(jobs[i].duration, jobs[i].idx));
4f8         while (t && !s.empty()) {
cb7             auto it = begin(s);
14c             if (it->first ≤ t) {
f63                 t -= it->first;
383                 schedule.push_back(it->second);
9d9             } else {
403                 s.insert(make_pair(it->first - t, it->second));
a34                 t = 0;
4b2             }
df3             s.erase(it);
635         }
7b5     }
e9f     return schedule;
20c }
```

8.8 Maior subsequência comum

```
b42 int lcs(string s, string t) {
c61     int m = s.size(), n = t.size();
5c9     vector<vector<int>> l (m+1, vector<int>(n+1));
cf2     for (int i = 0; i ≤ m; i++)
fe8         for (int j = 0; j ≤ n; j++) {
dc2             if (i = 0 || j = 0)
534                 l[i][j] = 0;
28a             else if (s[i-1] == t[j-1])
b5c                 l[i][j] = l[i-1][j-1] + 1;
295             else
d0c                 l[i][j] = max(l[i-1][j], l[i][j-1]);
b67         }
1dd     return l[m][n];
4f5 }
```

8.9 PD de perfil quebrado

```
91b vector<vector<ll>> dp (N+1, vector<ll>(1LL<<M));

e40 void calc(int x=0, int y=0, int s=0, int nxts=0) {
a4b     if (x == n) { return; }
dc2     if (y ≥ m) { dp[x+1][nxts] += dp[x][s]; return; }
715     int cs = 1 << y;
b04     if (s & cs) { calc(x, y+1, s, nxts); }
4e6     else {
c49         calc(x, y+1, s, nxts | cs);
688         if (y+1 < m && !(s & cs) && !(s & (cs << 1)))
a94             calc(x, y+2, s, nxts);
d3d     }
9da }
```

```
f5c int solve() {
30e     dp[0][0] = 1;
18b     for (int x = 0; x < n; x++)
2dd         for (int s = 0; s < (1<<m); s++)
6f5             calc(x, 0, s, 0);
531     return dp[n][0];
7d7 }
```

8.10 Submáscaras

```
8ea for (int s = 0; s < (1LL<<n); s++)
860 for (int ss = s; ss > 0; ss = (ss-1)&s)
41a     c[i][s] = min(c[i][s], c[i][ss] + c[i][s-ss]);
```

9 Problemas

9.1 Frequência de frequências/moda com Mo

```
54b vector<int> v (N), freqfreq (N), freq (N);
9b7 vector<ll> sumfreq (N);
1a4 int ans = 0;
890 void update_freq(int x, int d) {
d75     freqfreq[freq[x]]--;
2a4     if (freq[x]) { sumfreq[freq[x]] -= x; }
e2e     freq[x] += d;
6c4     if (freq[x]) { sumfreq[freq[x]] += x; }
dfa     freqfreq[freq[x]]++;
69c     if (freqfreq[ans+1]) { ans++; }
21e     if (ans > 0 && freqfreq[ans] == 0) { ans--; }
608 }
5ce void ins(int i, char dir) { update_freq(v[i], +1); }
248 void rem(int i, char dir) { update_freq(v[i], -1); }
8e9 int get_ans() { return ans; }
```

9.2 Rainhas no tabuleiro

```
675 int n, s;
6f6 void queens(int i, ll rw = 0, ll ld = 0, ll rd = 0) {
754     if (rw == (1<<n)-1) { s++; return; }
63e     ll pos = ((1<<n)-1) & ~(rw | ld | rd);
94f     while (pos) {
db7         ll p = pos & -pos; pos -= p;
27d         queens(i+1, rw | p, (ld | p) << 1, (rd | p) >> 1);
38c     }
2b2 }
```

9.3 Remoção de valores com deque

```
15c vector<deque<int>> l (N/B + 1);
068 vector<int> remove(vector<int>& v, vector<int>& r) {
dab     vector<int> ans;
3d2     int n = v.size();
830     for (int i = 0; i < n; i++)
d13         l[i/B].push_back(v[i]);
13a     for (int i : r) {
c65         auto it = begin(l[i/B])+(i%B);
0c0         ans.push_back(*it); l[i/B].erase(it);
338         int bi = i/B;
432         while (l[bi+1].size() > 0) {
873             l[bi+1].push_back(l[bi].back());
844             l[bi].pop_back();
a21             bi++;
e97         }
d96     }
ba7     return ans;
432 }
```

9.4 Desembarcadouro

```
9b1 const int K = 300;
dfa vector<int> c (1e5+15);
ae7 vector<vector<int>> dp (1e5+15 + K, vector<int>(K));
e8d int main() {
c5f     int n, q; cin >> n >> q;
1f4     while (q--) {
9a3         int a, l, d; cin >> a >> l >> d;
b2f         if (d ≥ K)
c7b             for (int i = 0; i < l; i++) { c[a + i * d]++; }
309         else { dp[a][d]++; dp[a+(l*d)][d]--; }
99b     }
e34     for (int j = 1; j < K; j++)
a68         for (int i = j; i < n + 1; i++)
e98             dp[i][j] += dp[i - j][j];
78a     for (int i = 1; i ≤ n; i++) {
e34         for (int j = 1; j < K; j++)
b54             c[i] += dp[i][j];
12b         cout << c[i] << ' ';
511     }
019 }
```


9.5 Inversões usando algoritmo de Mo

```
04b ll ans = 0;
3a1 void ins(int i, char dir) {
a52   if (dir == 'l') { ans += query(v[i]-1); }
650   else { ans += query(N) - query(v[i]); }
714   add(v[i], +1);
65f }
c83 void rem(int i, char dir) {
f52   if (dir == 'l') { ans -= query(v[i]-1); }
ccd   else { ans -= query(N) - query(v[i]); }
aad   add(v[i], -1);
fb7 }
64e ll get_ans() { return ans; }
```

9.6 Média e mediana

```
20e double solve(double x, vector<double>& v) {
3d2   int n = v.size();
d87   vector<double> mem (n);
45b   for (int i = n-1; i ≥ 0; i--) {
eea     double ans = max(mem[i+1], mem[i+2]);
420     mem[i] = v[i] + ans;
cec   }
de3   return max(mem[0], mem[1]);
8f5 }
```

```
e04 void get_mean_med(vector<int> a) {
798   double mean; {
e3c     double lo = 0, hi = 1e10;
4f3     for (int it = 0; it < 100; it++) {
331       double mi = (lo+hi)/2;
ac1       vector<double> b (n);
830       for (int i = 0; i < n; i++)
c4d         b[i] = a[i] - mi;
f83       if (solve(mi, b) ≥ 0) { lo = mi; }
450       else { hi = mi; }
d2d     }
411     mean = lo;
752   }
21d   double med; {
e3c     double lo = 0, hi = 1e10;
4f3     for (int it = 0; it < 100; it++) {
331       double mi = (lo+hi)/2;
ac1       vector<double> b (n);
830       for (int i = 0; i < n; i++)
21e         b[i] = (a[i] ≥ mi) ? +1 : -1;
a21       if (solve(mi, b) > 0) { lo = mi; }
450       else { hi = mi; }
64a     }
131     med = lo;
3b8   }
1f2   return make_pair(mean, med);
663 }
```

9.7 Distância de edição

```
ba3 void edit_distance(string s1, string s2) {
f22   int n1 = s1.size(), n2 = s2.size();
1ee   vector<vector<int>> dist (n1+1, vector<int>(n2+1));
883   vector<vector<int>> op (n1+1, vector<int>(n2+1));
07f   for (int i = 1; i ≤ n1; i++) {
614     dist[i][0] = i; op[i][0] = 3;
1cf   }
8de   for (int j = 1; j ≤ n2; j++) {
972     dist[0][j] = j; op[0][j] = 2;
79e   }
063   for (int i = 1; i ≤ n1; i++)
8de     for (int j = 1; j ≤ n2; j++) {
7bd       int bdist, bop;
aea       if (s1[i-1] == s2[j-1]) {
ba5         bdist = dist[i-1][j-1]; bop = 0;
9d9       } else {
6cb         int repl = dist[i-1][j-1] + 1;
456         bdist = repl; bop = 1;
d7d         int ins = dist[i][j-1] + 1;
e4f         if (ins < bdist) { bdist = ins; bop = 2; }
102         int del = dist[i-1][j] + 1;
c62         if (del < bdist) { bdist = del; bop = 3; }
```

```
63c       }
ba5       dist[i][j] = bdist;
266       op[i][j] = bop;
e97     }
761   return dist[n1][n2];
6a9 }
```

9.8 Código de Gray

```
df6 vector<int> gray_code(int n) {
4c8   vector<int> ans (1<<n);
2f0   for (int i = 0; i < (1<<n); i++)
01c     ans[i] = i^(i>>1);
ba7   return ans;
748 }
```

9.9 Árvore de segmentos de GCD

```
e8d int main() {
c5f   int n, q; cin >> n >> q;
22b   gcdst_t gcdst; sumst_t sumst;
4f7   vector<ll> a (n+1), b (n+1);
78a   for (int i = 1; i ≤ n; i++) {
ee7     cin >> a[i]; b[i] = a[i] - a[i-1];
d01   }
bdb   gcdst.build(b, n); sumst.build(b, n);

1f4   while (q--) {
431     int t; cin >> t; int l, r; cin >> l >> r;
710     if (t == 1) {
e38       ll v; cin >> v;
651       sumst.add_value(l, +v, n);
f74       gcdst.add_value(l, +v, n);
651       if (r+1 ≤ n) {
18d         sumst.add_value(r+1, -v, n);
dcc         gcdst.add_value(r+1, -v, n);
7aa       }
9d9     } else {
// gcd(al, al + 1, ..., ar)
// gcd(al, al + 1 - al, al + 2 - al + 1, ..., ar - ar - 1)
4d8     ll first = sumst.op_inclusive(1, l, n);
025     ll ans = gcdst.op_inclusive(l+1, r, n);
a21     cout << gcd(first, ans) << "\n";
616   }
c84 }
7df }
```

9.10 Distância em árvore

```
9f6 vector<int> vis (N), dist (N), distix (N), dist2 (N);
414 int cts = 1;
```

```
921 void insert_max_dist(int u, int val, int ix) {
9aa   if (val > dist[u]) {
9c5     dist2[u] = dist[u]; dist[u] = val; distix[u] = ix;
9d9   } else {
394     dist2[u] = max(dist2[u], val);
9a0   }
e2d }
```

```
816 void dfs_down(int u) {
ae6   vis[u] = cts;
5b3   for (int v : g[u]) if (vis[v] ≠ cts) {
6fc     dfs_down(v); insert_max_dist(u, 1 + dist[v], v);
925   }
16c }
```

```
c94 void dfs_up(int u) {
ae6   vis[u] = cts;
5b3   for (int v : g[u]) if (vis[v] ≠ cts) {
97f     int d = (distix[u] ≠ v ? dist[u] : dist2[u]);
a4a     insert_max_dist(v, 1 + d, u);
fb9     dfs_up(v);
f54   }
e41 }
```

```
4b2 void tree_distance() {
c75   dfs_down(0); cts++; dfs_up(0);
0f6 }
```