目次

1 序論

1.1 概要

AtCoder などで競技プログラミングをしているので、課題ついでに競プロ用のライブラリを整備します。競技プログラミングでは速度が問われ、一からアルゴリズムを実装していると間に合わないことがあります。一方で、ライブラリを貼るだけで通る問題はほとんどなく、それを理解して応用できる力がある必要があり、コードを理解しつつコンテスト中に競プロの本質的な部分に集中できるようライブラリを整備していこうと思います。

1.2 環境

実際に AtCoder でサポートされている言語バージョンに準じて作っていきます。Python と PyPy がサポートされています。

```
# On AtCoder's Custom Code Test
import sys
print(sys.version)

import pkg_resources
for e, dist in enumerate(pkg_resources.working_set):
    print(dist.project_name, dist.version, end = ", " + "\n" * (e%8 == 7))

>> Python
3.8.2 (default, Feb 26 2020, 02:56:10)
[GCC 7.4.0]
..., scipy 1.4.1, ..., numpy 1.18.2, numba 0.48.0, networkx 2.4, ...

>> PyPy
3.6.9 (7.3.0+dfsg-1~ppa1~ubuntu18.04, Dec 24 2019, 08:12:19)
[PyPy 7.3.0 with GCC 7.4.0] ...
```

簡単に言えば、Python は ver.3.8, PyPy は Python ver.3.6 のものが埋め込まれており、PyPy は定数倍高速化が超強力な代わりに numpy, scipy, numba, networkx などのサードパーティーライブラリは使用できない。

```
!apt-get install pypy
!apt install texlive-fonts-recommended texlive-fonts-extra cm-super dvipng
```

で Google Colab に PyPy, LaTeX 2_εをインストール。

1.3 その他

基本的に Google Colab のコードと同じものを載せていますが、都合上一部省略あるいは増強したものがあります。またより厳密で詳細な証明を書いていることもあります。

2　基礎的で非アルゴリズム的なもの

2.1　入力高速化

```
1 import sys
2 input = sys.stdin.readline
```

入力が 10^5 行を超える場合、`sys.stdin.readline` に置換することで大きく高速化できる。ただし `open(0)` の方が高速化されることも多い。

2.2　再帰制限の変更

```
1 import sys
2 sys.setrecursionlimit(50000)
```

Python では再帰上限が 1000 回に設定されており、これを超えると `RecursionError: maximum recursion depth exceeded in comparison` を出す。`sys.setrecursionlimit(n)` で上限を n 回に設定しこれを回避できる（ただし、メモリに負荷がかかり、ある程度超えると Python がクラッシュすることには変わりないので、乱用に注意）。

2.3 タイマーのセット

```python
def timer(func, v, *args, **kwargs):
    import time
    if __name__ == '__main__':
        start = time.time()
        P = func(*args, **kwargs)
        t = time.time() - start
        print(f"time: {t :.3e}[sec]")
        return (t, P) if v else t
```

時間計測用によく使うので、time モジュールを使ったタイマーも用意しておく。関数とその引数を渡して使う。

```python
import time
start = time.time()
while time.time() - start < time_limit:
    do()
```

マラソンコンテストではこういう形も使うかも。

3　整数論

3.1　素因数分解

定義 1. 自然数 $n \geq 2$ について **素因数分解** n_P を $e_p > 0$ となるように

$$n_P \coloneqq n = \prod_{p \in \mathbb{P}} p^{e_p} \tag{1}$$

と定める。n_P における p の集合を $\mathbb{P}_n$, $e_p > 0$ を $e(p,n)$ などと書く。ただし $\mathbb{P}_1 = \emptyset$ とする。

定理 2. 初等数論の基本定理　2 以上の全ての自然数は素因数分解できその表現は一意に定まる。

Proof. (証明) **素因数分解の可能性**：素数は素因数分解できるとしてよい。素因数分解できない合成数が存在すると仮定し、その中で最小のものを m とおく。すると $m = ab$ となるような自然数の組 $(a,b) \in (1,m)^2$ が存在する。m の最小性および素数の素因数分解可能性から a,b は素因数分解できるが、その積 m も素因数分解できることになり矛盾する。よって背理法より示される。

素因数分解の一意性：素数は一意に素因数分解できるとしてよい。複数通りの素因数分解をもつ合成数が存在すると仮定し、その中で最小のものを n とおく。そのうち 2 つ n_{P1}, n_{P2} について考え、p,q を n_{P1}, n_{P2} それぞれに現れる最小の素数とする。p が n_{P2} に現れるとすると、p を n_{P1}, n_{P2} それぞれについて $e(p,n)$ の値を 1 下げる（$e(p,n) = 1$ のとき p を除く）ことで n/p の素因数分解が出現しこれは n の最小性に矛盾するから、p は n_{P2} に現れず、特に $p \neq q$ である。さらに、n は合成数なので p,q の定義から $p^2, q^2 \leq n$ であり、$p \neq q$ より $pq < n$ である。$0 < n - pq < n$ より $n - pq$ の素因数分解は一意に定まり、p,q は共にこれを割り切るので $pq \mid n - pq \therefore pq \mid n \therefore p \mid n/q$ となるが、$n/q < n$ の素因数分解は一意なのでこれに p は現れ、すると n_{P2} にも現れることになる。これは先ほどの議論に矛盾するので背理法より示された。

□

アルゴリズム 3.　試し割り法による素因数分解

```
def factorize(n) -> list:
    b, e = 2, 0
    fct = []
    while b * b <= n:
        while n % b == 0:
            n //= b
            e += 1
        if e:
            fct += (b, e),
        b += 1
        e = 0
    if n > 1:
        fct += (n, 1),
    return fct
# P_n, E_n = zip(*factorize(n))
```

n に対する素因数分解は、2 以上 $\sqrt{n}$ 以下の自然数についてそれぞれ割れるかどうかを試していけばよい。すなわち計算量は $O(\sqrt{n})$ 。証明は以下。さらに高速な素因数分解は後に示す。

補題 4. $\lfloor\sqrt{n}\rfloor$ までの素因数を全て用いて素因数分解を走査した時、残っているのは素数である。すなわち、

$$\#\left\{x \mid x \in \mathbb{P}_n, x > \lfloor\sqrt{n}\rfloor\right\} \leq 1 \tag{2}$$

であって、この要素数が 1 であるときその要素を y とすると、

$$e(y, n) = 1 \tag{3}$$

Proof. (証明) 命題の否定として残った数が合成数であると仮定する。すると $p \in \mathbb{P}, z \in \mathbb{Z}$ の積 pz で表すことができる。これらは未被走査なので $p, z > \sqrt{n}$ だが、すると $pz > \sqrt{n}^2 = n$ となり矛盾。よって背理法によって示された。 □

系 5. 試し割り法による素因数分解は正当性を持ち計算量は $O(\sqrt{n})$ である。

Proof. (証明) 補題 4 より alg.3 における走査は $\lfloor\sqrt{n}\rfloor$ までで十分である。また、小さいものから順に見ていけば、合成数については自身より小さい（すでに走査された）素数の積で表されるのですでに割り切れなくなっており通過する。よって示された。 □

3.2 最大公約数・最小公倍数

定義 6. 自然数の組 $(a,b) \in \mathbb{N}^2$ に対して、これらを共に割り切る自然数を公約数 *(Common Divisor)* とよび、その全体集合を $\mathrm{cd}(a,b)$ などとかく。公約数のうち最も大きいもの $\max \mathrm{cd}(a,b)$ を **最大公約数** *(Greatest Common Divisor)* とよび、$\gcd(a,b)$ などとかく。また、いずれかが 0 でない一般の整数の組 $(a,b) \in \mathbb{Z}^2$ に対して、$a = 0$ のとき $\gcd(a,b) = b$ 、$b = 0$ のとき $\gcd(a,b) = a$ とし、$\gcd(a,b) = \gcd(|a|,|b|)$ とする。さらにこれは少なくとも 1 つが 0 でない 3 つ以上の整数に対して拡張できる。

```
from math import*
gcd(6, 9), gcd(5, 0), gcd(-6, -9)
>> 3, 5, 3
```

補題 7. 以下 $(a,b) \in \mathbb{Z}^2$ $(b \neq 0)$ に対して $q = a//b, r = a\%b$ とする。ここで、

$$\gcd(a,b) = \gcd(b,r) \tag{4}$$

Proof. (証明) $r = a - bq$ より、

$$m \mid a \wedge m \mid b \Longrightarrow r \mid m \tag{5}$$

したがって、

$$\mathrm{cd}(a,b) \subseteq \mathrm{cd}(b,q) \tag{6}$$

すなわち

$$\gcd(a,b) \leq \gcd(b,q) \tag{7}$$

また $a = bq + r$ より、

$$b \mid m \wedge r \mid m \Longrightarrow a \mid m \tag{8}$$

したがって、

$$\mathrm{cd}(a,b) \supseteq \mathrm{cd}(b,q) \tag{9}$$

すなわち

$$\gcd(a,b) \geq \gcd(b,q) \tag{10}$$

よって (7), (10) より示された。 □

アルゴリズム 8. ユークリッドの互除法 *(Euclidean algorithm)*

```
def gcd(a, b):
    if a == 0: return b
    return gcd(b%a, a)
```

以上のアルゴリズムで最大公約数を算出できる。計算量は $O(\log n)$. 証明は以下。

定義 9. ユークリッドの互除法において i 回目の操作における商、余りをそれぞれ q_i, r_i とおく。

ただし $r_{-1} = a, r_0 = b$ とする。

補題 10. r_i は狭義単調減少列である。

Proof. (証明) $i = 0, 1, 2 \ldots$ に対して

$$r_i = r_{i+1}q_{i+2} + r_{i+2}, 0 \leq r_{i+2} < r_{i+1} \tag{11}$$

よって $r_{i+1} > r_{i+2}$ が示された。 □

系 11. ユークリッドの互除法の正当性 ユークリッドの互除法は停止し正しい答えを返す。

Proof. (証明) 操作の繰り返しは補題 *7* より正しい。停止性を示す。*10* と $i > 0$ において $r_i \geq 0$ より r_i は有限回で 0 に到達する。そのような i を $n + 1$ とすると、結局 $\gcd(a, b) = \ldots = \gcd(r_n, r_{n+1}) = r_n$ と実際の数で出力される。よって示される。 □

補題 12. ユークリッドの互除法の計算量 ユークリッドの互除法の計算量は $O(\log n)$.

Proof. (証明) $a > b$ のときは $b\%a = b$ となり、$\gcd(b, a)$ が呼ばれるので $a \leq b$ の場合に帰着する。よって $a \leq b$ と仮定としてもそれ以外の場合操作回数は 1 回しか変わらず一般性は失われない。$q_1 \geq 1, a \geq r_1$ から $b \geq a + r_1 \geq 2r_1$ すなわち $r_1 \leq b/2$. よって 2 回操作を繰り返せば a と b は半分以下になる。よって計算回数はざっと $2 \log n$ 程度なので計算量は $O(\log n)$. □

定義 13. 自然数の組 $(a, b) \in \mathbb{N}^2$ に対して、これらを共に割り切る自然数を公倍数 *(Common Multipie)* 、公約数のうち最も大きいものを **最小公倍数** *(Least Common Mutipie)* とよび、$\mathrm{lcm}(a, b)$ などとかく。また、いずれかが 0 でない一般の整数の組 $(a, b) \in \mathbb{Z}^2$ に対して、$a = 0$ または $b = 0$ のとき $\mathrm{lcm}(a, b) = 0$ とし、$\mathrm{lcm}(a, b) = \mathrm{lcm}(|a|, |b|)$ とする。さらにこれは少なくとも 1 つが 0 でない 3 つ以上の整数に対して拡張できる。

```python
def lcm(a, b):
 return abs(a * b) // gcd(a, b)
```

定義 14. 自然数の組 $(a_1, a_2, \dots a_n)$ に対して $e_p \geq 0$ となるように n の拡張された素因数分解を

$$n = \prod_{p \in \mathbb{P}(a_1, \dots a_n)} p^{e_p} \tag{12}$$

と定める。ただし

$$\mathbb{P}(a_1, \dots, a_n) \coloneqq \bigcap_{i=1}^{n} \mathbb{P}_{a_i} \tag{13}$$

e_p を $e_0(p, n)$ などとかく。

系 15. いずれかが 0 でない一般の整数の組 $(a, b) \in \mathbb{Z}^2$ に対して、以下が成り立つ。

$$|ab| = \gcd(a, b)\mathrm{lcm}(a, b) \tag{14}$$

Proof. (証明) $|a|, |b|$ に対する拡張された素因数分解を考えると、定義から明らかに

$$\mathbb{P}_{\gcd(a,b)}, \mathbb{P}_{\mathrm{lcm}(a,b)} \subseteq \mathbb{P}(|a|, |b|) \tag{15}$$

となり $|a|, |b|, \gcd(a, b), \mathrm{lcm}(a, b)$ に対して自然に拡張できる。すなわち、

$$\begin{aligned} \forall p \in \mathbb{P}(|a|, |b|), \quad & e_0(p, \gcd(a, b)) + e_0(p, \mathrm{lcm}(a, b)) && (16)\\ &= \max(e_0(p, |a|), e_0(p, |b|)) + \min(e_0(p, |a|), e_0(p, |b|)) && (17)\\ &= e_0(p, |a|) + e_0(p, |b|) && (18) \end{aligned}$$

よって $|a||b| = \gcd(a, b)\mathrm{lcm}(a, b)$ となり示された。 □

```
from functools import*
reduce(gcd, (3, 6, 12, 240)), reduce(lcm, (3, 6, 12, 240))
>> 3, 240
```

系 16. $\gcd(a,b)$ を $\langle a,b\rangle$ などとかく。自然数の組 $(a_1,a_2,\dots a_n)\in\mathbb{N}^n$ に対して

$$\langle a_1,a_2,\dots a_n\rangle=\langle\dots\langle\langle a_1,a_2\rangle,a_3\rangle,\dots\rangle \tag{19}$$

lcm についても同様の命題が成り立つ。

アルゴリズム 17. 拡張ユークリッドの互除法 *(Extended Euclidean algorithm)*

```
def egcd(a, b):
    if a == 0:
        return b, 0, 1
    g, y, x = egcd(b % a, a)
    return g, x - (b // a) * y, y
```

上記のアルゴリズムは $ax+by=\gcd(a,b)$ の最小解 $(x,y)=(p,q)$ に対して (g,p,q) を返す。

補題 18. 拡張ユークリッドの互除法の正当性

拡張ユークリッドの互除法は停止し正しい解を返す。すなわち、

$$x\coloneqq\left\{\begin{matrix}x_{-1}=1,x_0=0\\ x_i=q_ix_{i-1}+x_{i-2}\ (i\geq 1)\end{matrix}\right\},\ y\coloneqq\left\{\begin{matrix}y_{-1}=0,y_0=y\\ y_i=q_iy_{i-1}+y_{i-2}\ (i\geq 1)\end{matrix}\right\} \tag{20}$$

とするとき、$i=-1,0,1,\dots,n$ に対して

$$ax_i-by_i=(-1)^{i-1}r_i. \tag{21}$$

Proof. (証明) 数学的帰納法による。以下 $i-1$ まで仮定

まず $r_i = r_{i+1}q_{i+2} + r_{i+2}$ より

$$r_{i+2} = r_i - r_{i+1}q_{i+2} \tag{22}$$

$$\therefore r_i = r_{i-2} - r_{i-1}q_i \tag{23}$$

よって

$$ax_i - by_i = a(q_i x_{i-1} + x_{i-2}) - b(q_i y_{i-1} + y_{i-2}) \tag{24}$$

$$= q_i(ax_{i-1} - by_{i-1}) + ax_{i-2} - by_{i-2} \tag{25}$$

$$= (-1)^{i-2}q_i r_{i-1} + (-1)^{i-3}r_{i-2} \tag{26}$$

$$= (-1)^i(q_i r_{i-1} - r_{i-2}) \tag{27}$$

$$= (-1)^i(-1)r_i \tag{28}$$

$$= (-1)^{i-1}r_i \tag{29}$$

□

3.3 剰余類環

```
1 MOD = 10**9+7
2 MOD2 = 998244353
3 INF = float("inf") # 10**9
```

const を定義しておく。競技プログラミングでは特殊な素数として MOD は 10^9+7 や 998244353 が使われることが多い。無限大は INF を使えば良いが float 型のため違う型同士での比較が重い PyPy で使う / 上限として前準備に使うなどするときは大きな整数にするべきかも。

定義 19. 関数 $f : A^n \longmapsto B$ を、集合 A に対して定義された ***n* 項演算** *(n-ary operation)* という。以下 2 項演算を扱う。ここで、$B \subseteq A$ のとき A は f について**閉じている** *(closed)* という。

例) $\mathbb{N}$ は加法について閉じているが、減法について閉じていない。また $\mathbb{Z}$ は加法、減法、乗法について閉じているが除法について閉じていない。

定義 20. 集合 A と A において定義された演算 $f_1, f_2, \ldots, f_k$ の組を **代数系** *(algebraic system)* といい $\langle A, f_1, f_2, \ldots, f_k \rangle$ などとかく。

定義 21. 集合 A で定義された演算 $*$ について、$\forall x \in A, e * x = x * e = x$ となるような $e \in A$ が存在するときこれを $*$ についての（両側）**単位元** *(identity element)* という。また、単位元 e を持つような代数系 $\langle A, * \rangle$ において、$x \in A$ に対して $xy = yx = e$ となるような y が存在するときこれを演算 $*$ に対する x の（両側）**逆元** *(inverse element)* とよび $y = x^{-1}$ などとかく。

定義 22. 代数系 $\langle S, * \rangle$ について、$*$ が結合的で閉じた演算であり、単位元が存在し S の各元が逆元をもつとき、これを群 *(group)* という。また、ある群 G の任意の 2 元に対し交換則が成立するとき、G をアーベル群 *(abelian group)* という。また、和に関してアーベル群である R において、積について結合律が成立しかつ積についての単位元 1 があり、乗算が加算に対して分配律がなりたつとき R は環 *(ring)* であるという。さらに R において積についての可換性が成立するとき R は可換環 *(commutative ring)* であるという。また、可換環 K において 0 以外の任意の元が積についての逆元を持つとき K を体 *(field)* という。さらに、群 G の部分集合 H が、G における演算 $*$ で群を作るとき $*$ について H は G の部分群 *(subgroup)* であるという。また、可換環をなす R について、R の加法に関して群である R の部分集合 I を考え、$\forall (r, a) \in RI, ra \in I$ のとき、I を R についての（両側）イデアル *(ideal)* であるという。

定義 23. 整数集合 $\mathbb{Z}$ の各要素について、自然数 $n \geq 2$ で割った剰余が等しい元をすべて集めたものを、n を法とする合同類あるいは剰余類という。合同式の性質から明らかなように、n を法として以下のように *well-defined* な加算と乗算を定義できる。

$$(a + n\mathbb{Z}) + (b + n\mathbb{Z}) \coloneqq (a + b) + n\mathbb{Z} \tag{30}$$

$$(a + n\mathbb{Z}) \cdot (b + n\mathbb{Z}) \coloneqq (a \cdot b) + n\mathbb{Z} \tag{31}$$

演算 $+$ が *well-defined* であるのは、$n\mathbb{Z}$ が $\mathbb{Z}$ の部分群であることにより、演算 $\cdot$ が *well-defined* なのは、$n\mathbb{Z}$ が $\mathbb{Z}$ の（両側）イデアルであることによる。すなわち、n を法とする合同は整数環 $\langle \mathbb{Z}, +, \cdot \rangle$ の構造と両立し、そのような剰余類は可換環をなす。これを n を法とする**剰余類環** *(class ring modulo n)* といい、$\mathbb{Z}/n\mathbb{Z}$ などとかく。

上の定義は、例えば競プロにおいて K で割ったあまりを求めるような問題で全ての加算や乗算をしていてはあまりに膨大な数になって処理しきれないときに、「取れるときに取っておく」で構わないことを示している。

例題 24. 数列 $7, 77, 777, \ldots$ で初めて K の倍数が現れるのは何項目か。ただし数列中に存在しないときは -1 と出力せよ。$1 \leq K \leq 10^6$　(出典：ABC174 C - Repsept)

解） K を法とする剰余類環で考える。これを表す頂点数 K の有向グラフを考えると遷移は 1 通りに定まるから、辺の数は高々 K 本である。よってそのような数が存在すると K 項目まで探索すれば必ず発見され、それ以外のとき存在しない。よって $O(N)$ でこの問題を解くことができた。

```python
k, R, a = int(input()), 7, -1
for i in range(1, k + 1):
    if not R%k:
        a = i
        break
    R = (R * 10 + 7) % k
print(a)
```

定義 25. $(a,b) \in \mathbb{Z}^2_{\neq 0},\ \gcd(a,b) = 1$ のとき a,b は互いに素であるといい、$a \perp b$ とかく。

補題 26. $a \perp b$ のとき $[0,b) \cap \mathbb{Z} = \mathbb{Z}/b\mathbb{Z} = (a\mathbb{Z})/b(a\mathbb{Z})$

Proof. (証明) 任意の $l \neq m$ なる $(l,m) \in \mathbb{Z}/p\mathbb{Z}^2$ について $la \equiv ma \mod b$ と仮定すれば、

$$|l-m|a \in b\mathbb{Z} \tag{32}$$

だが

$$|l-m| \in [1,b) \therefore |l-m| \notin b\mathbb{Z} \tag{33}$$

ここで $a \perp b$ より矛盾。背理法より

$$\forall (s,t) \in (b\mathbb{Z})^2, s \not\equiv t \mod b \text{ if } s \neq t \tag{34}$$

すなわち

$$|(a\mathbb{Z})/b(a\mathbb{Z})| = b-1 \wedge \forall e \in (a\mathbb{Z})/b(a\mathbb{Z}), e < b \tag{35}$$

よって示された（なお一番目の等号は定義そのものなので自明)。 □

補題 27. $\exists (x,y) \in \mathbb{Z}^2,\ ax + by = 1 \iff a \perp b$

Proof. (証明) $\Longrightarrow$) 対偶を示す。$\gcd(a,b) = d \geq 2$ とおくと $d \mid ax + by$ となり解を持たない。

$\Longleftarrow$) 補題 26 より

$$\exists m \in [1,b) \cap \mathbb{N},\ ma \equiv 1 \mod b \tag{36}$$

ここで ma を b で割った商を $n \in \mathbb{Z}$ とおいたとき

$$ma = bn + 1 \iff am - bn = 1 \tag{37}$$

となり $(m,-n)$ は整数解となっている。 □

定義 28. 整数の法 m における合同類環 $\mathbb{Z}/m\mathbb{Z}$ について、$a \in \mathbb{Z}$ に対する乗法逆元を **mod m における a についての乗法逆元** あるいは **モジュラ逆数** *(modular multiplicative inverse)* という。定義は明らかに $a^{-1} \equiv x$ 及び $ax \equiv 1$ と同値である。

```python
def modinv(a, m = MOD): # python 3.7 or later
    return pow(a, -1, m)
```

補題 29. $\mod m$ においての $a \in \mathbb{Z}$ の乗法逆元が存在することは $a \perp m$ と必要十分である。

Proof. (証明) 補題 *27,* 定義 *28* より

$$\exists (x, y) \in \mathbb{Z}^2,\ ax + my = 1 \Longleftrightarrow \exists z \in \mathbb{Z}, za \equiv 1 \mod m \tag{38}$$

を示せば良い。$\Longrightarrow$) $my \equiv 0 \mod m$ より $ax \equiv 1$.

$\Longleftarrow$) $az - 1 = qm$ となる整数 q が存在、よって $az + m(-q) = 1$. □

系 30. $p \in \mathbb{P}$ に対し $\mathbb{Z}/p\mathbb{Z}$ は体 $\mathbb{F}_p$ をなす。

Proof. (証明) p はそれ未満の自然数に対して互いに素だから補題 *29* より明らか。 □

定理 31. フェルマーの小定理 *(Fermat's little theorem)*

$$a \perp p, p \in \mathbb{P}, a \in \mathbb{Z} \Longrightarrow a^{p-1} \equiv 1 \mod p \tag{39}$$

Proof. (証明) 補題 *26* より

$$\prod a(\mathbb{Z}/p\mathbb{Z}) = (p-1)!a^{p-1} \equiv (p-1)! \mod p \tag{40}$$

$p \in \mathbb{P}$ より $p \perp (p-1)!$ なので、合同式両辺を $(p-1)!$ で割ることができ示される。 □

系 32. フェルマーの小定理を用いた逆元　$p \in \mathbb{P}$ のとき整数 a の $\bmod p$ 逆元は a^{p-2}.

Proof. (証明) 定理 31 より、$a \cdot a^{p-2} = a^{p-1} \equiv 1 \mod p$. これは定義 *28* を満たす。 □

アルゴリズム 33. 繰り返し二乗法 / 二分累乗法

```
def pow(x, y, z):
    x %= z
    res = 1
    while y:
        if y & 1:
            res = res * x % z
        y >>= 1
        x = x * x % z
    return res
```

以上のように累乗を計算すると $y \simeq 2^k$ のとき再帰回数は約 k 回、また $k = \log_2 y$ よって計算量はおよそ $O(\log y)$ となる。Python の組み込み関数 `pow()` はおよそこれに同義である。

アルゴリズム 34. フェルマーの小定理を用いた逆元

```
def modinv(a, p = MOD):
    return pow(a, p - 2, p)
```

系 *32, alg.33* からわかるように $p \in \mathbb{P}$ のとき整数 a の $\bmod p$ 逆元を $O(\log p)$ で計算できる。系 *30* で述べたように体 $\mathbb{F}_p$ が形成される。

系 35. 拡張ユークリッドの互除法を用いた逆元 $a \in \mathbb{Z}, m \in \mathbb{N}, a \perp m$ のとき、$\mathrm{egcd}(a, p)$ で得られる x を m で割ったあまりは a の逆元である。

Proof. (略証) 逆元 x は定義から $ax \equiv 1$ を満たす。適当な整数 $-y$ を用いて $ax - 1 = -ym$ となる。よって $ax + ym = 1$. これは拡張ユークリッドの互除法で解くことができる。 □

アルゴリズム 36. 拡張ユークリッドの互除法を用いた逆元

```
def modinv(a, m = MOD):
    g, x, y = egcd(a, m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
        return x % m
```

系 *35*、補題 *12* により $a \in \mathbb{Z}, m \in \mathbb{N}, a \perp m$ のとき $O(\log m)$ 程度で逆元を求めることができる。

定義 37.

$$ {}_nC_r := n!r!^{-1}(n-r)!^{-1} \tag{41}$$

```
from math import factorial as f
P = lambda n, r: f(n) // f(n - r)
C = lambda n, r: f(n) // (f(n - r) * f(r))
H = lambda n, r: f(n + r - 1) // (f(r) * f(n - 1))
from itertools import*
P = lambda n, r: list(permutations(range(n), r))
C = lambda n, r: list(combinations(range(n), r))
H = lambda n, r: list(combinations_with_replacement(range(n), r))
```

アルゴリズム 38. 二項係数 $\bmod p$ は、階乗と階乗逆元のリストを $\mathbb{Z}/p\mathbb{Z}$ で作っておくことで（前計算 $O(n_{\max} + \log p)$ の計算量）、一回あたり $O(1)$ で求めることができる。${}_nP_r, {}_nH_r$ も同様。

```
class Factorial:
    def __init__(self, n, mod):
        # O(n [+ log mod])
        self.f = f =[0] * (n + 1)
        f[0] = b = 1
        for i in range(1, n + 1):
            f[i] = b = b * i % mod
        self.inv = inv = [0] * (n + 1)
        inv[n] = b = modinv(self.f[n], mod)
        for i in range(n, 0, -1):
            inv[i - 1] = b = b * i % mod
        self.mod = mod

    def __call__(self, n, k):
        return self.C(n, k)

    def factorial(self, i):
        return self.f[i]

    def ifactorial(self, i):
        return self.inv[i]

    def C(self, n, k):
        if not 0 <= k <= n: return 0
        return self.f[n] * self.inv[n - k] * self.inv[k] % self.mod

    def P(self, n, k):
        if not 0 <= k <= n: return 0
        return self.f[n] * self.inv[n - k] % self.mod

    def H(self, n, k):
        if (n == 0 and k > 0) or k < 0: return 0
        return self.f[n + k - 1] * self.inv[k] % self.mod * self.inv[n - 1]
            % self.mod
```

例題 39. N 個のブロックからなるタワーのそれぞれのブロックについて、赤で塗ると A 点、青で塗ると B 点、緑で塗ると $A+B$ 点となる。いま、タワーの 0 個以上のブロックをそれぞれ任意の色で塗って合計点を K 点にできる塗り方は何通りか、998244353 で割ったあまりを求めよ。$1 \leq N, A, B \leq 3\times10^5,\ 0 \leq K \leq 18\times10^{10}$. 出典：AGC025 B - RGB Coloring

解） 緑で塗る $\Leftrightarrow$ 赤と青両方で塗る として良い。すると、赤と青それぞれ塗るブロックの選び方は独立。よって赤 a 個、青 b 個塗るとすれば答えは題意命題 $P(a,b) = aA + bB = k, (a,b) \in \mathbb{Z}_{0+}^2$ を満たす a, b に対し $\sum_{P(a,b)} {}_NC_a {}_NC_b$. 計算量は $O(N + K/A)$

```
N, A, B, K = map(int, input().split())
F = Factorial(N, MOD2)
c = 0
for a in range(K // A + 1):
    b, r = divmod(K - a*A, B)
    if r:
        continue
    c = (c + F.C(N, a) * F.C(N, b)) % MOD2
print(c)
# in: 90081 33447 90629 6391049189
# out: 577742975
```

例題 40. 自然数 N, M について $\prod a = M$ なる長さ N の数列 a が何通りあるか $\bmod 10^9+7$ で求めよ。$1 \leq N \leq 10^5,\ 1 \leq M \leq 10^9$. 出典：ABC110 D - Factorization

解） $M = \prod p_i^{e_{iM}}$ と素因数分解し、さらに各要素拡張素因数分解を $a_j = \prod p_i^{e_{ij}} (e_{ij} \in \mathbb{Z}_{0+})$ とする。各 i について独立に $e_{iM} = \sum_{j=1}^{n} e_{ij}$ となるように考えればよく、p_i を配る先を数列の要素 n 個から重複を許して e_{iM} 個選ぶと考えれば、答えは $\prod_i {}_nH_{e_{iM}} \bmod p$ である。なお、${}_nH_k$ の計算時のリストの添字参照は最大 $n+k-1$. ここで $\max_i e_{iM} = \log_2 M_{\max} = \log_2 10^9 \sim 29.897$ より、前計算は $n+30$ 程度で十分である。計算量 $O(n+m)$.

```python
n, m = map(int, input().split())
F = Factorial(n + 30, MOD)
c = 1
for _, E in factorize(m):
    c = c * F.H(n, E) % MOD
print(c)
# in: 100000 1000000000
# out: 957870001
```

3.4 素因数分解 (2) / 約数列挙 / 素数判定

アルゴリズム 41.　試し割り法による約数列挙

```python
def div(n) -> list:
    if n == 0:
        return [0]
    i = 1
    lower_table, upper_table = [], []
    while i * i <= n:
        if n % i == 0:
            lower_table += i,
            upper_table += n // i,
        i += 1
    return lower_table + upper_table[::-1]
```

alg. 3 と同様に約数列挙も行うことができる。全探索的に走査するが、$n \mid p$ のとき $n \mid n/p$ であることを用いれば $O(\sqrt{n})$ ですむ。

アルゴリズム 42.　試し割り法による素数判定

```python
def is_prime(n):
    i = 2
    while i * i <= n:
        if n % i == 0:
            return False
        i += 1
    return n != 1
```

$$p \in \mathbb{P} \coloneqq \forall x \in \left[\min\left(2, \sqrt{p}\right), \sqrt{p}\right] \cap \mathbb{N}, p \nmid x \quad (p \geq 2) \tag{42}$$

素数判定も同じ要領で $O(\sqrt{N})$ で行うことができる。例えば数列 a $\left(\max a = N, |a| = n\right)$ の要素を全て素数判定したいときは最大 $\max O\left(\sum \sqrt{a}\right) = O\left(n\sqrt{N}\right)$ かかってしまう。もう少し厳密にいうと $[2, n)$ の正整数を全て素数判定したいときの計算量 $\simeq \int_0^n \sqrt{x}dx = \frac{2}{3}n\sqrt{n} = O\left(n\sqrt{n}\right)$.

アルゴリズム 43.　エラトステネスの篩 (Sieve of Eratosthenes)

```python
def era(n, option = False) -> list:
    p = [1] * (n + 1)
    p[0] = p[1] = 0
    for x in range(2, int(n**.5) + 1):
        if p[x]:
            for y in range(x*2, n + 1, x):
                p[y] = 0
    return [e for e, q in enumerate(p) if q] if option else p
```

$\sqrt{n}$ 以下の素数について、その倍数（自身を除く）にチェックを入れていくだけの単純なアルゴリ

ズムで前計算 $O(n \log\log n)$（素数リスト作成)、判定 $O(1)$ で素数判定を行えるのが知られている。これを **エラトステネスの篩** (Sieve of Eratosthenes) という。計算量の証明は素数の逆数和の計算量を示せば与えることができる。以下実区間 I に対して $I_{\mathbb{P}} \coloneqq I \cap \mathbb{P}$ などとかく。

3.4.1 素数の逆数和の簡易な計算量証明

補題 44.

$$\log(n+1) < \sum_{i=1}^{n} \frac{1}{i} < 1 + \log n \tag{43}$$

Proof. (証明) グラフから明らかに、

$$\int_0^n \frac{1}{\lceil x \rceil} < 1 + \int_1^n \frac{1}{x} dx, \int_1^{n+1} \frac{1}{x} dx < \int_1^{n+1} \frac{1}{\lfloor x \rfloor} \tag{44}$$

ここで、

$$\int_0^n \frac{1}{\lceil x \rceil} = \int_1^{n+1} \frac{1}{\lfloor x \rfloor} = \sum_{i=1}^{n} \frac{1}{i} \tag{45}$$

$$\int_1^n \frac{1}{x} dx = \Big[\log x\Big]_1^n = \log n \tag{46}$$

$$\int_1^{n+1} \frac{1}{x} dx = \Big[\log x\Big]_1^{n+1} = \log(n+1) \tag{47}$$

より示される。 □

なお、実際 $\lim_{n\to\infty} \sum_{k=1}^{n} \frac{1}{k} - \ln n = \gamma \sim 0.577$ が知られている。

補題 45.

$${}_{2n}C_n = \frac{(2n)!}{n!^2} = \frac{\prod(n, 2n]_{\mathbb{N}}}{n!} = \prod_p p \, \frac{\prod(n, 2n]_{\mathbb{N}\cap\overline{\mathbb{P}}}}{n!} \in \mathbb{N}_+ \tag{48}$$

$$\prod_p p \perp n! \therefore \frac{\prod(n, 2n]_{\mathbb{N}\cap\overline{\mathbb{P}}}}{n!} \in \mathbb{N}_+ \Leftrightarrow \prod_p p \leq {}_{2n}C_n \tag{49}$$

ここで二項定理から

$${}_{2n}C_n = \frac{1}{2} \cdot 2 \cdot {}_{2n}C_n \leq \frac{1}{2} \sum_{i=0}^{2n} {}_{2n}C_i = \frac{1}{2}(1+1)^{2n} = 2^{2n-1} \tag{50}$$

すなわち、(49), (50) から、

$$\sum_{p\in(n,2n]_{\mathbb{P}}} \log p = \log \prod_{p} p \leq \log \frac{(2n)!}{n!^2} < \log 2^{2n} \leq 2n. \tag{51}$$

系 46. 補題 *45* より、

$$\sum_{p\in(n,2n]_{\mathbb{P}}} \frac{1}{p} = \sum_{p} \frac{\log n}{p \log n} \leq \sum_{p} \frac{\log p}{n \log n} \leq \frac{2n}{n \log n} \leq \frac{2}{\log n} \tag{52}$$

定理 47. n 以下の素数の逆数和は $O(\log \log n)$.

Proof. (略証) $2^k < n \leq 2^{k+1}$ となるような自然数 k をとる。すると

$$k \lesssim \log_2 n \tag{53}$$

ここで、補題 46 において $n = 2^i$ とすれば

$$\sum_{p\in(2,n]_{\mathbb{P}}} \frac{1}{p} \leq \sum_{i=1}^{k} \sum_{p\in(2^i,2^{i+1}]_{\mathbb{P}}} \frac{1}{p} \leq \sum_{i=1}^{k} \frac{2}{\log 2^i} = 2\sum_{i=1}^{k} \frac{1}{i} \tag{54}$$

さらに、補題 44 より

$$\sum_{i=1}^{k} \frac{1}{i} \simeq O(\log k) \tag{55}$$

したがって、(53), (54), (55) より示された。 □

3.4.2 素数の逆数和の厳密な計算量証明

定理 48. ルジャンドルの定理　以下の等式が成り立つ。

$$e(p, n!) = \sum_{i=1}^{\infty} \left\lfloor \frac{n}{p^i} \right\rfloor \tag{56}$$

Proof. (証明)

$$e(p, n!) = \sum_{i=1}^{n} e(p, i) \tag{57}$$

であり、求めるのは n 以下の $p, p^2, p^3 ...$ それぞれの倍数の数の合計である。よって自明。 □

補題 49. 定理 *48* より、

$$\frac{n}{p} \leq e(p, n!) < \frac{n}{p-1} \tag{58}$$

両辺 $\log p$ をかけ総和をとれば、区間 $I = [2, n]_{\mathbb{P}}$ において、

$$\forall p \in I, n! \mid p \Leftrightarrow e(p, n!) > 0 \ \text{ can be defined.} \tag{59}$$

$$\therefore \sum_{p} \frac{n \log p}{p} \leq \sum_{p} e(p, n!) \log p < \sum_{p} \frac{n \log p}{p-1} \tag{60}$$

補題 50. $n!_P$ の両辺 log を取ることにより、

$$\log n! = \log \prod_{p} p^{e_p} = \sum_{p} \log p^e = \sum_{p} e(p, n!) \log p \tag{61}$$

定理 51. 補題 *49,50* より、

$$\sum_{p} \frac{n \log p}{p} \leq \log n! < \sum_{p} \frac{n \log p}{p-1} \tag{62}$$

補題 52.

$$\log n! = \log \prod_{k=1}^{n} k = \sum_{k=1}^{n} \log k \tag{63}$$

補題 53. 単調増加関数 $f(x)$ において自明に

$$f(k) \leq \int_k^{k+1} f(x)dx \leq f(k+1) \tag{64}$$

定理 54. 補題 *53* において $f(1) \geq 0$ ならば

$$\int_1^n f(x)dx \leq \sum_{k=1}^{n} f(k) \leq \int_1^n f(x)dx + f(n) \tag{65}$$

補題 55. 定理 *54* において $f = \log$ とすれば

$$\int_1^n \log x dx \leq \sum_{k=1}^{n} \log k \leq \int_1^n \log x dx + \log n \tag{66}$$

ここで

$$\int_1^n \log x dx = \Big[x \log x - x\Big]_1^n = n \log n - n + 1 \tag{67}$$

よって

$$n \log n - n + 1 \leq \sum_{k=1}^{n} \log k \leq n \log n - n + 1 + \log n \tag{68}$$

定理 56. 補題 *52,55* より

$$\log n! = n \log n - n + c_1, \text{ 但し } c_1 \begin{cases} \leq O(\log n) \\ \geq O(1) \end{cases} \tag{69}$$

補題 57.

$$\lim_{x \to \infty} \frac{x}{e^x} = 0 \tag{70}$$

Proof. (証明) $x < 2^x$ より

$$\frac{x}{e^x} < \frac{2^x}{e^x} \tag{71}$$

また

$$\lim_{x\to\infty}\frac{2^x}{e^x}=\lim_{x\to\infty}\left(\frac{2}{e}\right)^x=0 \tag{72}$$

よって (71), (72) から題意は示された。 □

補題 58.

$$\lim_{x\to+\infty}\frac{\log x}{x}=0 \tag{73}$$

Proof. (証明) 補題 57 において $t=e^x$ i.e. $x=\log t$ とすると、$x\to+\infty$ なら $t\to+\infty$.

$$\therefore\ \lim_{t\to+\infty}\frac{\log t}{t}=0 \tag{74}$$

(74) の t を x に置換して (73) を得る。 □

補題 59. 定理 *56* 両辺 n で割れば

$$\frac{1}{n}\log n!=\log n-1+c_2 \tag{75}$$

ここで c_2 は補題 *58* から

$$c_2\le O\left(\frac{\log n}{n}\right)=o(1)\le O(1) \tag{76}$$

$$c_2\ge O\left(\frac{1}{n}\right)=o(1) \tag{77}$$

すなわち

$$\frac{1}{n}\log n!=\log n+O(1) \tag{78}$$

定理 60. 定理 *51,* 補題 *59* より

$$\sum_p\frac{\log p}{p}\le\frac{1}{n}\log n!=\log n+O(1) \tag{79}$$

補題 61. 定理 *51* における (最右辺)−(最左辺) について n で割れば

$$\sum_p \frac{\log p}{p-1} - \sum_p \frac{\log p}{p} = \sum_p \frac{\log p\{p-(p-1)\}}{(p-1)p} = \sum_p \frac{\log p}{(p-1)p} \tag{80}$$

補題 62. $x \geq 1$ において

$$2\sqrt{x} > \log x \tag{81}$$

Proof. (証明) $f(x) = 2\sqrt{x} - \log x$ とする。ここで

$$f(1) = 2\sqrt{1} - \log 1 = 2 > 0 \tag{82}$$

さらに

$$f'(x) = \frac{1}{\sqrt{x}} - \frac{1}{x} = \frac{\sqrt{x}-1}{x} \tag{83}$$

$f'(x) = 0$ のとき $x = 1$ であり $x \geq 1$ において単調性は変化しない。さらに

$$f'(4) = \frac{\sqrt{4}-1}{4} = \frac{1}{2} > 0 \tag{84}$$

よって $f(x)$ は $x \geq 1$ において単調増加。これと (81) より

$$f(x) > 0 \tag{85}$$

よって題意は示された。 □

定義 63. リーマンのゼータ関数 $\zeta(s)$ を以下のように定める。

$$\zeta(s) \coloneqq \sum_{k=1}^{\infty} \frac{1}{k^s} \tag{86}$$

定理 64. $s > 1$ なる実数 s について $\zeta(s)$ は収束する。

Proof. (証明) グラフを考えれば

$$\int_0^l \frac{dx}{\lceil x \rceil^s} = \sum_{k=1}^{l} \frac{1}{k^s} < 1 + \int_1^l \frac{dx}{x^s} \tag{87}$$

右辺は

$$1 + \left[\frac{x^{1-s}}{1-s} \right]_1^l = 1 + \frac{l^{1-s} - 1}{1-s} \tag{88}$$

$1 - s < 0$ より $\lim_{l \to \infty}$ について

$$\sum_{k=1}^{\infty} \frac{1}{k^s} < 1 + \frac{1}{s-1} \tag{89}$$

よって単調増加で上に有界な数列は収束することを既知とすれば題意は示される。 □

定理 65.

$$\text{数列和} \sum_{i=1}^{\infty} a_i \text{ が収束するとき定数} c \text{ 倍の} \sum_{i=1}^{\infty} ca_i \text{ もまた収束する。} \tag{90}$$

Proof. (証明)

$$\sum_{i=1}^{\infty} ca_i = c \sum_{i=1}^{\infty} a_i \text{ より自明。}$$

□

補題 66.

$$\sum_{x=1}^{\infty} \frac{\log x}{x^2} \quad \text{は収束する。} \tag{91}$$

Proof. (証明) 定理 *64, 65* を用いれば、

$$\sum_{x=1}^{\infty} \frac{\log x}{x^2} \leq \sum_{x=1}^{\infty} \frac{2\sqrt{x}}{x^2} = \sum_{x=1}^{\infty} \frac{2}{x^{3/2}} < \infty \tag{92}$$

□

補題 67.

$$\sum_p \frac{\log p}{(p-1)p} \quad \text{は収束する。} \tag{93}$$

Proof. (証明) 定理 *65,* 補題 *66* を用いれば、

$$0 < \sum_p \frac{\log p}{(p-1)p} \leq 2 \sum \frac{\log p}{p^2} \leq 2 \sum_{x=1}^{\infty} \frac{\log x}{x^2} < \infty \tag{94}$$

□

定理 68. 定理 *60,* 補題 *61,* 補題 *67* より、以下の式評価を満たす定数 c_3, c_4 が存在する。

$$\sum_p \frac{\log p}{p} \geq \sum_p \frac{\log p}{p-1} - c_3 > \frac{1}{n} \log n! - c_3 \geq \log n - c_4 \tag{95}$$

定理 69. 定理 *60,* 定理 *68* をまとめると、

$$\sum_p \frac{\log p}{p} = \log n + O(1) \tag{96}$$

補題 70. 非負整数 $a < b$ について、$f(n), u(n)$ が $\mathbb{N}$ 上の関数であり、

$$U(t) = \sum_{n \leq t} u(n) \quad \text{とするとき、} \tag{97}$$

$$\sum_{n=a+1}^{b} u(n) f(n) = U(b) f(b) - U(a) f(a+1) - \sum_{n=a+1}^{b-1} U(n)(f(n+1) - f(n)) \tag{98}$$

Proof. (証明) $U(n) - U(n-1) = u(n)$ である。

$$\sum_{n=a+1}^{b} u(n) f(n) = \sum_{n=a+1}^{b} (U(n) - U(n-1)) f(n) \tag{99}$$

$$= \sum_{n=a+1}^{b} U(n) f(n) - \sum_{n=a+1}^{b} U(n-1) f(n) \tag{100}$$

$$= \sum_{n=a+1}^{b} U(n) f(n) - \sum_{n=a}^{b-1} U(n) f(n+1) \tag{101}$$

$$= U(b)f(b) + \sum_{n=a+1}^{b-1} U(n)f(n) - U(a)f(a+1) - \sum_{n=a+1}^{b-1} U(n)f(n+1) \quad (102)$$

$$= U(b)f(b) - U(a)f(a+1) - \sum_{n=a+1}^{b-1} U(n)(f(n+1) - f(n)) \quad (103)$$

□

補題 71. 部分和法 補題 *70* における $u(n), f(n), U(n)$ において、実数 $x \geq 1$ について $f(t)$ が区間 $[1, x]$ において微分可能な関数で、さらに $f'(t)$ はこの区間において連続な関数とするとき、

$$\sum_{n \geq x} u(n)f(n) = U(x)f(x) - \int_1^x U(t)f'(t)dt \quad (104)$$

Proof. (証明) 補題 *70* において $a = 0, b = \lfloor x \rfloor$ とすると、$U(a) = 0$ より

$$\sum_{n \geq x} u(n)f(n) = \sum_{n=1}^{b} u(n)f(n) \quad (105)$$

$$= U(b)f(b) - \sum_{n=1}^{b-1} U(n)(f(n+1) - f(n)) \quad (106)$$

ここで、b の定義から $x \in [b, b+1)$ より、$t \in [b, b+1)$ のとき $U(t) = U(b)$ となり、

$$\int_b^x U(t)f'(t)dt = U(b)\int_b^x f'(t)dt = U(b)(f(x) - f(b)) \quad (107)$$

さらに、同様に $n \in [1, b)$ に対して、$t \in [n, n+1)$ のとき $U(t) = U(n)$ となり、

$$\int_n^{n+1} U(t)f'(t)dt = U(n)\int_n^{n+1} f'(t)dt = U(n)(f(n+1) - f(n)) \quad (108)$$

(107), (108) から

$$\int_1^x U(t)f'(t)dt = \int_b^x U(t)f'(t)dt + \sum_{n=1}^{b-1}\int_n^{n+1} U(t)f(t')dt \quad (109)$$

$$= U(b)(f(x) - f(b)) + \sum_{n=1}^{b-1} U(n)(f(n+1) - f(n)) \quad (110)$$

さらに $U(b) = U(x)$ であり、

$$U(b)f(b) - \sum_{n=1}^{b-1} U(n)(f(n+1) - f(n)) = U(x)f(x) - \int_1^x U(t)f'(t)dt \quad (111)$$

(106), (111) から示された。 □

補題 72. 以下次のようにすると（但し $U(t)$ は (97) の通り和を取る関数)、

$$f(t) = \frac{1}{\log n},\ u(m) = \begin{cases} \dfrac{\log p}{p}, & \text{if } m = p \in \mathbb{P} \\ 0, & \text{otherwise} \end{cases} \tag{112}$$

自明に以下が成り立つ。

$$U(t) = \sum_p \frac{\log p}{p} \tag{113}$$

補題 73. 以下次のようにすると、

$$g(t) = U(t) - \log t \tag{114}$$

定理 *69* から、

$$|g(t)| = O(1) \tag{115}$$

したがって、$p \geq 2$ について、

$$\left| \int_p^\infty \frac{g(t)}{t \log^2 t} dt \right| \leq \int_p^\infty \frac{|g(t)|}{t \log^2 t} dt = O(1) \left[-\frac{1}{\log t} \right]_p^\infty = \frac{O(1)}{\log p} \tag{116}$$

補題 74. 補題 *71* から、

$$\sum_p \frac{1}{p} = \sum_{k \leq n} f(k)u(k) \tag{117}$$

$$= U(n)f(n) + \int_1^n U(t)f'(t)dt \tag{118}$$

$$= U(n)f(n) + \int_2^n U(t)f'(t)dt \tag{119}$$

$$= \frac{\log n + g(n)}{\log n} + \int_2^n (\log t + gt)f'(t)dt \tag{120}$$

補題 75. 補題 *74* と $f'(t) = -t^{-1} \log^{-2} t$ から、

$$\sum_p \frac{1}{p} = \frac{\log n + g(n)}{\log n} + \int_2^n \frac{dt}{t \log t} + \int_2^n \frac{g(t)}{t \log^2 t} dt \tag{121}$$

$$= 1 + \frac{g(n)}{\log n} + \Big[\log\log t\Big]_2^n + \int_2^n \frac{g(t)}{t\log^2 t}dt \tag{122}$$

$$= 1 + \frac{g(n)}{\log n} + \log\log n + \int_2^\infty \frac{g(t)}{t\log^2 t}dt - \int_n^\infty \frac{g(t)}{t\log^2 t}dt \tag{123}$$

定理 76. 補題 *73* の (115) あるいは (116) と 補題 *75* から、

$$\sum_p \frac{1}{p} = \log\log n + 1 + \frac{O(1)}{\log 2} + \frac{g(n) - O(1)}{\log n} \tag{124}$$

$$= \log\log n + O\left(\frac{1}{\log n}\right) + O(1) \tag{125}$$

$$= \log\log n + o(1) \tag{126}$$

あるいは $$= O(\log\log n) \tag{127}$$

以上により素数の逆数和は $O(\log\log n)$. これが示されるべきことであった。

定理 77. エラトステネスの篩の計算量 エラトステネスの篩の前計算量は $O(n\log\log n)$ である。

Proof. (証明) 定理 *47* あるいは *76* から、n 以下の素数の逆数和は $O(\log\log n)$ である。また、エラトステネスの篩で探索する各素数について走査する倍数の数は n/p 程度である。また、素数 p は $\sqrt{n}$ 程度で探索を打ち切る。よって、

$$\sum_{p<\sqrt{n}} \frac{n}{p} = n \sum_{p<\sqrt{n}} \frac{1}{p} \tag{128}$$

$$= n\ O(\log\log\sqrt{n}) \tag{129}$$

$$= n\ O(\log\log n + \log 1/2) \tag{130}$$

$$= O(n\log\log n) \tag{131}$$

□

アルゴリズム 78. 改良されたエラトステネスの篩

```
def prime_checker(n, option = False) -> list:
    p = [False, True, False, False, False, True] * (n // 6 + 1)
    del p[n + 1:]
    p[1 : 4] = False, True, True
    for x in range(5, int(n**.5 + 1)):
        if p[x]:
            p[x * x :: 2*x] = [False] * ((n // x - x) // 2 + 1)
    return [e for e, q in enumerate(p) if q] if option else p
```

全く同じエラトステネスの篩を `bool` と配列を使って定数倍高速化したもの。2 と 3 の倍数を初めにチェックしている（リストを $\mathrm{lcm}(2,3)=6$ 周期で複製する）ことは簡単に分かる。これを Wheel factorization という。およそ 4〜8 倍速くなっている。さらに、内側を $2p$ とばしでチェックをしている。これは 2 でない素数 p に対して $2np$ はすでにチェックされていることは自明だからである。これでおよそ 2 倍速くなっている。また、内側の開始インデックスは p^2 になっている。これもよく考えれば自明である。なお、計算量は

$$\sum_{p\leq\sqrt{n}}\frac{n}{p}-p=\sum_{p\leq\sqrt{n}}\frac{n}{p}-\sum_{p\leq\sqrt{n}}p \tag{132}$$

ここで自明に

$$\sum_{p\leq\sqrt{n}}<\sum_{k=1}^{\lfloor\sqrt{n}\rfloor}k\simeq\frac{\sqrt{n}(\sqrt{n}+1)}{2}=O\left(\sqrt{n}^2\right)=O(n)<O(n\log\log n) \tag{133}$$

なのでオーダー表記では変わらない。

アルゴリズム 79. Osa_k 法による素因数分解

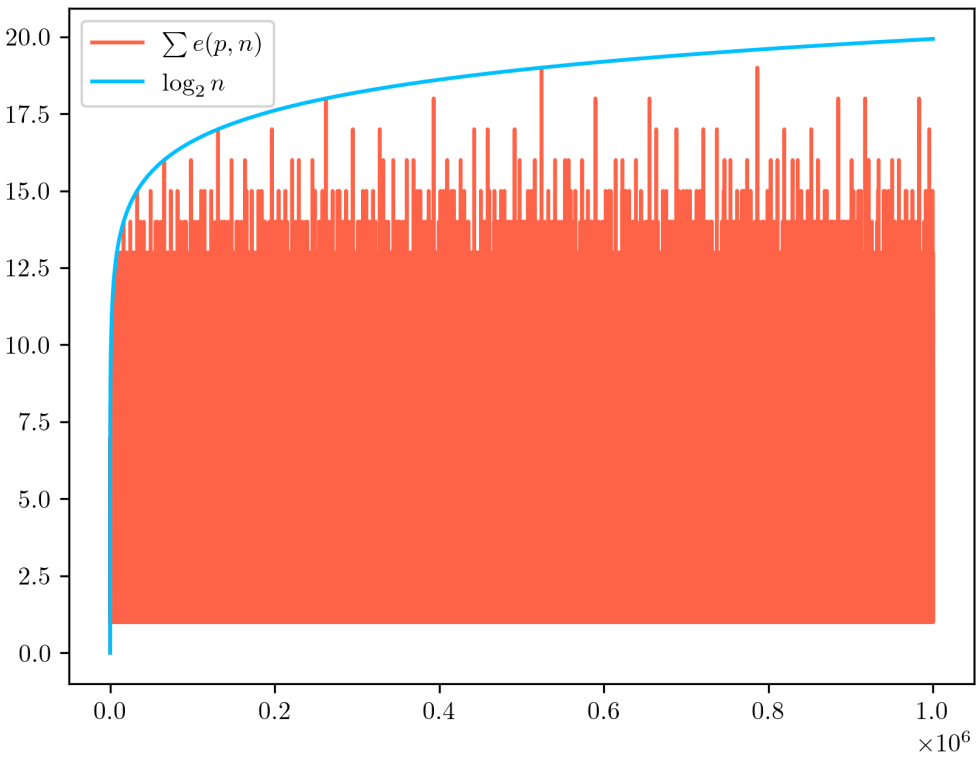
```python
class Osa_k:
    def __init__(self, n_max):
        self.n_max = n_max
        self.min_factor = min_factor = list(range(n_max + 1))
        min_factor[2::2] = [2] * (n_max // 2)
        min_factor[3::6] = [3] * ((n_max + 3) // 6)
        for i in range(5, int(n_max ** .5) + 1, 2):
            if min_factor[i] == i:
                for j in range(i*i, n_max + 1, i):
                    if min_factor[j] == j:
                        min_factor[j] = i

    def __call__(self, n):
        if not 1 <= n <= self.n_max: raise ValueError("Invaild Value!")
        min_factor = self.min_factor
        n_twoes = (n & -n).bit_length() - 1
        res = [2] * n_twoes
        n >>= n_twoes
        resappend = res.append
        while n > 1:
            p = min_factor[n]
            resappend(p)
            n //= p
        return res
```

エラトステネスの篩をするときに捨てている「最小の素因数」の情報を記録することで、「最小の素因数で割っていく」を繰り返して $N = \max_i n_i$ について前計算 $O(N \log\log N)$, 各数 $O(\log n)$ で素因数分解できる。これを **Osa_k 法** という。bit でしている処理は素因数 2 の場合を定数倍高速化するものであり、最悪ケースで速くなる。（次に続く）

補題 80. *Osa_k* 法による素因数分解の計算量は前計算 $O(N \log\log N)$, 各数 $O(\log n)$ である。

Proof. (証明) 前計算はエラトステネスの篩による。1 クエリに対する処理計算回数は、実装から $\sum e(p,n)$ 回だとわかる。これは n が最小の素数である 2 の冪乗であるとき値に対して最大になるとわかる。すなわち最大の場合 $\log_2 n$ で回数は求められるから、計算量は $O(\log n)$. □

```python
import math
import matplotlib
import matplotlib.pyplot as plt
plt.rc('text', usetex = True)
fig = plt.figure()
r = range(1, 10**6)
*x, = r
y1 = [len(P(i)) for i in r]
y2 = [math.log2(i) for i in r]
plt.plot(x, y1, color = "tomato", label = "$\sum e(p,n)$")
plt.plot(x, y2, color = "deepskyblue", label = "$\log_2n$")
plt.legend()
fig.savefig("img.png", dpi = 300)
```



補題 81.

$$d(n) = \prod e(p, n) + 1 \tag{134}$$

である。ただし $d(n)$ は n の正の約数の個数である。

アルゴリズム 82. Osa_k 法による約数列挙

```
Class Osa_k():
    # Folding

    def d(self, n):
        min_factor = self.min_factor
        t = []
        tappend = t.append
        while n > 1:
            if t and t[-1][0] == min_factor[n]:
                t[-1][1] += 1
            else:
                tappend([min_factor[n], 1])
            n //= min_factor[n]
        res = 1
        for _, i in t:
            res *= i + 1
        return res

    def div(self, n):
        factors = self.__call__(n)
        res = {1, n}
        for i in range(1, len(factors)):
            for j in combinations(factors, i):
                tmp = 1
                for i in j:tmp *= i
                res |= {tmp}
        return res
```

補題 *81* から明らかなように、約数個数は簡単に求めることができる。計算量 $O(\log n)$. 約数列挙は $O(\log n + d(n))$ ？ ここでは最適化できていない。

アルゴリズム 83. 区間篩

```python
def segmented_seive(l, r):
    rs = int(r ** .5) + 1
    # [2, √r]
    prime_small = [True] * rs
    prime_small[4::2] = [False] * (rs//2 - 1)
    # [l, r]
    prime = [True] * (r - l + 1)
    prime[l%2::2] = [False] * (r // 2 - (l - 1) // 2)
    for x in range(3, rs, 2):
        if prime_small[x]:
            for y in range(x*x, rs, x*2):
                prime_small[y] = False
            for z in range(((l + x - 1) // x | 1) * x, r, x*2):
                prime[z - l] = False
```

前述したように区間 $[l, r]$ の素因数は $\sqrt{r}$ 以下であるから、これを利用して特定区間に対して篩をかけることができる。これを**区間篩**といい、計算量は（おそらく）$O((r-l)\log\log r)$. ただし $\log\log\sqrt{r} = \log\log r + \log 2$ による。ただ上のコードは最適化されておらず 10^9 を超えたあたりですぐに `MemoryError` を吐き（numba の jit である程度は回避可能）、逆に 10^7 あたりであればエラトステネスの篩を普通に使って求めた方が明らかに高速なので、Python で使う限り出番があるかは微妙... numpy のベクトルを使えば最適化はできるかもしれない。もちろんこれを用いて

素因数分解もできる。

例題 84. $\sum_{K=1}^{N} Nd(N)$ を求めよ。$1 \leq N \leq 10^7$. TL : 3 sec　　ABC172 D - Sum of Divisors

---

```python
# 2667 ms (PyPy)
P = Osa_k(10**7)
print(sum(i * P.d(i) for i in range(1, int(input()) + 1)))
```

```python
# 2096 ms (PyPy)
n = int(input())
F = [1] * -~n
for i in range(2, n + 1):
    for j in range(i, n + 1, i):
        F[j] += 1
print(sum(e * v for e, v in enumerate(F)))
```

例題 85. 長さ N の数列 A が与えられるので、自然数 $i \leq N$ であって $\forall j \in [1, N]_{\mathbb{Z}}, A_j \nmid A_i$ であるようなものの数を求めなさい。$1 \leq N \leq 2 \times 10^5, 1 \leq A_i \leq 10^6$　ABC170 D - Not Divisible

---

```python
# 1515 ms (PyPy)
from collections import Counter
n, *a = map(int, open(0).read().split())
b, c, P, cnt = set(a), Counter(a), Osa_k(10**6), 0
for i in b:
    if c[i] > 1:continue
    for d in P.div(i):
        if d != i and d in b:break
    else:
        cnt += 1
```

```
print(1 if n == 1 else cnt)
```

```
# 665 ms (Python)
from collections import Counter
n, *A = map(int, open(0).read().split())
A.sort()
c, s = Counter(A), set()
M = max(A) + 1
ans = 0
for a in A:
    if a in s:
        continue
    ans += c[a] == 1
    for i in range(a, M, a):
        s.add(i)
print(ans)
```

アルゴリズム 86. ロー法による素因数分解

```
# O(⁴√n polylog(n))
def fast_prime_factorization(n):
    from subprocess import Popen, PIPE
    return [*map(int, Popen(["factor", str(n)], stdout=PIPE).communicate
        ()[0].split()[1:])]

def fast_prime_factorization_many(lst):
    from subprocess import Popen, PIPE
    res = Popen(["factor"] + list(map(str, lst)), stdout=PIPE).communicate
        ()[0].split(b"\n")[:-1]
    return [[*map(int, r.split()[1:])] for r in res]
```

---

bash には `factor` コマンドがあり、これを用いて比較的高速に素因数分解することができる。

$O(\sqrt[4]{n}\mathrm{Li}(n))$ 程度（ただし $\mathrm{Li}(n)$ は多重対数 (polylog) 関数）。

アルゴリズム 87. 2015, Forisek らによる素数判定

```python
import array
import bz2
import gzip
import base64
# depends on an env.
int32 = "l" if array.array("l").itemsize == 4 else "i"

bases = b'H4sIAEUphV0C/yy9X0zUabbv/dB0W...' # ...
bases = base64.b64decode(bases)
bases = gzip.decompress(bases)
bases = array.array(int32, bases)
assert len(bases) == 16384

def is_SPRP(n, a):
    if n==a: return True
    if n%a==0: return False
    d = m1 = n-1
    s = (d & -d).bit_length() - 1
    d >>= s
    cur = pow(a, d, n)
    if cur == 1: return True
    for _ in range(s):
        if cur == m1: return True
        cur = cur * cur % n
    return False

def is_prime(x):
    if x in {2, 3, 5, 7}: return True
    if x%2==0 or x%3==0 or x%5==0 or x%7==0: return False
    if x<121: return x>1
    if not is_SPRP(x, 2): return False
    h = ((x >> 32) ^ x) * 0x45d9f3b3335b369 & 0xffffffffffffffff
    h = ((h >> 32) ^ h) * 0x3335b36945d9f3b & 0xffffffffffffffff
```

```
    h = ((h >> 32) ^ h)
    b = bases[h & 16383]
    return is_SPRP(x, b&4095) and is_SPRP(x, b>>12)
```

Forisek and Jancina, 2015 を基に設計された素数判定アルゴリズムである（実装は *Lgeu* 氏のものによります)。小さな数字では効果は見られないが、例えば超巨大な 23 桁程度の数も 41ms, メモリ 4KB と非常に軽動作で処理することができる。`bases` はあまりに長いので省略している。

4　テクニック

アルゴリズム 88. 二分探索 *(Binary Search)* とは、ソート済みのリストに対する検索を行うアルゴリズムであり、中央の値を見て、検索したい値との大小関係を用いて、検索したい値が中央の値の右にあるか、左にあるかを判断して、片側には存在しないことを確かめながら検索する手法。区間の範囲が半分半分に絞られていくイメージ。すなわち計算量 $O(\log N)$.

```
from bisect import*
bisect_left(a, x, low = 0, high = len(a))
bisect_right(a, x, low = 0, high = len(a))
# bisect() = bisect_right()
# insort(a, x) := a.insert(bisect(a, x))
```

Python には標準ライブラリに `bisect` モジュールがある。

例題 89. $A_i < B_j < C_k$ となるような整数 $1 \leq i, j, k \leq N$ の組の数を求めよ。$1 \leq N \leq 10^5$, $1 \leq A_i, B_i, C_i, \leq 10^9$. ABC077 C - Snuke Festival

```python
from bisect import*
(n,), a, b, c = [sorted(map(int, o.split())) for o in open(0)]
print(sum(bisect_left(a, B_j) * (n - bisect(c, B_j)) for B_j in b))
```

解）j を固定すると、二分探索によって B_j より小さい / 大きい要素の数を数えることができるので、j を全探索すれば良く、$O(N \log N)$ で解くことができた。

例題 90. L_i から三辺を選んで作れるような三角形の種類の数を求めよ。$3 \leq N \leq 2 \times 10^3$, $1 \leq L_i \leq 10^3$. ABC143 D - Triangles

```python
from bisect import*
n, *l = map(int, open(c:=0).read().split())
l.sort()
for i in range(n):
    for j in range(i+1, n):
        d = bisect_left(l, l[i] + l[j])
        c += max(0, d - j - 1)
print(c)
```

解）L をソートし、$i < j < k$ かつ $L_k < L_i + L_j$ となるように選べば良い。$O(N^2 \log N)$.

アルゴリズム 91. 二分法 *(Bisection method)* では、単調増加関数 f に対して二分探索をする。これに `bisect` は対応していないので、自己実装する必要がある。下のように半開区間 $(ng, ok]$ / $[ok, ng)$ で考える二分探索 / 二分法を提唱者 *(@meguru_comp)* からめぐる二分探索という。二

分探索は自明に計算量は $O(\log n)$. 二分法も $O(\log |ok - ng|)$ 程度。

```
def meg_bisect(ng, ok, func):
    while abs(ok - ng) > 1:
        mid = (ok + ng) // 2
        if func(mid):
            ok = mid
        else:
            ng = mid
    return ok
```

例題 92. $d(N) \coloneqq N$ の十進表記桁数 とするとき、区間 $[1, 10^9]$ にあるような整数 N を買うのに必要な金額は $AN + Bd(N)$ 円である。X 円で買える最大の整数はなにか。存在しない場合は 0 とする。$1 \leq A, B \leq 10^9$, $1 \leq X \leq 10^{18}$. ABC146 C - Buy an Integer

---

```
a, b, x = map(int, input().split())
print(meg_bisect(10**9 + 1, 0, lambda i:a * i + b * len(str(i)) <= x))
```

例題 93. 長さ A_i の丸太 N 本を合計 K 回まで切ったとき丸太の長さの最大値の最小値切り上げを求めよ。$1 \leq N \leq 2 \times 10^5$, $0 \leq K \leq 10^9$, $1 \leq A_i \leq 10^9$. ABC174 E - Logs

---

```
n, k, *a = map(int, open(0).read().split())
print(meg_bisect(0, max(a), lambda x:sum(0--b // x - 1 for b in a) <= k))
```

最大値の最小化は二分探索！ 典型！ x を最大値にするのに必要な切断回数は $\sum \lceil A_i/x \rceil - 1$.

例題 94. 赤い花 R 本, 青い花 B 本のうちいくつかを使って、それぞれ (x 本, 1 本) で赤い花束、(1 本, y 本) で青い花束を作れる。このとき、合計で花束は最大何本作れるか。$1 \leq R, B \leq 10^{18}$, $2 \leq x, y \leq 10^9$. ARC050 B - 花束

```
R, B, x, y = map(int, open(0).read().split())
print(meg_bisect(min(R, B) + 1, 0, lambda k:(R - k) // (x - 1) + (B - k)
    // (y - 1) >= k))
```

---

k 本の花束を作るには赤青とも k 本は必ず必要であり、それをまず最初に引いて考えると、作れる花束の数は $\left\lfloor \frac{R-k}{x-1} \right\rfloor + \left\lfloor \frac{B-k}{y-1} \right\rfloor$ であり、これが k 本を超えていれば良い。解 x がとりあえず存在すると仮定して Greedy に可能かを判定する問題として解くのは典型。

例題 95. 最初高度 H_i にある風船 N 個について、競技開始 0 秒後から 1 秒に 1 個の任意の風船を撃ち落とすことができるが、それぞれの風船は毎秒 S_i の速度で上がっていく。それぞれの風船についてそれを撃ったときの高度がペナルティになるので、その最大値の最小値を求めよ。$1 \leq N \leq 10^5$, $1 \leq H_i, S_i \leq 10^9$. TL : 5 sec ABC023 D - 射撃王

```
inf = 10**16 # max(H + N * S)
(n,), *d = [[*map(int, o.split())] for o in open(0)]
print(meg_bisect(0, inf, lambda x: all(l >= e for e, l in enumerate(sorted
    ((x - h) // s for h, s in d)))))
```

最大値を x に保つには、それぞれの i についてリミット $\left\lfloor \frac{x-H_i}{S_i} \right\rfloor$ 秒を記録し、ソートして厳しい順に Greedy に処理していって間に合うか試せば良い。計算量 $O(N \log \inf)$.

例題 96. $f(t) := At + B\sin(Ct\pi)$. $f(t) = 100$ となるような 非負実数 t をひとつ求めよ。誤差は $|f(t) - 100| \leq 10^{-6}$ であれば認める。$1 \leq A, B, C \leq 100$. ABC026 D - 高橋君ボール 1 号

```
def meg_bisect_r(ng, ok, func, eps = 1e-9):
    while abs(ok - ng) > eps:
        mid = (ok + ng) / 2
    # folding...
from math import*
A, B, C = map(int, input().split())
f = lambda t:A * t + B * sin(C * t * pi)
print(meg_bisect_r(-1, 200, lambda t: f(t) >= 100, 1e-12))
```

```
from scipy.optimize import*
print(newton(lambda t: f(t) - 100, 0))
```

実数の二分法をするには、誤差を設定して行う。

例題 97. 濃度 p_i で w_i グラムの食塩水 N 個について、K 個混ぜ合わせたときの濃度の最大値を求めよ。$1 \leq K, N \leq 1000, 1 \leq w_i \leq 10^9$. ABC034 D - 食塩水

```
(n, k), *d = [[*map(int, o.split())] for o in open(0)]
print(meg_bisect_r(100.01, 0, lambda q: sum(sorted(w * (p - q) for w, p in
    d)[-k:]) >= 0))
```

混ぜ合わせた後の食塩水の濃度が q 以上、つまり $\dfrac{\sum w_i p_i}{\sum w_i} \leq q$ となるには、$\sum w_i p_i - q\sum w_i \geq 0$ すなわち $\sum w_i(p_i - q) \geq 0$ ならばよい。

アルゴリズム 98. 三分探索 *(Ternary Search)* とは、高々極値が 1 つであるような凸関数に対する三分割探索法。

```
def trisect(func, left, right, eps = 1e-9):
    while abs(right - left) > eps:
    # for _ in [None] * limit:
        mid_l = (left * 2 + right) / 3
        mid_r = (left + right * 2) / 3
        if func(mid_l) < func(mid_r):
            right = mid_r
        else:
            left = mid_l
        return (right + left) / 2
```

例題 99. ムーアの法則が正しいとすると、x 年後にコンピュータの速度は $2^{x/1.5}$ 倍になる。現代で P 年かかる計算は最小でいつ終わらせることができるか。すなわち、計算を始めるまでの時間とそこから計算にかかる時間の合計の最小値をもとめよ。$P \in (0, 10^{18}]_{\mathbb{R}}$.

```
P = float(input())
f = lambda x: x + P / pow(2, x / 1.5)
print(f(trisect(f, 0, 100)))
```

$f(x) := x + 2^{-x/1.5}P$ は凸関数である。三分探索で $\min f(x) = f(\mathrm{argmax} f(x))$ と求める。

アルゴリズム 100. 累積和 *(accumulate)* とは、事前に列 A_i について $A'_i = \sum_{j=0}^{i-1} A_j$ $(A'_0 = 0)$ となるようなリストを作っておくことで配列上の区間の総和を求めるクエリを $O(1)$ で処理する方法である。例えば $\sum_{k \in [i,j)} A_k = A'_j - A'_i$ と求めることができる。Python では `itertools.accumulate` を用いれば良いが、A'_0 を含んでくれないので適宜調整する。

例題 101. 長さ N である数列 A の **空でない連続する** 部分列であって、その総和が 0 になるものの個数を求めよ。なお、列として同じでも位置が違うものは違う部分列とする。$1 \leq N \leq 2 \times 10^5$, $-10^9 \leq A_i \leq 10^9$. AGC023 A - Zero-Sum Ranges

---

```
from itertools import*
from collections import*
*A, = accumulate(map(int, open(0).read().split()[1:]))
print(sum(k * (k - 1) // 2 for k in Counter([0] + A).values()))
```

添字区間 $[i, j]$ での部分列の総和が 0 になるとは、$A'_{j-1} = A'_i$ となることである。このような (i, j) の選び方は、ある要素が数列 A'_i に含まれる数を k とすると $\sum_k {}_kC_2 = \sum_k k(k-1)/2$ と求められる。

例題 102. Q 個のクエリで奇数 $l_i \leq r_i$ が与えられるので、区間 $[l_i, r_i]$ 内で x も $(x+1)/2$ も素数となるような奇数 x の数を求めよ。$1 \leq N, l_i, r_i \leq 10^5$. ABC084 D - 2017-like Number

---

```
INF = 10**5 + 1
p, q = prime_checker(INF), [0] * INF
from itertools import*
```

```
for i in range(1, INF, 2):
    q[i] = p[i] * p[(i + 1)//2]
*q, = accumulate([0] + q)
for _ in [None] * int(input()):
    l, r = map(int, input().split())
    print(a[r + 1] - a[l])
```

事前に配列 p としてエラトステネスの篩を構成する。そして $q_i \coloneqq p_i \wedge p_{(i+1)/2}$ とすれば q は条件を満たすか示す配列である。あとは累積和 q' をとってクエリに回答する。

例題 103. 数列 A_i のうち、最大 1 つの要素を $[1, 10^9]$ 内の整数に書き換えたとき、全体の最大公約数の最大値を求めよ。$2 \leq N \leq 10^5$, $1 \leq A_i \leq 10^9$. ABC125 C - GCD on Blackboard

---

```
from itertools import*
from math import*
n, *a = map(int, open(0).read().split())
*f, = accumulate([0] + a, gcd)
*b, = accumulate(a[::-1], gcd)
print(max(gcd(s, t) for s, t in zip(f, b[::-1])))
```

まず、違う整数に置き換えたところで最大公約数が増えることはなく、置き換える動作は取り除く動作としてよい。`itertools.accumulate` は第 2 引数を取ることができ、gcd は結合則が成り立つことから累積でいいとわかる。自分より左の区間と右の区間が必要なので逆からの累積もとる。

アルゴリズム 104. 2 次元累積和 累積和は同様に 2 次元にも拡張できる。

```
# [x1, x2) × [y1, y2)
def ac2(s:list, x1, x2, y1, y2):
    return s[x2][y2] - s[x1][y2] - s[x2][y1] + s[x1][y1]
```

例題 105. $N \times N$ のたこ焼き器があり、それぞれの場所に美味しさ D_{ij} が決まっている。いま、Q 人の店員がやってきて、長方形の形をした領域で最大 P_k このたこ焼きを焼くとき、それぞれの店員について美味しさの総和の最大値を求めよ。$1 \leq N \leq 50$, $1 \leq D_{ik} \leq 100$, $1 \leq Q, P_k \leq N^2$.

TL:5sec　ABC005 D - おいしいたこ焼きの焼き方

```
import numpy as np
from itertools import*
n = int(input())
D = np.array([[0]*-~n]+[[0]+[*map(int, input().split())] for _ in [0]*n])
D = np.cumsum(D, axis = 0)
D = np.cumsum(D, axis = 1)
dp = [0] * (n * n + 1)
for x1 in range(n):
    for i in range(1, n - x1 + 1):
        x2 = x1 + i
        for y1 in range(n):
            for j in range(1, n - y1 + 1):
                y2 = y1 + j
                dp[i*j] = max(dp[i*j], ac2(D, x1, x2, y1, y2))
*dp, = accumulate(dp, max)
for _ in [None] * int(input()):
    print(dp[int(input())])
```

`np.cumsum` を使うと実装が楽。`itertools.accumulate(dp, max)` は「今までで出た最大値」を取

る動作になっている。$O(N^4)$.

アルゴリズム 106. imos 法 は、配列上の特定区間 $[l, r]$ に v 加算するクエリを、0 で初期化した配列に対し l に v 加算、$r+1$ に v 減算しての操作を繰り返し最後に累積和を取ることで処理する方法である。*@imos* 氏が京都大学院での修士論文で取りまとめたことからこの名がある。

```python
from itertools import*
class Imos:
    def __init__(self, n):
        self.B = [0] * n
        self.n = n

    def __call__(self, l, r, v = 1):
        l, r = max(l, 0), min(r, self.n - 1)
        self.B[l] += v
        if r + 1 != self.n:
            self.B[r + 1] -= v

    def out(self):
        *res, = accumulate(self.B)
        # self.__init__(self.n)
        return res
```

例題 107. 黒面に 0、白面に 1 がかかれた N 個のオセロが最初全て黒面を表に一列に並べられている。Q 回の操作で $[l_i, r_i]$ 番目全てを裏返すことを繰り返すとき最終盤面を出力せよ。$1 \leq l_i \leq r_i \leq N, Q \leq 2 \times 10^5$. ABC035 C - オセロ

```
(n, q), *Q = [[*map(int, o.split())] for o in open(0)]
imos = Imos(n + 1)
for l, r in Q:
    imos(l, r)
print(*[i%2 for i in imos.out()[1:]], sep = "")
```

加算する操作として最後に　mod 2 を取ればよい。

例題 108. 濃さ 0 から 10^6 までの濃さの絵具がそれぞれ売られている。n 人の顧客がそれぞれ $[a_i, b_i]$ 内の濃さの絵具なら購入するとき、最も多くの顧客に買われる絵具は何人に買われるだろうか。$1 \leq n \leq 10^5$, $0 \leq a_i \leq b_i \leq 10^6$. TL:3sec　ABC014 C - AtColor

---

```
imos = Imos(10**6 + 1)
_, *q = [[*map(int, o.split())] for o in open(0)]
for l, r in q:
    imos(l, r)
print(max(imos.out()))
```

例題 109. C 個のチャンネルがあるテレビで N 個のテレビ番組を録画したい。番組 i は時刻 $[s_i, t_i)$ に c_i 番目のチャンネルで放送されている。ある録画機で $[s, t)$ の番組を予約すると $[s-0.5, t)$ の間 **他のチャンネル** の録画には使えないので、全て録画するのに必要な録画機の最小個数を求めよ。$1 \leq N \leq 10^5$, $1 \leq c_i \leq C \leq 30$, $1 \leq s_i < t_i \leq 10^5$. ABC080 D - Recording

---

```
from collections import defaultdict
d = defaultdict(list)
imos = Imos(10 ** 5 + 1)

(N, C), *D = [[*map(int, o.split())] for o in open(0)]
for s, t, c in D:
    d[c] += (s, t),
for p in d.values():
    p.sort()
    ps, pt = p[0]
    for s, t in p[1:] + [[0,0]]:
        if pt == s:
            pt = t
        else:
            imos(ps - 1, pt - 1)
            ps, pt = s, t
print(max(imos.out()))
```

ソートし、同じチャンネルで番組が続くような場合は結合して一つの番組として扱うような実装にする。$[s-0.5, t)$ を占領するとは $[s-1, t-1]$ を占領するとして問題ないことがわかる。

例題 110. N 個の電球が数直線上にあり、電球 i は座標 (i) にある。電球 x の光の強さが d であるとき、その電球は $[x-d-0.5, x+d+0.5]$ を照らす。初めの光の強さは A_i で与えられる。今、「それぞれの電球について電球 i の強さを座標 (i) を照らしている電球の数に置き換える」という操作を K 回繰り返すとき、最終的な各電球の光の強さを求めよ。$1 \leq N, K \leq 2 \times 10^5$, $0 \leq A_i \leq N$　東京海上日動 2020 C - Lamps

---

```
from itertools import*
n, k, *A = map(int, open(0).read().split())
for _ in [0] * min(k, 41):
    imos = Imos(n)
    for i, a in enumerate(A):
        imos(i - a, i + a)
    A = imos.out()
print(*A)
```

K が十分大きいとき、$\log N$ 回程度で数列は収束し $A'_i = N$ のような配列になることがわかる。最悪ケースで試しても実際 41 回で終了するし、全ての要素が N になる時に打ち切れば十分である。計算量は $O(N \min(K, \log N))$.

例題 111. M 個の区間が与えられ、i 番目の区間では N 個の教室のうち $[s_i, t_i]$ の教室を掃除する。この中で、その区間を抜いても全教室を掃除できるようなものの総数とその番号 i を順に出力せよ。$1 \leq s_i \leq t_i \leq N \leq 3 \times 10^5$, $1 \leq M \leq 10^5$　ARC045 B - ドキドキデート大作戦高橋君

```
(n, m), *q = [[*map(int, o.split())] for o in open(0)]
imos = Imos(n + 2)
for s, t in q:
    imos(s, t)
```

```
*c, = accumulate([0] + [i > 1 for i in imos.out()])
ans = []
for e, (s, t) in enumerate(q, 1):
    if c[t + 1] - c[s] == t - s + 1:
        ans += e,
print(len(ans))
for i in ans:
    print(i)
```

---

imos 法の配列のそれぞれの要素について重複しているか $?(i > 1)$ を判別しそれで累積和 c を取る。区間 $[s,t]$ を抜いてもいいとは $c_{t+1} - c_s = t - s + 1$ となることである。

例題 112. N 個の遺跡と M 種類の宝石がある。遺跡 i では、得点 s_i 点と $[l_i, r_i]$ の宝石を獲得する。違う遺跡をいくつか探索したとき、得点の最大値はいくらか。ただし**全種類の宝石を集めてはならない**。$1 \leq N, M \leq 10^5$, $1 \leq s_i \leq 5000$, $1 \leq l_i \leq r_i \leq M$. ABC017 C - ハイスコア

```
(n, m), *D = [[*map(int, o.split())] for o in open(0)]
imos = Imos(m + 1)
for l, r, s in D:
    imos(l, r, s)
print(sum(d[2] for d in D) - min(imos.out()[1:]))
```

それぞれの宝石について、「その宝石を集めないとしたとき、**全ての遺跡を訪れたときからの得点減少分**」を imos 法で記録する。これは $[l,r]$ に s 加算する操作で良い。

例題 113. ある日の雨の降り始めた時刻 S_i と降り終えた時間 E_i の組を書いた降水メモが N 個ある。まず雨の降り始め・降り終わりをそれぞれ直前・直後の 5 分単位の時刻に丸める。この丸め終えたあとのメモにおいて、時間の重複があった場合それをつなげて改めて出力せよ。

$1 \leq N \leq 30000,\ 00:00 \leq S_i \leq E_i \leq 24:00$

```
def I(t: "hhmm") -> "min.":
    h, m = map(int, (t[:2], t[2:]))
    return h * 60 + m

def O(t: "min.") -> "hhmm":
    h, m = divmod(t, 60)
    return str(h).zfill(2) + str(m).zfill(2)

def round5(a, b):
    return a // 5 * 5, 0--b // 5 * 5

r, imos = range(24 * 60 + 1), Imos(24 * 60 + 1)
for i in range(int(input())):
    imos(*round5(*map(I, input().split("-"))))

D = imos.out()
S = [[*l] for k, l in groupby(r, key = lambda i: D[i] > 0) if k]
for s, *_, e in S:
    print(f"{O(s)}-{O(e)}")
```

いわゆる「やるだけ」問題だが、imos 法を使うとわかりやすくかつ高速に処理することができる。

5　データ構造

アルゴリズム 114. 素集合データ構造 *(Disjoint-set Data Structure)* あるいは **UnionFind 木** とは、互いに素な集合群に対するデータ構造であり、*Union*：*2* つの集合を *1* つに統合する　及び　*Find*：要素がどの集合に属しているか求める　の二種類の操作ができるものをいう。

```
1  class UnionFind():
2      def __init__(self, n):
3          self.n = n
4          self.parents = [-1] * n
5
6      def find(self, x):
7          if self.parents[x] < 0:
8              return x
9          else:
10             self.parents[x] = self.find(self.parents[x])
11             return self.parents[x]
12
13     def union(self, x, y):
14         x = self.find(x)
15         y = self.find(y)
16         if x == y:
17             return
18         if self.parents[x] > self.parents[y]:
19             x, y = y, x
20         self.parents[x] += self.parents[y]
21         self.parents[y] = x
22
23     def size(self, x):
24         return -self.parents[self.find(x)]
25
26     def same(self, x, y):
27         return self.find(x) == self.find(y)
28
29     def members(self, x):
30         root = self.find(x)
```

```
        return [i for i in range(self.n) if self.find(i) == root]

    def roots(self):
        return [i for i, x in enumerate(self.parents) if x < 0]

    def group_count(self):
        return len(self.roots())
```

具体的には Union(x, y) で x が含まれている集合と y が含まれている集合を結合し、Find(x) で x の根番号を求める。この実装では計算量を落とす (Union by Rank) ために根の親の値に要素数を負の値で格納している。サイズやランクをリストで保持する実装が必要なくなっている。さらに `find` では経路圧縮 (Path Compression) というテクニックを使っている。調べた要素の親を根に変更しつなぎ直すことで深い形状を解消している。これらのテクニックを落とすことでクエリ計算量を $O(\alpha(n))$ と実質定数にできる。

定義 115. アッカーマン関数 $A_k(n) : \mathbb{R} \to \mathbb{R}$ を以下のように再帰的に定義する。

$$A_k(0) = k + 1 \tag{135}$$

$$A_0(n) = A_1(n-1) \tag{136}$$

$$\text{otherwise}: \quad A_k(n) = A_{A_{k-1}(n)}(n-1) \tag{137}$$

定義 116. $\alpha(n)$ はアッカーマン関数の逆関数である。

$$\alpha(n) = A_k^{-1}(m) \tag{138}$$

$A_k(n)$ は $n \geq 4$ において非常に大きく増加する関数であり、その逆関数である $\alpha(n)$ は例えば $k = 2$ のとき $\alpha(2^{65536} - 3) = 4$. ここで $2^{65536} >> 10^{2\times 10^5}$ であるから、実質十分小さい定数とみなして良いといえる。クエリの計算量が $O(\alpha(n))$ である証明は非常に煩雑なので省略する。

定義 117. 集合 X のある部分集合族 $\mathscr{F}$ について、$\mathscr{F}$ のどの要素においても $F \in \mathscr{F}$ が真部分集合でないとき、F は $\mathscr{F}$ について極大な部分集合であるという。

定義 118. 任意の 2 頂点間に何らかの経路が存在するグラフのことを **連結グラフ** *(connected graph)* という。部分グラフのうち連結グラフであるようなものの集合族において極大で連結な部分グラフを、**連結成分** *(connected component)* という。

例題 119. 自己ループと二重辺を含まない N 頂点 M 辺の無向連結グラフが与えられる。i 番目の辺は頂点 a_i, b_i を結んでいる。それを取り除いたときグラフ全体が非連結になるような辺のことを橋と呼ぶ。M 本のうち橋の数を求めよ。$2 \leq N \leq 50$, $N-1 \leq M \leq \min(n(n-1)/2, 50)$, $1 \leq a_i < b_i \leq N$. ABC075 C - Bridge

---

```
(n, m), *q = [[*map(int, o.split())] for o in open(0)]
uf, c = UnionFind(n), 0
for i in range(m):
    for j, (a, b) in enumerate(q):
        if i == j: continue
        uf.union(a - 1, b - 1)
    c += UF.group_count() > 1
print(c)
```

制約が小さいので実際に構築し全ての頂点が同じ集合に属しているか確認すれば良い。

定義 120. グラフの頂点 v に接続する辺の本数を v の次数 *(degree)* という。また、閉路を含まないグラフを**森** *(forest)* といい、連結な森を **木** *(tree)* と呼ぶ。

定理 121. T が n 頂点の木ならば、T は $n-1$ 本の辺をもつ。

Proof. (証明) $n=1$ のとき明らか。以下 $n \geq 2$ における帰納法による。頂点数が m 以下まで仮定。T の頂点数を $m+1$ とする。T から次数 1 の頂点とその頂点に接続する辺を除去して得られる木を T' とすると、T' の頂点数は m. 帰納法の仮定より、T' の辺数は $m-1$. T' の構成方法より、T の辺数は T' より 1 つだけ大きい。したがって、T の辺数は m である。

□

例題 122. 自己ループと二重辺を含まない N 頂点 M 辺の無向グラフが与えられる。i 番目の辺は頂点 u_i, v_i を結んでいる。このグラフの連結成分のうち木であるようなものの個数を求めよ。$2 \leq N \leq 100$, $1 \leq M \leq N(N-1)/2$, $1 \leq u_i < v_i \leq N$. ARC037 B - バウムテスト

```python
(n, _), *q = [[*map(int, o.split())] for o in open(0)]
uf = UnionFind(n)
for u, v in q:
    uf.union(u - 1, v - 1)

edge = [0] * n
for u, _ in uv:
    edge[uf.find(u)] += 1

node = [0] * n
for i in range(n):
    node[uf.find(i)] += 1

print(sum(node[i] == edge[i] + 1 for i in range(n)))
```

例題 123. ある街には N 個の交差点と M 本の道路があり、i 番目の道路は交差点 a_i, b_i を結んでいる。どの交差点からも道路で別のどの交差点にも行けるようにするには最小で何本の道路を建設する必要があるか。$1 \leq N \leq 10^5$, $0 \leq M \leq 10^5$, $1 \leq a_i < b_i \leq N$　ARC032 B - 道路工事

```
(n, m), *q = [[*map(int, o.split())] for o in open(0)]
uf = UnionFind(n)
for a, b in q:
    uf.union(a - 1, b - 1)
print(uf.group_count() - 1)
```

例題 124. とある SNS では、A_i と B_i $(1 \leq i \leq M)$ の間に友達関係があり、C_i と D_i $(1 \leq i \leq K)$ の間にブロック関係がある。異なる人の組 (a, b)　であって、友達関係でもブロック関係でもなく、数列 c_i であって（ $c_0 = a \wedge c_L = b \wedge \forall i \in [0, L), c_i$ と c_{i+1} は友達関係である）を満たすようなものが存在しているとき、人 a, b は友達候補であるとする。それぞれの人に対し友達候補の数を求めよ。$2 \leq N \leq 10^5$, $0 \leq M, K \leq 10^5$.　ABC157 D - Friend Suggestions

```
(n, m, k), *q = [[*map(int, o.split())] for o in open(0)]
UF, non_cand = UnionFind(n), [1] * n
for a, b in q[:m]:
    a -= 1; b -= 1
    UF.union(a, b)
    non_cand[a] += 1; non_cand[b] += 1
for c, d in q[m:]:
    c -= 1; d -= 1
    if UF.same(c, d):
        non_cand[c] += 1; non_cand[d] += 1
print(*[UF.size(i) - non_cand[i] for i in range(n)])
```

友達関係で UnionFind を構築する。それぞれの人に対して属する連結成分のサイズを求め、非候補［自分（1 で初期化）、友達関係、同じ連結成分に属しブロック関係、の合計］を引けば良い。

定義 125. 木の末端の辺を葉という。グラフ $G(V,E)$ の頂点集合 V と辺集合の部分集合 $T \subseteq E$ からなる $G'(V,T)$ が木のとき、G' は G の**全域木** であるという。

例題 126. N 以下の正整数を並び替えた数列 $p_1,\ldots,p_N$ と N 以下の正整数の組 (x_i,y_i) が M 個与えられる。「任意の j を選び p_{x_j},p_{y_j} をスワップする」動作を任意の回数行えるとき、最終的に $p_i=i$ となる i の最大数を求めよ。$2 \leq N \leq 10^5$, $1 \leq M \leq 10^5$. ARC 097 D - Equals

---

```
(n, m), p, *q = [list(map(int, o.split())) for o in open(0)]
UF = UnionFind(n + 1)
for a, b in q:
    UF.union(a, b)
print(sum(UF.same(p[i], i + 1) for i in range(n)))
```

頂点集合 $V=[1,N]\cap\mathbb{Z}$, 辺集合 $E=\{(x_i,y_i) \mid 1 \leq i \leq M\}$ から成るグラフ $G(V,E)$ を考える。G の連結成分の集合 $G'=g_k$ に対して $\exists g_k \in G', (p_i,i) \in g_k^2$ ならば $p_i=i$ とできる（証明は以下）。また逆は明らかなので、UnionFind を構築しその数を数えれば良い。

Proof. 各 k に対して、全単射 $f : g_k \to \{p_i \mid i \in g_k\}$ であって

$$\exists g_k \in G', (p_i,i) \in g_k^2 \iff f(i)=i \tag{139}$$

と成るようなものが存在し、うまく全域木をとり葉 i に $f(i)$ をおく操作を繰り返せばよい、 □

例題 127. N 枚のカードについて、i 番目に書かれた数字は A_i で、1 か 2 のどちらかである。今、M 個の情報が与えられ、$A_{X_i}+A_{Y_i}+Z_i \in 2\mathbb{Z}$ である。少なくともいくつ A_i の値を知れば全ての A_i の値がわかるか。$2 \leq N \leq 10^5$, $1 \leq M \leq 10^5$, $1 \leq Z_i \leq 100$. ABC126 E - 1 or 2

```
(n, m), *q = [list(map(int, o.split())) for o in open(0)]
UF = UnionFind(n)
for x, y, _ in q:
    UF.union(x - 1, y - 1)
print(UF.group_count())
```

Z_i の偶奇によって、A_{X_i}, A_{Y_i} は偶奇が同じ / 異なるという情報を得る。数字は 1 か 2 のみだから、X_i, Y_i を繋ぐ UnionFind の連結成分数だけわかれば最終的に連鎖的に値は決定する。

例題 128. N 個の島および島 A_i, B_i を結ぶ全て異なる橋が M 本ある。はじめどの 2 島についても何らかの経路が存在するが、橋は 1 番目から順に全て崩落する。行き来できない島の組 $a < b$ の数を不便さと呼ぶとき、i 番目の橋が崩落した直後の不便さを答えよ。$2 \leq N \leq 10^5$, $1 \leq M \leq 10^5$　ABC120 D - Decayed Bridges

```
(n, m), *q = [map(int, o.split()) for o in open(0)]
UF = UnionFind(n)
conv = [n * (n - 1) // 2]
for a, b in q[::-1]:
    a -= 1; b -= 1
    if UF.same(a, b):
        conv += conv[-1],
    else:
        conv += conv[-1] - UF.size(a) * UF.size(b),
    UF.union(a, b)
assert conv[-1] == 0
for i in conv[-2::-1]: print(i)
```

クエリを先読みして逆順に処理し不便さを減らしていくようにすれば求めることができる。

例題 129. ある星には M 種類の言語がある。N 人の人がいて、それぞれ K_i 種類の言語 $L_{i1}, \ldots L_{iK_i}$ を話すことができる。ある 2 人はともに話すことの出来る言語が存在する、あるいはともにコミュニケーションをとることができる人が存在するとき、コミュニケーションをとることができる。全ての人が他の全ての人とコミュニケーションをとれるか判定せよ。$2 \leq N \leq 10^5$, $1 \leq M \leq 10^5$, $\sum K_i \leq 10^5$. CODE FESTIVAL '16 Final C - Interpretation

---

```
(n, m), *LL = [[*map(int, o.split())] for o in open(0)]
UF = UnionFind(n + m)
for e, (k, *L) in enumerate(LL):
    for l in L:
        UF.union(e, l + n - 1)
print(len([*filter(lambda i: i < n, UF.roots())]) == 1 and "YES" or "NO")
```

人を $[0, N)$ 、言語を $[N, N+M)$ で管理し、それぞれの人について話せる言語に辺を貼り、連結成分の数を数えれば良い。ただし、番号が N 未満の根に絞らないと言語が残っていても判定されてしまうので注意。

例題 130. ある国には N 個の都市と M 本の道路があり、i 本目の道路は都市 a_i, b_i を結び y_i 年に作られた。Q 人の住民について、j 人目の住民は都市 v_j に住んでおり w_j 年以前に作られた道路を通らない。それぞれの住民に対して行くことができる都市の数を求めよ。$1 \leq N, Q \leq 10^5$, $0 \leq N \leq 10^5$, $1 \leq y_i \leq 2 \times 10^5$, $0 \leq w_i \leq 2 \times 10^5$. ABC 040 D - 道路の老朽化対策について

```
(n, m), *d = [[*map(int, o.split())] for o in open(0)]
I, q, J = d[:m], int(*d[m]), d[m + 1:]
data = []
for a, b, y in I:
```

```
    data += [-y, 1, a, b],
for e, (v, w) in enumerate(J):
    data += [-w, 0, e, v],
data.sort()

UF, ans = UnionFind(n + 1), [0] * q
for _, F, c, d in data:
    if F:
        UF.union(c, d)
    else:
        ans[c] = UF.size(d)

for s in ans:
    print(s)
```

年の大きい方から処理していけばよい。なお、同じ年の中では住民クエリを先に処理する必要があることに注意。

例題 131. 先頭が 0 でない N 桁の整数を 2 回メモしたが、一部字が汚くアルファベットに見えている。同じアルファベットは同じ数字を表すとするとき（逆は必ずしも成り立たない)、最大で何種類の数が考えられるか。$1 \leq N \leq 18$.　ARC027 B - 大事な数なので Z 回書きま L た。

```
# print(ord("A")) -> 65
code = lambda char: ord(char) - 55 if char.isalpha() else int(char)

n = int(input())
*s1, = map(code, input())
*s2, = map(code, input())

UF = Un1onF1nd(36)
for s, t in zip(s1, s2):
    UF.un1on(s, t)
```

```
11
12 ans = 1
13 used = set()
14 for e, s in enumerate(s1):
15     if any(UF.same(i, s) for i in range(10)) or (t := UF.find(s)) in used:
16         continue
17     used |= {t}
18     ans *= 10 - (e == 0)
19 print(ans)
```

---

s_{1i}, s_{2i} を結んで得られるグラフの連結成分のうち数字が含まれているものは確定する。そうでないとき、10 通りの可能性がある。ただし、先頭には 0 は使えないので、それを考慮して前から見ていきすでにチェックしたルートはパスするようにすればよい。

例題 132. N 頂点 M 辺の森が与えられる。頂点は **0−indexed** で、辺は頂点 x_i, y_i を結ぶ。各頂点には a_i という値が決まっている。今、この森にいくつかの辺を張って木にすることを考える。ただし、頂点 i, j を結ぶときコスト $a_i + a_j$ がかかり、さらに頂点は**一度しか選択できない**。コストを最小化せよ。木にできないときは `'Impossible'` と出力せよ。$1 \leq N \leq 10^5$, $0 \leq M \leq N-1$, $1 \leq a_i \leq 10^9$. APC001 D - Forest

---

```
1 (n, m), a, *q = [[*map(int, o.split())] for o in open(0)]
2 UF = UnionFind(n)
3 k = (n - m - 1) * 2
4 if k > n: exit(print("Impossible"))
5 if not k: exit(print(0))
6
7 for c, f in q:
8     UF.union(c, f)
9
10 from collections import*
```

```
d = defaultdict(list)
for e, i in enumerate(a):
    d[UF.find(e)] += i,

ans, b = 0, []
for r, p in d.items():
    p.sort()
    ans += p.pop(0)
    b += p
    k -= 1
b.sort()
print(ans + sum(b[:k]))
```

$N-1$ 辺にするのが目標である。よって、はる辺数は $N-M-1$ 本。必要な次数の合計は $K=2(N-M-1)$ 本だが、同じ頂点には二度とはれないことから少なくとも K 頂点が必要になる。$K>N$ なら不可能で、そうでなければ構築可能なことがわかる。さらに、それぞれの連結成分に対して少なくとも 1 つの頂点を選ぶ必要があるが、その残りはどの連結成分から選んでも構築方法が存在するとわかる。よって、まず全ての連結成分からコスト最小のものを選び、残りは全体の小さい順から選んでいけばよい。

定義 133. グラフ $G(V,E)$ において、頂点集合 V を 2 つの部分集合 V_1,V_2 に分割して各集合内の頂点同士の間には辺が無いようにできるとき、G は **2 部グラフ** *(Bipartite Graph)* であるといい、$G(V_1+V_2,E)$ などとかく。またこのとき、任意の $(u,v)\in V_1V_2$ について辺 (u,v) が存在するとき、G は **完全 2 部グラフ** *(Complete Bipartite Graph)* であるという。

系 134. 完全 2 部グラフ $G(V_1+V_2,E)$ において、辺の数は $|V_1||V_2|$ である。

例題 135. xy 座標平面上の第一象限に点 (x_i, y_i) が重複なく N 個ある。$(a,b),(a,d),(c,b),(c,d)$ のうち 3 箇所に点があるような整数 $a \neq c, b \neq d$ を選び、残りの 1 箇所に点を追加するという操作を行える限り繰り返す。この操作は有限回しか行えないことが証明できるが、操作回数の最大値はいくらか。$1 \leq N, x_i, y_i \leq 10^5$.　ABC131 F - Must Be Rectangular!

```
(n,), *q = [[*map(int, o.split())] for o in open(0)]
MAX = 10 ** 5 + 1
UF = UnionFind(2 * MAX)
for x, y in q:
    UF.union(x, y + MAX)
cnt_x = [0] * (2 * MAX)
cnt_y = cnt_x[:]
for i in range(MAX):
    cnt_x[UF.find(i)] += 1
    cnt_y[UF.find(i + MAX)] += 1
print(sum(X * Y for X, Y in zip(cnt_x, cnt_y)) - n)
```

まず、2 部グラフ $G(X+Y, E)$ を考えていくと、連結成分ごとに分解できる。ここで X, Y はそれぞれ x, y 座標の集合で、E は与えられた点の座標組である（座標平面を 2 部グラフとみなすのは典型）。そこで、この操作はそれぞれの連結成分について完全 2 部グラフになるまで行うことができるとわかる。よって、答えは $\sum |X||Y| - N$ である。

例題 136. N 個の駐車スペースと駐車スペース u_i, v_i を結ぶ M 本の道がある駐車場に、N 人の人が順にやってくる。i 番目の人は、駐車場の入り口から空いている駐車場と道を通って駐車スペース i に車を駐めようとするが、たどり着くことができないときは帰ってしまう。駐車スペース S が駐車場の入り口と繋がっているとき、駐車場に駐めることができる人の番号を順に出力せよ。$1 \leq N, M \leq 2 \times 10^5$.　ARC056 B - 駐車場

```
(n, m, s), *q = [[*map(int, o.split())] for o in open(0)]
from collections import*
d = defaultdict(list)
for u, v in q:
    d[min(u, v, s)] += (u, v),
UF = UnionFind(n + 2)
UF.union(s, n + 1) # Entrance
ans = []
for i in range(s, 0, -1):
    for u, v in d[i]:
        UF.union(u, v)
    if UF.same(i, n + 1):
        ans += i,
for i in ans[::-1]:
    print(i)
```

まず、S より番号が大きい人が駐めることはない。それ以外のとき、それぞれの人について自分の番号より大きい駐車場同士を結ぶ道しか通れないとしてよい。よって、番号 S の人から逆順で見ていき、道をそれぞれ貼っていけばいい。そして作られたグラフにおいて入口と駐車スペースが同じ連結成分にあるとき駐めることができる。なお、駐車されなかったときは入口からたどり着く道がないときなので考慮に入れなくてよい。

例題 137. N 個の都市がある。K 本の道路と L 本の鉄道が都市の間に伸びている。道路 i は都市 p_i, q_i を結び、鉄道 j は都市 r_i, s_i を結ぶ。それぞれの都市について、自分を含め道路と鉄道両方で到達できる都市の数を求めよ。$2 \leq N \leq 2 \times 10^5$, $1 \leq K, L \leq 10^5$. ABC049 D - 連結

```
(n, k, l), *d = [[*map(int, o.split())] for o in open(0)]
R, T = UnionFind(n), UnionFind(n)
```

```
for p, q in d[:k]:
    R.union(p - 1, q - 1)
for r, s in d[k:]:
    T.union(r - 1, s - 1)
C = []
for i in range(n):
    C += (R.find(i), T.find(i)),
from collections import*
c = Counter(C)
print(*[c[g] for g in C])
```

道路と鉄道それぞれ UnionFind を構築したあと、それぞれの都市について道路と鉄道について根を求めて、実際に記録していく。それぞれの都市について道路と鉄道両方について根が同じような都市の数を求めればよい。

例題 138. N 個の街があり、最初道はない。今、整数 w_i, x_i, y_i, z_i からなる Q 個のクエリが時系列順に与えられる。$w_i = 1$ のとき、街 x_i, y_i 間に長さ z_i メートルの道路を敷設することを表し、$w_i = 2, z_i = 1$ のときその時点で街 x_i, y_i を合計 **偶数** メートルで移動できるか質問することを表す。それ以外のクエリは与えられないので、後者のクエリに回答せよ。$1 \leq N, Q, z_i \leq 10^5$.

ARC036 D - 偶数メートル

```
(n, _), *q = [[*map(int, o.split())] for o in open(0)]
UF = UnionFind(n * 2)
for w, x, y, z in q:
    x -= 1; y -= 1
    if w == 1:
        if z%2:
            UF.union(x, y + n)
            UF.union(y, x + n)
        else:
```

```
            UF.union(x, y)
            UF.union(x + n, y + n)
    else:
        print("YES" if UF.same(x, y) else "NO")
```

$2N$ 個の点を用意し、敷設クエリで各 x_i, y_i について、z_i が奇数の時 x_i, y_i' と x_i', y_i を、偶数のとき x_i, y_i と x_i', y_i' をそれぞれ結べば、判定クエリでは x_i, y_i が同じ連結成分にあるか判定すればよい。

アルゴリズム 139. 重み付き UnionFind *(WeightedUnionFind)* は、通常の UnionFind に加え、辺の距離を保持するデータ構造。

```
class WeightedUnionFind():
    def __init__(self, n):
        self.n = n
        self.parents = [-1] * n
        self.weight = [0] * n

    def find(self, x):
        if self.parents[x] < 0:
            return x
        else:
            px = self.find(self.parents[x])
            self.weight[x] += self.weight[self.parents[x]]
            self.parents[x] = px
            return px

    def union(self, x, y, w):
        w += self.weight[x] - self.weight[y]
        x = self.find(x)
        y = self.find(y)
        if x == y:
```

```
            return
        if self.parents[x] > self.parents[y]:
            x, y, w = y, x, -w
        self.parents[x] += self.parents[y]
        self.parents[y] = x
        self.weight[y] = w
        return

    def weig(self, x):
        self.find(x)
        return self.weight[x]

    def diff(self, x, y):
        return self.weigh(y) - self.weigh(x)

    # folding; cf.) UnionFind
```

実際の実装では、Find での経路圧縮の際に差分の重みを累積和的に更新している。Union では根同士をつなぐので w は修正が必要。また、swap 操作の際には w の符号は反転する。Weig は経路圧縮してから重みを返す。重み差分を返す Diff はそのままの実装。

例題 140. N 人の人がいて、それぞれ数直線上の座標 (x_i) に立っている。いま、人 R_i は人 L_i より距離 D_i だけ右にいるという M 個の情報が与えられる。これら全てに矛盾しないような座標の組が存在するか答えよ。$1 \leq N \leq 10^5$, $0 \leq M \leq 2 \times 10^5$, $0 \leq D_i \leq 10^4$. ABC087 D - People on a Line

---

```
(n, m), *q = [[*map(int, o.split())] for o in open(0)]
UF = WeightedUnionFind(n + 1)
for l, r, d in q:
    if UF.same(l, r):
        if d != abs(UF.diff(l, r)):
```

```
            print("No"); exit()
    else:
        UF.union(l, r, d)
print("Yes")
```

実際に矛盾がないか調べていけばよい。矛盾がある場合前から見ていっても必ず誤りの判定がされるのでこれで十分。

例題 141. 複数テストケースが与えられる。0 は入力の終わりを示す。それぞれのテストケースについて、N 個の情報が与えられ、それぞれ `"1 A = 10^x B"` という形をしており、単位 A と単位 B の大きさの比較である。情報が矛盾しないか判定せよ。単位は空白を含まないアルファベット小文字 1〜16 文字からなる。$1 \leq N \leq 100$, $-100 \leq x \leq 100$. AOJ2207 - 無矛盾な単位系

```
while 1:
    n = int(input())
    if n == 0:
        exit()
    Q, d = [], set()
    while n:
        _, a, _, p, b = input().split()
        Q += (a, b, int(p[3:])),
        d |= {a, b}
        n -= 1
    D = {c : e for e, c in enumerate(d)}
    UF = WeightedUnionFind(len(d))
    for a, b, p in Q:
        a, b = D[a], D[b]
        if UF.same(a, b):
            if p != UF.diff(a, b):
                print("No"); break
        else:
            UF.union(a, b, p)
```

```
    else:
        print("Yes")
```

対数で考えることを除けば前の問題とほぼ同じ。文字列の処理が少し面倒...

定義 142. 変更される際に変更前のバージョンを常に保持するデータ構造を、**永続データ構造** *(Persistent Data Structure)* という。全バージョンにアクセス可能で、最新版だけを変更可能なデータ構造を**半永続的 / 部分永続的** *(partially persistent)* という。全バージョンにアクセスも更新も可能なデータ構造を**全永続的 / 完全永続的** *(fully persistent)* という。最新版以外の 2 つのバージョンから新たなバージョンを作成・マージする操作があるなら、そのデータ構造を融合永続的 *(confluently persistent)* という。永続的でないデータ構造は短命 *(ephemeral)* であるという。

アルゴリズム 143. 部分永続 UnionFind *(Partially Persistent UnionFind)* は、部分永続化された *UnionFind* である。

```
INF = 10**9
from biesct import*
class PPUnionFind():
    def __init__(self, n):
        self.n = n
        self.parents = [-1] * n
        self.time = [INF] * n
        self.number_time = [[0] for _ in [None] * n]
        self.number_dots = [[1] for _ in [None] * n]

    def find(self, x, t):
        while self.time[x] <= t:
            x = self.parents[x]
```

```
14          return x
15
16      def union(self, x, y, t):
17          x = self.find(x, t)
18          y = self.find(y, t)
19          if x == y:
20              return 0
21          if self.parents[x] > self.parents[y]:
22              x, y = y, x
23          self.parents[x] += self.parents[y]
24          self.parents[y] = x
25          self.number_time[x] += [t]
26          self.number_dots[x] += [-self.parents[x]]
27          self.time[y] = t
28          return t
29
30      def size(self, x, t):
31          x = self.find(x, t)
32          return self.number_dots[x][bisect_left(self.number_time[x], t) - 1]
33
34      def whensame(self, x, y, t = 0):
35          if x == y:
36              return t
37          if self.time[x] == self.time[y] == INF:
38              return -1
39          if self.time[x] > self.time[y]:
40              x, y = y, x
41          return self.whensame(self.parents[x], y, self.time[x])
42
43      def same(x, y, t):
44          return self.find(x, t) == self.find(y, t)
```

少し複雑になってしまったが、実装はさほど変わらない。基本的に行える操作は通常の UnionFind に「時刻 t で」という条件を付け足しただけのもの。あまりに文献が少ないので自分で書いてみたが、「いつ同じ連結成分になるか」を示す whensame() が特にとても重く時間があれば改修したい...

例題 144. 集合 $\{1\}, \{2\}, \ldots, \{N\}$ が与えられる。時刻 i に要素 a_i を持つ集合と b_i を持つ集合を結合するという操作を M 回行う。クエリが Q 個与えられるので、j 回目のクエリでは要素 x_j と要素 y_j が同じ集合にはじめて属するのは何回目の操作後かを回答せよ。ただし最終的に同じ集合に属さない場合 -1 を出力せよ。$2 \leq N \leq 10^5$, $0 \leq M \leq 10^5$, $1 \leq Q \leq 10^5$. CODE THANKS FESTIVAL 2017 H - Union Sets

---

```python
# PyPy 564ms
(n, m), *q = [[*map(int, o.split())] for o in open(0)]
UF = PPUnionFind(n + 1)
for t, (a, b) in enumerate(q[:m], 1):
    UF.union(a, b, t)

for x, y in q[m + 1:]:
    print(UF.whensame(x, y))
```

まさに操作を行うだけの問題。

例題 145. N 頂点 M 辺の連結無向グラフがあり、i 番目の辺は頂点 a_i, b_i を結んでいる。このグラフの上で、Q 組の兄弟がスタンプラリーをする。j 組目の兄弟について、最初、兄、弟はそれぞれ頂点 x_j, y_j におり、兄弟でちょうど z_j 個の頂点を訪れる。同じ頂点を兄弟ともに訪れても 1 つと数える。兄または弟が通った辺の番号の最大値がスコアになる。それぞれの兄弟についてスコアの最小値を求めよ。$3 \leq N \leq 10^5$, $N-1 \leq M \leq 10^5$, $x_j < y_j$. AGC002 D - Stamp Rally

---

```python
# PyPy 1607ms
UF = PPUnionFind(n + 1)

(n, m), *q = [[*map(int, o.split())] for o in open(0)]
```

```
def size2(x, y, t):
    a = UF.size(x, t)
    if not UF.same(x, y, t):
        a += UF.size(y, t)
    return a

for t, (a, b) in enumerate(q[:m], 1):
    UF.union(a, b, t)

for x, y, z in q[m+1:]:
    print(meg_bisect(0, m, lambda m: size2(x, y, m) >= z))
```

時刻 i に辺 i を順に足していくと考える。すると、兄弟で訪れることのできる頂点がはじめて z_i 個以上になったときの時刻が答えで、二分法を使えばよい。二分探索の実装で `//` を `>>` にしないと TLE するので注意。また、兄弟がいる連結成分が同じかそうでないかで場合わけが必要。

アルゴリズム 146. 優先度付きキュー *(Priority queue)* は最も優先度が高いものの取り出し、優先度の情報を持ったままの要素の追加を共に $O(\log N)$ で行えるようなデータ構造をいう。*Python* では二分探索木 `heapq` として最小値についてのみサポートされている。

例題 147. N 個の買い物をすることを考える。i 番目に買う品物は A_i 円である。M 枚の割引券があり、X 円の品物を Y 枚の割引券を使って買うと $\lfloor X/2^Y \rfloor$ 円で買うことができる。うまく割引券を割り振ると最小何円で全ての品物を買えるか。$1 \leq N, M \leq 10^5$, $1 \leq A_i \leq 10^9$. ABC141 D - Powerful Discount Tickets

---

```
from heapq import*
n, m, *a = eval(",-".join(open(0).read().split()))
```

```
a.sort()
while m:
    heappush(a, 0--heappop(a)//2)
    m += 1
print(-sum(a))
```

まず、heapq は最小値しか扱えないので符号を逆にして考える。また、空でないリストをヒープ化させるには sort か heapify が必要。0--b//2 は切り上げ演算（正負逆転で切り下げになる）。M 個の割引券について、「最大の物を選んで 2 で割る」という操作を繰り返せばいいことがわかる（正確な証明は煩雑なので省略するが、bit で考えると直感的）。

例題 148. N 件の日雇いアルバイトがあり、i 件目の日雇いアルバイトを請けて働くと、その A_i 日後に報酬 B_i が得られる。今日から M 日後まで 1 日 1 つまで選んで働くことができる。一度選んだものは選択できないとき、最終的な報酬を最大化せよ。$1 \leq N, M, A_i, \leq 10^5$, $1 \leq B_i \leq 10^4$.ABC137 D - Summer Vacation

---

```
(N, M), *t = [map(int, s.split()) for s in open(0)]
q, z = [], 0
v = [[] for _ in [None] * 10**5]
for a, b in t:
    v[a - 1] += b,
for i in v[:M]:
    for j in i:
        heappush(q, -j)
    z += -heappop(q) if q else 0
print(z)
```

M 日後から遡って考えていくと、ヒープキューが使えそうとわかる。

例題 149. 工場に N 個の機械があり、最初 a_i 秒でプレゼントを一個作る。しかし、機械はそれぞれ 1 回使うたびに劣化し、追加で b_i 秒かかるようになる。複数の機械を同時に動かすことはできないので、プレゼント K 個を作るのにかかる最小時間を求めよ。$1 \leq N, K \leq 10^5$, $1 \leq a_i \leq 10^9$, $0 \leq b_i \leq 10^9$　CODE THANKS FESTIVAL 2017 C - Factory

---

```
from heapq import*
(n, k), *d = [[*map(int, o.split())] for o in open(0)]
q = []
for a, b in d:
    heappush(q, (a, b))
ans = 0
for _ in [0] * k:
    a, b = heappop(q)
    ans += a
    heappush(q, (a + b, b))
print(ans)
```

タプルを使って b_i の情報を保持すると楽。

例題 150. N 頂点の木がある。今、頂点 1 を選び、それ以降今まで選ばれた頂点と辺で結ばれている頂点のうち、まだ選ばれていない頂点を 1 つ選ぶという操作を可能な限り繰り返す。選ばれた頂点の番号を順に並べた数列として構成可能な辞書順最小のものを出力せよ。$2 \leq N \leq 10^5$.

Indeed なう 予選 B C - 木

```
(n,), *e = [[*map(int, o.split())] for o in open(0)]
from collections import*
d = defaultdict(list)
used = defaultdict(lambda: False)
```

```
for a, b in e:
    d[a] += b,
    d[b] += a,
from heapq import*
c, ans = [1], []
for _ in [None] * n:
    p = heappop(c)
    ans += p,
    used[p] = True
    for i in d[p]:
        if not used[i]:
            heappush(c, i)
print(*ans)
```

こちらも実装するだけ。既に使われたかどうかを探索すると一回あたり $O(|c|)$ かかってしまうので専用の連想配列を持った。

例題 151. $1, 2, 3$ の形をしたケーキがそれぞれ X, Y, Z 種類あり、それぞれの美味しさがそれぞれ数列 A_i, B_i, C_i として与えられる。これらから $1, 2, 3$ の形ひとつずつ選ぶ方法は XYZ 通りあるが、その美味しさの上位 K 個を出力せよ。$1 \leq X, Y, X \leq 10^3$, $1 \leq K \leq \min(3000, XYZ)$, $1 \leq A_i, B_i, C_i \leq 10^{10}$　　ABC123 D - Cake 123

```
(x, y, z, k), a, b, c = [[*map(int, o.split())] for o in open(0)]
a.sort(); b.sort(); c.sort()

h = [(-(a[-1] + b[-1] + c[-1]), -1, -1, -1)]
s = set(h)

from heapq import*
def push(i, j, k):
    global s, h
    try:
```

```
        t = (- (a[i] + b[j] + c[k]), i, j, k)
    except IndexError:
        return
    if t not in s:
        heappush(h, t)
        s |= {t}

for _ in [0] * k:
    p, i, j, k = heappop(h)
    print(-p)
    push(i, j, k - 1)
    push(i, j - 1, k)
    push(i - 1, j, k)
```

アルゴリズム 152. フェニック木 または **BIT** *(Binary Indexed Tree)* とは、要素 i への値 x の加算 / 区間和 $[0, i)$ の 2 つを高速に計算できるものをいう。1 – indexed で考えると、インデックス i ではその $2^{l_i - 1}$ 分の区間和の情報を持つ。ただしここで l_i は i の *LSB (Least Significant Bit)* であり 2 進数表現を右から見ていってはじめて 1 が現れる桁数目をさす。

```
class Bit:
    # Binary Indexed Tree
    def __init__(self, n):
        self.size = n
        self.tree = [0] * (n + 1)

    def __iter__(self):
        psum = 0
        for i in range(self.size):
            csum = self.sum(i + 1)
            yield csum - psum
            psum = csum
        raise StopIteration()

    def __str__(self): # O(nlogn)
```

```
16          return str(list(self))
17
18      def sum(self, i):
19          # Σ [0, i)
20          if not (0 <= i <= self.size): raise ValueError("error!")
21          s = 0
22          while i > 0:
23              s += self.tree[i]
24              i -= i & -i
25          return s
26
27      def add(self, i, x):
28          if not (0 <= i < self.size): raise ValueError("error!")
29          i += 1
30          while i <= self.size:
31              self.tree[i] += x
32              i += i & -i
33
34      def __getitem__(self, key):
35          if not (0 <= key < self.size): raise IndexError("error!")
36          return self.sum(key + 1) - self.sum(key)
37
38      def __setitem__(self, key, value):
39          if not (0 <= key < self.size): raise IndexError("error!")
40          self.add(key, value - self[key])
```

実装上、区間和は LSB の **減算** とみなすことができる（0 になるまで続ける）。また、値の更新は LSB の **加算** とみなすことができる（n になるまで続ける）。

補題 153. *alg.152* において、LSB は `k & -k` で求められる。

Proof. (証明) k の LSB を x とすると、2 進数表現での右から $1 \ldots x-1$ 番目は 0 で右から x 番目は 1. よって、$\sim k$ の 2 進数表現での右から $1 \ldots x-1$ 番目は 1 で右から x 番目は 0. よって $-k = \sim k + 1$ から繰り上がりを計算しながら $-k$ を求めると 2 進数表現での右から $1 \ldots x-1$ 番

目は 0 で右から x 番目は 1. さらに x より左側について k と $-k$ のビットは全て異なる。よって、`k & -k` で得られるビットは一意に x 番目に定まる。 □

例題 154. 長さ N の英小文字から成る文字列 S が与えられるので、*type1:* S の i 文字を英小文字 c に変更する、*type2:* S の l 文字目から r 文字目までに現れる文字の種類数を答えなさい。$1 \leq N \leq 5 \times 10^5$, $1 \leq Q \leq 2 \times 10^4$. ABC157 E - Simple String Queries

---

```python
# PyPy 600ms
(n,), (s,), _, *q = map(str.split, open(0))
*s, = s
BIT = [Bit(int(n)) for _ in [0] * 26]
for e, t in enumerate(s):
    BIT[ord(t) - 97].add(e, 1)
for a, b, c in q:
    if a == "1":
        i = int(b)
        BIT[ord(s[i - 1]) - 97].add(i - 1, -1)
        s[i - 1] = c
        BIT[ord(c) - 97].add(i - 1, 1)
    else:
        l, r = int(b), int(c)
        print(sum(BIT[i].sum(r) - BIT[i].sum(l - 1) > 0
              for i in range(26)))
```

実際に 26 本 BIT を持ってアルファベットごとに処理する。

例題 155. N 個の色玉が一列に並んでいて、色は数列 c_i で与えられる。クエリが Q 個与えられるので $[l, r]$ での色の種類数を答えよ。$1 \leq N, Q \leq 5 \times 10^5$. ABC174 F - Range Set Query

```python
(n, q), c, *d = [[*map(int, o.split())] for o in open(0)]

d = sorted(enumerate(d), key = lambda x: x[1][1])

B = Bit(n)
ans = [-1] * q
last = [-1] * (n + 1)
ind = 0
for i, (l, r) in d:
    for j in range(ind, r):
        if last[c[j]] != -1:
            B.add(last[c[j]], -1)
        B.add(j, 1)
        last[c[j]] = j
    ind = r
    ans[i] = B.sum(r) - B.sum(l - 1)
for a in ans:
    print(a)
```

区間を終端 r でソートして「最後に出現した位置」を動かしていくイメージ。

アルゴリズム 156. BIT 上でいもす法を行うことができる。

```python
class BitImos:
    """
    Query : O(log n)
    """
    def __init__(self, n):
        self.bit = Bit(n + 1)

    def add(self, s, t, x):
        # [s, t) += x
```

```
        self.bit.add(s, x)
        self.bit.add(t, -x)

    def get(self, i):
        return self[i]

    def __getitem__(self, key):
        return self.bit.sum(key + 1)
```

アルゴリズム 157. BIT を 2 本持つことで区間加算クエリにも対応することができる。

```
class Bit2:
    def __init__(self, n):
        self.bit0 = Bit(n)
        self.bit1 = Bit(n)

    def add(self, l, r, x):
        # [l, r) += x
        self.bit0.add(l, -x * (l - 1))
        self.bit1.add(l, x)
        self.bit0.add(r, x * (r - 1))
        self.bit1.add(r, -x)

    def sum(self, l, r):
        res = 0
        res += self.bit0.sum(r) + self.bit1.sum(r) * (r - 1)
        res -= self.bit0.sum(l) + self.bit1.sum(l) * (l - 1)
        return res
```

定義 158. ある列の**転倒数** *(inversion number)* とは、その整列性の測度である。厳密には数列 A の転倒数は $\mathrm{inv}(A) = \{(A_i, A_j) \mid i < j, A_i > A_j\}$ として定められる。

アルゴリズム 159. BIT を用いて転倒数を求めるアルゴリズムを愚直 $O(n^2)$ から $O(\max(A) + n \log n)$ に改善することができる。$(n = |A|)$

```python
def mergecount(A: list):
    bit = Bit(max(A) + 1)
    cnt = 0
    for i, a in enumerate(A):
        cnt += i - bit.sum(a + 1)
        bit.add(a, 1)
    return cnt
```

自分より大きい要素の数をそれぞれ求めている。なお、この計算量は座標圧縮というテクニックを使うことで単純 $O(n \log n)$ に落とすことができる。

アルゴリズム 160. 座標圧縮 とは値の大小関係を維持したまま最小限の整数に圧縮する操作。

```python
def compress(a: list) -> list:
    d = {v : i for i, v in enumerate(sorted(set(a)))}
    return [d[i] for i in a]
```

定義 161. 隣接要素の比較交換を繰り返すソートを **バブルソート** *(bubble sort)* という。

系 162. ある列における転倒数はその列をバブルソートするのに発生する swap 回数に等しい。

Proof. (証明) 1 回の swap で転倒数は 1 減少し、ソートが完了すると 0 となるので自明。 □

例題 163. N 人の人が一列に並んでいる。この列を左から見た時、ある人について自分より左に並んでいる全ての人よりも身長 H_i が高い場合に限り見ることが出来るので、全員が見えるように隣り合う人と入れ替わることを繰り返し並び替える時のコストの合計を求めよ。ただし 1 回の入れ替わりでのコストは 2 人のうち低い方の身長となる。また、全員が見えるような並び替えが不可能な時は -1 と出力せよ。$1 \leq N \leq 10^5, 1 \leq H_i \leq 10^8$. Indeed なう Final-B E - Line up!

```python
def mergecount_(A: list):
    bit = Bit(n + 1)
    cnt = 0
    for i, (h, d) in enumerate(zip(A, compress(A))):
        cnt += h * (i - bit.sum(d + 1))
        bit.add(d, 1)
    return cnt

n, *h = map(int, open(0).read().split())
print(-1 if len(set(h)) != n else mergecount_(h))
```

重複する要素がある時は不可能。それ以外の時、座標圧縮をした上で転倒数のアルゴリズムを改造すれば求められる。

アルゴリズム 164. BIT 上での二分探索

```python
class Bit:
    def __init__(self, a):
        if hasattr(a, "__iter__"):
            le = len(a)
            self.n = 1 << le.bit_length()
            self.values = values = [0] * (self.n + 1)
            values[1:le+1] = a[:]
```

```
8            for i in range(1, self.n):
9                values[i + (i & -i)] += values[i]
10       elif isinstance(a, int):
11           self.n = 1 << a.bit_length()
12           self.values = [0] * (self.n + 1)
13       else:
14           raise TypeError
15
16   def add(self, i, val):
17       n, values = self.n, self.values
18       while i <= n:
19           values[i] += val
20           i += i & -i
21
22   def sum(self, i): # (0, i]
23       values = self.values
24       res = 0
25       while i > 0:
26           res += values[i]
27           i -= i & -i
28       return res
29
30   def bisect_left(self, v):
31       n, values = self.n, self.values
32       if v > values[n]:
33           return None
34       i, step = 0, n >> 1
35       while step:
36           if values[i + step] < v:
37               i += step
38               v -= values[i]
39           step >>= 1
40       return i + 1
```

この実装では `n` を `le` を超える最小の 2 冪（`1 << le.bit_length()`）にし`len(values)` を 2 冪 +1 にすることで二分探索の条件を減らしている。また`values` の 0 番目は使っていない。`bisect_left(v)` で `sum(i)` が `v` 以上になる最小の `i` を求める。

例題 165. 空集合 S があるので、Q 個のクエリを処理しなさい。$T = 1$ のとき整数 X を S に追加し、$T = 2$ のとき S で X 番目に小さい数を答えそれを削除しなさい。$1 \leq Q, X \leq 2 \times 10^5$

ARC033 C - データ構造

```
_, *Q = [[*map(int, o.split())] for o in open(0)]

bit = Bit(200000)

for t, x in Q:
    if t == 1:
        bit.add(x, 1)
    else:
        key = bit.bisect_left(x)
        print(key)
        bit.add(key, -1)
```

制約から実際に X に対する BIT を構築して操作するだけ。

定義 166. 二分木 *(binary tree)* は、データ構造のひとつで、根付き木構造の中であるノードが持つ子の数が高々 2 であるものをいう。全ての葉が同じ「深さ」を持つ二分木を完全二分木 *(perfect binary tree)* という。

定義 167. 集合 S における閉じた演算 $*$ が与えられ、これが結合律をみたしかつ単位元 e を持つとき、組 $(S, *, e)$ あるいは代数系 $\langle S, * \rangle$ は **モノイド** *(monoid)* であるという。

アルゴリズム 168. セグメント木 (Segment Tree) は、完全二分木の一つ。

```python
class SegmentTree(object):
    __slots__ = ["elem_size", "tree", "default", "op", "real_size"]

    def __init__(self, a, default = float("inf"), op: "func" = min):
        self.default = default
        self.op = op
        if hasattr(a, "__iter__"):
            self.real_size = len(a)
            self.elem_size = elem_size = 1 << (self.real_size-1).bit_length()
            self.tree = tree = [default] * (elem_size * 2)
            tree[elem_size : elem_size + self.real_size] = a
            for i in range(elem_size - 1, 0, -1):
                tree[i] = op(tree[i << 1], tree[(i << 1) + 1])
        elif isinstance(a, int):
            self.real_size = a
            self.elem_size = elem_size = 1 << (self.real_size-1).bit_length()
            self.tree = [default] * (elem_size * 2)
        else:
            raise TypeError

    def get_value(self, x: int, y: int) -> int: # [x, y)
        l, r = x + self.elem_size, y + self.elem_size
        tree, result, op = self.tree, self.default, self.op
        while l < r:
            if l & 1:
                result = op(tree[l], result)
                l += 1
            if r & 1:
                r -= 1
                result = op(tree[r], result)
            l, r = l >> 1, r >> 1
        return result

    def __setitem__(self, i: int, value: int) -> None:
        k = self.elem_size + i
        op, tree = self.op, self.tree
        tree[k] = value
```

```
        while k > 1:
            k >>= 1
            tree[k] = op(tree[k << 1], tree[(k << 1) + 1])

    def __getitem__(self, i):
        return self.tree[i + self.elem_size]

    def debug(self):
        print(self.tree[self.elem_size : self.elem_size + self.real_size])
```

`__slots__` は高速化用。`op` には二項演算関数を与える。`get_value` では半開区間 $[x, y)$ に `op` を適用し続けた結果を返す。完全二分木を保持するので、計算量は $\sum_{i=0}^{\infty} \frac{n}{2^i} \leq 2N$ より空間 $O(N)$. クエリは $O(a \log N)$ で、これは木の高さが約 $\log N$ で各深さで選ばれるノードが高々 2 個であることから従う。ただし与えた演算に $O(a)$ かかるとき。なお、セグ木に乗せる代数系はモノイドであれば十分である。これは、区間の併合に結合則を要求することと、初期化に単位元を要求することからわかる。なお、上の実装ではデフォルトを Range Minimum Query (RmQ) にしており、これはモノイド $\langle \mathbb{R}, \min \rangle$ におけるセグ木である。単位元は ∞ とすればよい。

例題 169. 東西長さ L の廊下がある。この廊下を全て照らすために N 本の蛍光灯のうちいくつかを使う。それぞれ西から l_i ～ r_i の位置を照らすことができ、設置にコスト c_i かかる。コストを最小化せよ。$1 \leq N, L, C_i \leq 10^5$　　ARC026 C - 蛍光灯

```
(N, L), *d = [[*map(int, o.split())] for o in open(0)]
d.sort()
dp = [0] + [float("inf")] * L
dp = SegmentTree(dp)
for l, r, c in d:
    dp[r] = min(dp[r], dp.get_value(l, r + 1) + c)
print(dp[L])
```

$dp[i] \coloneqq i$ まで照らすのに使う最小コスト　とする。すると $dp[r]$ の最小値更新は $[l, r]$ での最小値に c_i を足したものに行っていけばよい。半開区間に注意してセグ木を使う。