Python KACTL

KTH Algorithm Competition Template Library

Python Edition

November 05, 2025

134 Algorithms

Table of Contents

Combinatorial

Int Perm

Permutation -> integer conversion. (Not order preserving.)

Time: O(n)

combinatorial/int_perm.py

```
"""
Author: Simon Lindholm
Date: 2018-07-06
License: CC0
Description: Permutation -> integer conversion. (Not order
preserving.)
Integer -> permutation can use a lookup table.
Time: O(n)
"""

from typing import List

def perm_to_int(v: List[int]) -> int:
    """Convert permutation to integer"""
    use = 0
    i = 0
    r = 0
    for x in v:
        i += 1
        # Count how many bits are set in use for positions < x
        r = r * i + bin(use & -(1 << x)).count('1')
        use |= 1 << x
    return r
```

Multinomial

Computes multinomial coefficient (k1 + ... + kn)! / (k1! * k2! * ... * kn!)

combinatorial/multinomial.py

```
"""
Author: Mattias de Zalenski, Fredrik Niemelä, Per Austrin, Simon Lindholm
Date: 2002-09-26
Source: Max Bennedich
Description: Computes multinomial coefficient (k1 + ... + kn)! / (k1! * k2! * ... * kn!)
Status: Tested on kattis:lexicography
"""

from typing import List

def multinomial(v: List[int]) -> int:
    """
    Compute multinomial coefficient.
    v = list of partition sizes
    Returns (sum v_i)! / (v_1! * v_2! * ... * v_n!)
    """
    if not v:
        return 1

    c = 1
    m = v[0]
    for i in range(1, len(v)):
        for j in range(v[i]):
            m += 1
            c = c * m // (j + 1)
    return c
```

Number Theory

Continued Fractions

Given N and a real number x >= 0, finds the closest rational approximation p/q with p, q <= N.

Time: O(log N)

number_theory/continued_fractions.py

```
"""
Author: Simon Lindholm
Date: 2018-07-15
License: CC0
Source: Wikipedia
Description: Given N and a real number x >= 0, finds the closest rational approximation p/q with p, q <= N.
It will obey |p/q - x| <= 1/(qN).
Time: O(log N)
Status: stress-tested for n <= 300
"""

from typing import Tuple

def approximate(x: float, N: int) -> Tuple[int, int]:
    """
    Find closest rational approximation p/q to x with p, q <= N.
    Returns (p, q)
    """
    LP = 0
    LQ = 1
    P = 1
    Q = 0
    inf = 10**18
    y = x

    while True:
        lim = min(inf if P == 0 else (N - LP) // P,
                  inf if Q == 0 else (N - LQ) // Q)
        a = int(y)
        b = min(a, lim)
        NP = b * P + LP
        NQ = b * Q + LQ
        if a > b:
            # Semi-convergent case
            return (NP, NQ) if abs(x - NP / NQ) < abs(x - P / Q) else (P, Q)

        if abs(y) < 1e-15 or 1 / (y - a) > 3 * N:
            return (NP, NQ)

        y = 1 / (y - a)
        LP = P
        P = NP
        LQ = Q
        Q = NQ
```

Crt

Chinese Remainder Theorem.

Time: log(n)

number_theory/crt.py

```
"""
Author: Simon Lindholm
Date: 2019-05-22
License: CC0
Description: Chinese Remainder Theorem.

crt(a, m, b, n) computes x such that x ≡ a (mod m), x ≡ b (mod
n).
If |a| < m and |b| < n, x will obey 0 <= x < lcm(m, n).
Assumes mn < 2^62.
Time: log(n)
Status: Works
"""

from .euclid import euclid

def crt(a: int, m: int, b: int, n: int) -> int:
    """Chinese Remainder Theorem"""
    if n > m:
        a, b = b, a
        m, n = n, m

    g, x, y = euclid(m, n)
    assert (a - b) % g == 0, "No solution exists"

    x = (b - a) % n * x % n // g * m + a
    return x if x >= 0 else x + m * n // g
```

Eratosthenes

Prime sieve for generating all primes up to a certain limit. isprime[i] is true iff i is a prime.

Time: lim=100'000'000 ~ 0.8 s. Runs 30% faster if only odd indices are stored.

number_theory/eratosthenes.py

```
"""
Author: Håkan Terelius
Date: 2009-08-26
License: CC0
Source: http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes
Description: Prime sieve for generating all primes up to a
certain limit. isprime[i] is true iff i is a prime.
Time: lim=100'000'000 ~ 0.8 s. Runs 30% faster if only odd
indices are stored.
Status: Tested
"""

from typing import List

def eratosthenes_sieve(lim: int) -> List[int]:
    """Generate all primes up to lim using Sieve of
Eratosthenes"""
    isprime = [True] * lim
    if lim > 0:
        isprime[0] = False
    if lim > 1:
        isprime[1] = False

    for i in range(4, lim, 2):
        isprime[i] = False

    i = 3
    while i * i < lim:
        if isprime[i]:
            for j in range(i * i, lim, i * 2):
                isprime[j] = False
        i += 2

    pr = []
    for i in range(2, lim):
        if isprime[i]:
            pr.append(i)
    return pr
```

Euclid

Finds two integers x and y, such that ax+by=gcd(a,b).

number_theory/euclid.py

```
"""
Author: Unknown
Date: 2002-09-15
Source: predates tinyKACTL
Description: Finds two integers x and y, such that
ax+by=gcd(a,b).
If you just need gcd, use math.gcd instead.
If a and b are coprime, then x is the inverse of a (mod b).
"""

from typing import Tuple

def euclid(a: int, b: int) -> Tuple[int, int, int]:
    """
    Extended Euclidean Algorithm.
    Returns (gcd, x, y) where ax + by = gcd(a, b)
    """
    if b == 0:
        return (a, 1, 0)

    d, y, x = euclid(b, a % b)
    y -= (a // b) * x
    return (d, x, y)
```

Factor

Pollard-rho randomized factorization algorithm. Returns prime
Time: O(n^{1/4}), less for numbers with small factors.

number_theory/factor.py

```
"""
Author: chilli, SJTU, pajenegod
Date: 2020-03-04
License: CC0
Source: own
Description: Pollard-rho randomized factorization algorithm.
Returns prime
factors of a number, in arbitrary order (e.g. 2299 -> {11, 19,
11}).
Time: O(n^{1/4}), less for numbers with small factors.
Status: stress-tested
"""

import math
from typing import List
from .mod_mul_ll import modmul
from .miller_rabin import is_prime

def pollard(n: int) -> int:
    """Pollard's rho algorithm to find a factor of n"""
    x = 0
    y = 0
    t = 30
    prd = 2
    i = 1

    def f(x_val):
        return modmul(x_val, x_val, n) + i

    while t % 40 != 0 or math.gcd(prd, n) == 1:
        t += 1
        if x == y:
            i += 1
            x = i
            y = f(x)
        q = modmul(prd, max(x, y) - min(x, y), n)
        if q:
            prd = q
        x = f(x)
        y = f(f(y))

    return math.gcd(prd, n)

def factor(n: int) -> List[int]:
    """Factor n into prime factors"""
    if n == 1:
        return []
    if is_prime(n):
        return [n]
    x = pollard(n)
    l = factor(x)
    r = factor(n // x)
    l.extend(r)
    return l
```

Fast Eratosthenes

Prime sieve for generating all primes smaller than LIM.
Time: LIM=1e9 ≈ 1.5s

number_theory/fast_eratosthenes.py

```
"""
Author: Jakob Kogler, chilli, pajenegod
Date: 2020-04-12
License: CC0
Description: Prime sieve for generating all primes smaller than
LIM.
Time: LIM=1e9 ≈ 1.5s
Status: Stress-tested
Details: Despite its n log log n complexity, segmented sieve is
still faster
than other options due to low memory usage which reduces cache
misses.
This implementation skips even numbers.
"""

import math
from typing import List

def fast_eratosthenes(LIM: int) -> List[int]:
    """
    Fast segmented sieve for generating primes up to LIM.
    Returns list of all primes < LIM.
    """
    S = int(math.sqrt(LIM))
    R = LIM // 2

    pr = [2]
    sieve = [False] * (S + 1)

    # Generate small primes
    for i in range(3, S + 1, 2):
        if not sieve[i]:
            for j in range(i * i, S + 1, 2 * i):
                sieve[j] = True

    # Collect small primes with their starting positions
    cp = []
    for i in range(3, S + 1, 2):
        if not sieve[i]:
            cp.append((i, i * i // 2))

    # Segmented sieve
    for L in range(1, R + 1, S):
        block = [False] * S

        for idx_pair in cp:
            p, idx = idx_pair[0], idx_pair[1]
            i = idx
            while i < S + L:
                if i - L >= 0 and i - L < S:
                    block[i - L] = True
                i += p
            # Update idx for next iteration
            cp[cp.index(idx_pair)] = (p, i)

        for i in range(min(S, R - L)):
            if not block[i]:
                pr.append((L + i) * 2 + 1)

    return pr
```

Frac Binary Search

Given f and N, finds the smallest fraction p/q in [0, 1]
Time: O(log(N))

number_theory/frac_binary_search.py

```
"""
Author: Lucian Bicsi, Simon Lindholm
Date: 2017-10-31
License: CC0
Description: Given f and N, finds the smallest fraction p/q in
[0, 1]
such that f(p/q) is true, and p, q <= N.
Time: O(log(N))
Status: stress-tested for n <= 300
"""

from typing import Callable, Tuple

class Frac:
    def __init__(self, p: int, q: int):
        self.p = p
        self.q = q

def frac_binary_search(f: Callable[[Frac], bool], N: int) ->
Tuple[int, int]:
    """
    Binary search over fractions.
    f = predicate function taking a Frac
    N = maximum value for p and q
    Returns (p, q) where p/q is the smallest fraction satisfying
f
    """
    dir_flag = True
    A = True
    B = True
    lo = Frac(0, 1)
    hi = Frac(1, 1)  # Set to Frac(1, 0) to search (0, N]

    if f(lo):
        return (lo.p, lo.q)

    assert f(hi), "f(1/1) must be true"

    while A or B:
        adv = 0
        step = 1

        # Binary search for advancement
        si = 0
        while step:
            adv += step
            mid = Frac(lo.p * adv + hi.p, lo.q * adv + hi.q)

            if abs(mid.p) > N or mid.q > N or dir_flag == (not
f(mid)):
                adv -= step
                si = 2

            if si == 2:
                si = 0
            step = (step * 2) >> si if step else 0

        hi.p += lo.p * adv
        hi.q += lo.q * adv
        dir_flag = not dir_flag
        lo, hi = hi, lo
        A = B
        B = bool(adv)

# ... (continued)
```

Miller Rabin

Deterministic Miller-Rabin primality test.

Time: 7 times the complexity of a^b mod c.

number_theory/miller_rabin.py

```
"""
Author: chilli, c1729, Simon Lindholm
Date: 2019-03-28
License: CC0
Source: Wikipedia, https://miller-rabin.appspot.com/
Description: Deterministic Miller-Rabin primality test.
Guaranteed to work for numbers up to 7*10^18; for larger numbers,
use Python and extend A randomly.
Time: 7 times the complexity of a^b mod c.
Status: Stress-tested
"""

from .mod_mul_ll import modpow

def is_prime(n: int) -> bool:
    """Deterministic Miller-Rabin primality test"""
    if n < 2 or n % 6 % 4 != 1:
        return (n | 1) == 3

    A = [2, 325, 9375, 28178, 450775, 9780504, 1795265022]
    s = (n - 1) & -(n - 1)  # count trailing zeroes: isolate
lowest set bit
    s = s.bit_length() - 1  # convert to count
    d = n >> s

    for a in A:
        p = modpow(a % n, d, n)
        i = s
        while p != 1 and p != n - 1 and a % n and i:
            p = (p * p) % n
            i -= 1
        if p != n - 1 and i != s:
            return False
    return True
```

Mod Inverse

Pre-computation of modular inverses. Assumes LIM <= mod and that mod is a prime.

number_theory/mod_inverse.py

```
"""
Author: Simon Lindholm
Date: 2016-07-24
License: CC0
Source: Russian page
Description: Pre-computation of modular inverses. Assumes LIM <=
mod and that mod is a prime.
Status: Works
"""

from typing import List

def compute_inverses(LIM: int, mod: int) -> List[int]:
    """Pre-compute modular inverses for 1 to LIM-1 modulo mod
(mod must be prime)"""
    inv = [0] * LIM
    inv[1] = 1
    for i in range(2, LIM):
        inv[i] = mod - (mod // i) * inv[mod % i] % mod
    return inv
```

Mod Log

Returns the smallest x > 0 s.t. a^x = b (mod m), or

Time: O(√m)

number_theory/mod_log.py

```
"""
Author: Bjorn Martinsson
Date: 2020-06-03
License: CC0
Source: own work
Description: Returns the smallest x > 0 s.t. a^x = b (mod m), or
-1 if no such x exists. mod_log(a,1,m) can be used to
calculate the order of a.
Time: O(√m)
Status: tested for all 0 <= a,x < 500 and 0 < m < 500.
"""

import math
from typing import Dict

def mod_log(a: int, b: int, m: int) -> int:
    """
    Discrete logarithm: find smallest x > 0 such that a^x ≡ b
(mod m)
    Returns -1 if no solution exists
    Uses baby-step giant-step algorithm
    """
    n = int(math.sqrt(m)) + 1
    e = 1
    f = 1
    j = 1
    A: Dict[int, int] = {}

    # Baby step
    while j <= n:
        e = f = e * a % m
        if e == b % m:
            return j
        A[e * b % m] = j
        j += 1

    # Giant step
    if math.gcd(m, e) == math.gcd(m, b):
        for i in range(2, n + 2):
            e = e * f % m
            if e in A:
                return n * i - A[e]

    return -1
```

Mod Mul Ll

Calculate a*b mod c (or a^b mod c) for 0 <= a, b <= c <= 7.2*10^18.

Time: O(1) for modmul, O(log b) for modpow

number_theory/mod_mul_ll.py

```
"""
Author: chilli, Ramchandra Apte, Noam527, Simon Lindholm
Date: 2019-04-24
License: CC0
Source:
https://github.com/RamchandraApte/OmniTemplate/blob/master/src/nu
mber_theory/modulo.hpp
Description: Calculate a*b mod c (or a^b mod c) for 0 <= a, b <=
c <= 7.2*10^18.
Time: O(1) for modmul, O(log b) for modpow
Status: stress-tested, proven correct
"""

def modmul(a: int, b: int, M: int) -> int:
    """Multiply a*b mod M for large numbers"""
    # For Python, we can use native arbitrary precision
    # But for compatibility with C++ behavior, we implement the
same logic
    ret = a * b - M * int(1.0 / M * a * b)
    if ret < 0:
        ret += M
    if ret >= M:
        ret -= M
    return ret

def modpow(b: int, e: int, mod: int) -> int:
    """Compute b^e mod mod using modmul"""
    ans = 1
    while e:
        if e & 1:
            ans = modmul(ans, b, mod)
        b = modmul(b, b, mod)
        e //= 2
    return ans
```

Mod Pow

Modular exponentiation

number_theory/mod_pow.py

```
"""
Author: Noam527
Date: 2019-04-24
License: CC0
Source: folklore
Description: Modular exponentiation
Status: tested
"""

MOD = 1000000007  # faster if const

def modpow(b: int, e: int, mod: int = MOD) -> int:
    """Compute b^e mod mod"""
    ans = 1
    while e:
        if e & 1:
            ans = ans * b % mod
        b = b * b % mod
        e //= 2
    return ans
```

Mod Sqrt

Tonelli-Shanks algorithm for modular square roots. Finds x s.t. x^2 = a (mod p) (-x gives the other solution).

Time: O(log^2 p) worst case, O(log p) for most p

number_theory/mod_sqrt.py

```
"""
Author: Simon Lindholm
Date: 2016-08-31
License: CC0
Source: http://eli.thegreenplace.net/2009/03/07/computing-
modular-square-roots-in-python/
Description: Tonelli-Shanks algorithm for modular square roots.
Finds x s.t. x^2 = a (mod p) (-x gives the other solution).
Time: O(log^2 p) worst case, O(log p) for most p
Status: Tested for all a,p <= 10000
"""

from .mod_pow import modpow

def mod_sqrt(a: int, p: int) -> int:
    """
    Find x such that x^2 ≡ a (mod p), where p is prime
    Returns one solution; -x (mod p) is the other solution
    Raises assertion error if no solution exists
    """
    a %= p
    if a < 0:
        a += p
    if a == 0:
        return 0

    assert modpow(a, (p - 1) // 2, p) == 1, "No solution exists"

    if p % 4 == 3:
        return modpow(a, (p + 1) // 4, p)

    # Tonelli-Shanks algorithm
    s = p - 1
    r = 0
    while s % 2 == 0:
        r += 1
        s //= 2

    # Find a non-square mod p
    n = 2
    while modpow(n, (p - 1) // 2, p) != p - 1:
        n += 1

    x = modpow(a, (s + 1) // 2, p)
    b = modpow(a, s, p)
    g = modpow(n, s, p)

    while True:
        t = b
        m = 0
        while m < r and t != 1:
            t = t * t % p
            m += 1

        if m == 0:
            return x

        gs = modpow(g, 1 << (r - m - 1), p)
        g = gs * gs % p
        x = x * gs % p
        b = b * g % p
        r = m
# ... (continued)
```

Mod Sum

Sums of mod'ed arithmetic progressions.

Time: log(m), with a large constant.

number_theory/mod_sum.py

```
"""
Author: Simon Lindholm
Date: 2015-06-23
License: CC0
Source: own work
Description: Sums of mod'ed arithmetic progressions.

modsum(to, c, k, m) = sum_{i=0}^{to-1} ((ki+c) % m).
divsum is similar but for floored division.
Time: log(m), with a large constant.
Status: Tested for all |k|,|c|,to,m <= 50, and on kattis:aladin
"""

def sumsq(to: int) -> int:
    """Sum of first 'to' numbers, written to handle overflows"""
    return to // 2 * ((to - 1) | 1)

def divsum(to: int, c: int, k: int, m: int) -> int:
    """Sum of floored divisions"""
    res = k // m * sumsq(to) + c // m * to
    k %= m
    c %= m
    if not k:
        return res
    to2 = (to * k + c) // m
    return res + (to - 1) * to2 - divsum(to2, m - 1 - c, m, k)

def modsum(to: int, c: int, k: int, m: int) -> int:
    """Sum of modded arithmetic progression"""
    c = ((c % m) + m) % m
    k = ((k % m) + m) % m
    return to * c + k * sumsq(to) - m * divsum(to, c, k, m)
```

Modular Arithmetic

Class for modular arithmetic operations.

number_theory/modular_arithmetic.py

```
"""
Author: Lukas Polacek
Date: 2009-09-28
License: CC0
Source: folklore
Description: Class for modular arithmetic operations.
You need to set mod to some number first and then you can use the
structure.
"""

from .euclid import euclid

class Mod:
    """Modular arithmetic class"""
    mod = 10**9 + 7  # Default modulus, change as needed

    def __init__(self, x: int):
        self.x = x % self.mod

    def __add__(self, other: 'Mod') -> 'Mod':
        return Mod((self.x + other.x) % self.mod)

    def __sub__(self, other: 'Mod') -> 'Mod':
        return Mod((self.x - other.x + self.mod) % self.mod)

    def __mul__(self, other: 'Mod') -> 'Mod':
        return Mod((self.x * other.x) % self.mod)

    def __truediv__(self, other: 'Mod') -> 'Mod':
        return self * other.invert()

    def invert(self) -> 'Mod':
        """Modular inverse"""
        g, x, y = euclid(self.x, self.mod)
        assert g == 1, "Modular inverse does not exist"
        return Mod((x + self.mod) % self.mod)

    def __pow__(self, e: int) -> 'Mod':
        """Modular exponentiation"""
        if e == 0:
            return Mod(1)
        r = self ** (e // 2)
        r = r * r
        return self * r if e & 1 else r

    def __repr__(self) -> str:
        return f"Mod({self.x})"

    def __int__(self) -> int:
        return self.x

# Usage example:
# Mod.mod = 1000000007
# a = Mod(5)
# b = Mod(3)
# c = a + b  # 8
# d = a * b  # 15
# e = a / b  # modular division
```

Phi Function

Euler's φ function is defined as φ(n) := # of positive integers <= n that are coprime with n.

number_theory/phi_function.py

```
"""
Author: Håkan Terelius
Date: 2009-09-25
License: CC0
Source: http://en.wikipedia.org/wiki/Euler's_totient_function
Description: Euler's φ function is defined as φ(n) := # of
positive integers <= n that are coprime with n.
φ(1)=1, p prime => φ(p^k)=(p-1)p^(k-1), m,n coprime =>
φ(mn)=φ(m)φ(n).

Euler's thm: a,n coprime => a^φ(n) ≡ 1 (mod n).
Fermat's little thm: p prime => a^(p-1) ≡ 1 (mod p) for all a.
Status: Tested
"""

from typing import List

def calculate_phi(LIM: int) -> List[int]:
    """Calculate Euler's totient function for all numbers up to
LIM"""
    phi = [i if i & 1 else i // 2 for i in range(LIM)]

    for i in range(3, LIM, 2):
        if phi[i] == i:  # i is prime
            for j in range(i, LIM, i):
                phi[j] -= phi[j] // i

    return phi
```

Data Structures

Fenwick Tree

Computes partial sums a[0] + a[1] + ... + a[pos - 1], and updates single elements a[i],

Time: Both operations are O(log N).

data_structures/fenwick_tree.py

```
"""
Author: Lukas Polacek
Date: 2009-10-30
License: CC0
Source: folklore/TopCoder
Description: Computes partial sums a[0] + a[1] + ... + a[pos -
1], and updates single elements a[i],
taking the difference between the old and new value.
Time: Both operations are O(log N).
Status: Stress-tested
"""

from typing import List

class FenwickTree:
    def __init__(self, n: int):
        self.s = [0] * n

    def update(self, pos: int, dif: int):
        """a[pos] += dif"""
        while pos < len(self.s):
            self.s[pos] += dif
            pos |= pos + 1

    def query(self, pos: int) -> int:
        """sum of values in [0, pos)"""
        res = 0
        while pos > 0:
            res += self.s[pos - 1]
            pos &= pos - 1
        return res

    def lower_bound(self, sum_val: int) -> int:
        """min pos st sum of [0, pos] >= sum_val
        Returns n if no sum is >= sum_val, or -1 if empty sum
is."""
        if sum_val <= 0:
            return -1
        pos = 0
        pw = 1 << 25
        while pw:
            if pos + pw <= len(self.s) and self.s[pos + pw - 1] <
sum_val:
                pos += pw
                sum_val -= self.s[pos - 1]
            pw >>= 1
        return pos
```

Fenwick Tree 2D

Computes sums a[i,j] for all i<I, j<J, and increases single elements a[i,j].

Time: O(log^2 N). (Use persistent segment trees for O(log N).)

data_structures/fenwick_tree_2d.py

```
"""
Author: Simon Lindholm
Date: 2017-05-11
License: CC0
Source: folklore
Description: Computes sums a[i,j] for all i<I, j<J, and increases
single elements a[i,j].
Requires that the elements to be updated are known in advance
(call fake_update() before init()).
Time: O(log^2 N). (Use persistent segment trees for O(log N).)
Status: stress-tested
"""

from typing import List
from bisect import bisect_left
from .fenwick_tree import FenwickTree

class FenwickTree2D:
    def __init__(self, limx: int):
        self.ys = [[] for _ in range(limx)]
        self.ft = []

    def fake_update(self, x: int, y: int):
        """Register that position (x, y) will be updated"""
        while x < len(self.ys):
            self.ys[x].append(y)
            x |= x + 1

    def init(self):
        """Initialize after all fake_update() calls"""
        for v in self.ys:
            v.sort()
            self.ft.append(FenwickTree(len(v)))

    def ind(self, x: int, y: int) -> int:
        """Find index of y in ys[x]"""
        return bisect_left(self.ys[x], y)

    def update(self, x: int, y: int, dif: int):
        """Add dif to position (x, y)"""
        while x < len(self.ys):
            self.ft[x].update(self.ind(x, y), dif)
            x |= x + 1

    def query(self, x: int, y: int) -> int:
        """Query sum of rectangle [0, x) x [0, y)"""
        total = 0
        while x:
            total += self.ft[x - 1].query(self.ind(x - 1, y))
            x &= x - 1
        return total
```

Hash Map Note

data_structures/hash_map_note.py

```
"""
Hash Map in Python
==================

The C++ HashMap uses GNU PBDS (gp_hash_table) with custom hash
function.
Python has excellent built-in dictionary support that's usually
sufficient.

Method 1: Built-in dict (Recommended for most cases)
----------------------------------------------------
```python
# Python's dict is highly optimized
h = {}
h[key] = value
val = h.get(key, default)
```

Method 2: collections.defaultdict

```python
from collections import defaultdict

# Automatic default values
h = defaultdict(int)  # defaults to 0
h[key] += 1  # no KeyError

h = defaultdict(list)  # defaults to []
h[key].append(value)
```

Method 3: Custom Hash with salt (For competitive programming)

```python
import random
import sys

class FastHash:
    '''Hash map with custom hash to avoid hacking'''

    def __init__(self):
        self.RANDOM = random.randrange(2**62)
        self.data = {}

    def _hash(self, x):
        '''Custom hash function with random salt'''
        # Mix bits and add randomness
        x = (x ^ self.RANDOM) * 0xbf58476d1ce4e5b9
        x = (x ^ (x >> 30)) * 0x94d049bb133111eb
        return (x ^ (x >> 27)) & sys.maxsize

    def __setitem__(self, key, value):
        self.data[self._hash(key)] = (key, value)

    def __getitem__(self, key):
        h = self._hash(key)
        if h in self.data and self.data[h][0] == key:
            return self.data[h][1]
        raise KeyError(key)

    def get(self, key, default=None):
        try:
# ... (continued)
```

Lazy Segment Tree

Segment tree with ability to add or set values of large intervals, and compute max of intervals.

Time: O(log N).

data_structures/lazy_segment_tree.py

```
"""
Author: Simon Lindholm
Date: 2016-10-08
License: CC0
Source: me
Description: Segment tree with ability to add or set values of
large intervals, and compute max of intervals.
Can be changed to other things.
Time: O(log N).
Status: stress-tested a bit
"""

from typing import List, Optional

INF = 10**9

class LazySegmentTreeNode:
    def __init__(self, lo: int, hi: int, v: Optional[List[int]] =
None):
        self.l: Optional['LazySegmentTreeNode'] = None
        self.r: Optional['LazySegmentTreeNode'] = None
        self.lo = lo
        self.hi = hi
        self.mset = INF
        self.madd = 0
        self.val = -INF

        if v is not None:
            if lo + 1 < hi:
                mid = lo + (hi - lo) // 2
                self.l = LazySegmentTreeNode(lo, mid, v)
                self.r = LazySegmentTreeNode(mid, hi, v)
                self.val = max(self.l.val, self.r.val)
            else:
                self.val = v[lo]

    def query(self, L: int, R: int) -> int:
        """Query maximum in range [L, R)"""
        if R <= self.lo or self.hi <= L:
            return -INF
        if L <= self.lo and self.hi <= R:
            return self.val
        self.push()
        return max(self.l.query(L, R), self.r.query(L, R))

    def set(self, L: int, R: int, x: int):
        """Set all elements in range [L, R) to x"""
        if R <= self.lo or self.hi <= L:
            return
        if L <= self.lo and self.hi <= R:
            self.mset = x
            self.val = x
            self.madd = 0
        else:
            self.push()
            self.l.set(L, R, x)
            self.r.set(L, R, x)
            self.val = max(self.l.val, self.r.val)

    def add(self, L: int, R: int, x: int):
        """Add x to all elements in range [L, R)"""
        if R <= self.lo or self.hi <= L:
# ... (continued)
```
```

Line Container

Container where you can add lines of the form kx+m, and query maximum values at points x.

Time: O(log N)

data_structures/line_container.py

```
"""
Author: Simon Lindholm
Date: 2017-04-20
License: CC0
Source: own work
Description: Container where you can add lines of the form kx+m,
and query maximum values at points x.
Useful for dynamic programming ("convex hull trick").
Time: O(log N)
Status: stress-tested
"""

from sortedcontainers import SortedList
from typing import Optional

class Line:
    def __init__(self, k: int, m: int, p: int = 0):
        self.k = k
        self.m = m
        self.p = p  # intersection point with next line

    def __lt__(self, other):
        if isinstance(other, Line):
            return self.k < other.k
        else:
            return self.p < other

class LineContainer:
    """Container for convex hull trick"""
    INF = 10**18

    def __init__(self):
        self.lines = SortedList(key=lambda x: x.k)

    def div(self, a: int, b: int) -> int:
        """Floored division"""
        return a // b - (1 if (a ^ b) < 0 and a % b else 0)

    def isect(self, x: Line, y: Optional[Line]) -> bool:
        """Update intersection point and check if x should be
removed"""
        if y is None:
            x.p = self.INF
            return False
        if x.k == y.k:
            x.p = self.INF if x.m > y.m else -self.INF
        else:
            x.p = self.div(y.m - x.m, x.k - y.k)
        return x.p >= y.p

    def add(self, k: int, m: int):
        """Add line kx + m"""
        new_line = Line(k, m)
        idx = self.lines.bisect_left(new_line)
        self.lines.add(new_line)

        # Update intersections
        y_idx = idx + 1
        if y_idx < len(self.lines):
            y = self.lines[y_idx]
            while self.isect(new_line, y if y_idx <
len(self.lines) else None):
                if y_idx < len(self.lines):
# ... (continued)
```

Link Cut Tree

Represents a forest of unrooted trees. You can add and remove

Time: All operations take amortized O(log N).

data_structures/link_cut_tree.py

```
"""
Author: Simon Lindholm
Date: 2016-07-25
Source: https://github.com/ngthanhtrung23/ACM_Notebook_new
Description: Represents a forest of unrooted trees. You can add
and remove
edges (as long as the result is still a forest), and check
whether
two nodes are in the same tree.
Time: All operations take amortized O(log N).
Status: Stress-tested a bit for N <= 20
"""

from typing import Optional

class LCTNode:
    """Splay tree node for Link-Cut Tree"""

    def __init__(self):
        self.p: Optional[LCTNode] = None  # Parent in splay tree
        self.pp: Optional[LCTNode] = None  # Path parent
        self.c = [None, None]  # Children [left, right]
        self.flip = False  # Lazy flip flag
        self.fix()

    def fix(self):
        """Update parent pointers"""
        if self.c[0]:
            self.c[0].p = self
        if self.c[1]:
            self.c[1].p = self

    def push_flip(self):
        """Push down flip operation"""
        if not self.flip:
            return
        self.flip = False
        self.c[0], self.c[1] = self.c[1], self.c[0]
        if self.c[0]:
            self.c[0].flip = not self.c[0].flip
        if self.c[1]:
            self.c[1].flip = not self.c[1].flip

    def up(self) -> int:
        """Return which child this is of its parent (-1 if
root)"""
        if not self.p:
            return -1
        return 1 if self.p.c[1] == self else 0

    def rot(self, i: int, b: int):
        """Rotate operation"""
        h = i ^ b
        x = self.c[i]
        y = x if b == 2 else x.c[h]
        z = y if b else x

        y.p = self.p
        if self.p:
            self.p.c[self.up()] = y

        self.c[i] = z.c[i ^ 1]
        if b < 2:
# ... (continued)
```

Matrix

Basic operations on square matrices.

data_structures/matrix.py

```
"""
Author: Ulf Lundstrom
Date: 2009-08-03
License: CC0
Source: My head
Description: Basic operations on square matrices.
Usage: A = Matrix(3); A.d = [[1,2,3],[4,5,6],[7,8,9]]
       vec = [1,2,3]; vec = (A ** N) * vec
Status: tested
"""

from typing import List

class Matrix:
    def __init__(self, n: int):
        self.n = n
        self.d = [[0] * n for _ in range(n)]

    def __mul__(self, other):
        """Matrix multiplication or matrix-vector
multiplication"""
        if isinstance(other, Matrix):
            result = Matrix(self.n)
            for i in range(self.n):
                for j in range(self.n):
                    for k in range(self.n):
                        result.d[i][j] += self.d[i][k] *
other.d[k][j]
            return result
        elif isinstance(other, list):
            # Matrix-vector multiplication
            result = [0] * self.n
            for i in range(self.n):
                for j in range(self.n):
                    result[i] += self.d[i][j] * other[j]
            return result
        else:
            raise TypeError("Can only multiply Matrix by Matrix
or list")

    def __pow__(self, p: int):
        """Matrix exponentiation"""
        assert p >= 0
        result = Matrix(self.n)
        # Identity matrix
        for i in range(self.n):
            result.d[i][i] = 1

        base = Matrix(self.n)
        base.d = [row[:] for row in self.d]

        while p:
            if p & 1:
                result = result * base
            base = base * base
            p >>= 1
        return result
```

Mo Queries

Answer interval or tree path queries by finding an approximate TSP through the queries,

Time: O(N √Q)

data_structures/mo_queries.py

```
"""
Author: Simon Lindholm
Date: 2019-12-28
License: CC0
Source: https://github.com/hoke-t/tamu-kactl/blob/master/content/data-structures/MoQueries.h
Description: Answer interval or tree path queries by finding an approximate TSP through the queries,
and moving from one query to the next by adding/removing points at the ends.
Time: O(N √Q)
Status: stress-tested
"""

from typing import List, Tuple, Callable

def mo_queries(Q: List[Tuple[int, int]],
               add: Callable[[int, int], None],
               delete: Callable[[int, int], None],
               calc: Callable[[], int],
               blk: int = 350) -> List[int]:
    """
    Answer interval queries using Mo's algorithm.
    Q = list of (left, right) queries
    add(ind, end) = function to add a[ind] (end = 0 for left, 1 for right)
    delete(ind, end) = function to remove a[ind]
    calc() = function to compute current answer
    blk = block size (~N/√Q)
    Returns list of answers for each query
    """
    L = 0
    R = 0
    s = list(range(len(Q)))
    res = [0] * len(Q)

    # Sort queries by (block, right endpoint with alternating order)
    def sort_key(i):
        block = Q[i][0] // blk
        r = Q[i][1]
        if block & 1:
            r = -r
        return (block, r)

    s.sort(key=sort_key)

    for qi in s:
        q = Q[qi]
        # Expand/contract the window
        while L > q[0]:
            L -= 1
            add(L, 0)
        while R < q[1]:
            add(R, 1)
            R += 1
        while L < q[0]:
            delete(L, 0)
            L += 1
        while R > q[1]:
            R -= 1
            delete(R, 1)
        res[qi] = calc()

    return res
# ... (continued)
```

Order Statistic Tree Note

data_structures/order_statistic_tree_note.py

```
"""
Order Statistic Tree in Python
==============================

The C++ OrderStatisticTree uses GNU Policy-Based Data Structures (PBDS),
which is not available in Python. However, Python has several alternatives:

Method 1: SortedContainers (Recommended)
----------------------------------------
```python
from sortedcontainers import SortedList

class OrderStatisticTree:
    def __init__(self):
        self.tree = SortedList()

    def insert(self, x):
        '''Insert element x'''
        self.tree.add(x)

    def remove(self, x):
        '''Remove element x'''
        self.tree.remove(x)

    def find_by_order(self, k):
        '''Find k-th smallest element (0-indexed)'''
        return self.tree[k] if 0 <= k < len(self.tree) else None

    def order_of_key(self, x):
        '''Count elements strictly less than x'''
        return self.tree.bisect_left(x)

    def lower_bound(self, x):
        '''Find first element >= x'''
        idx = self.tree.bisect_left(x)
        return self.tree[idx] if idx < len(self.tree) else None

# Usage
tree = OrderStatisticTree()
tree.insert(8)
tree.insert(10)
print(tree.order_of_key(10))  # 1
print(tree.find_by_order(0))  # 8
```

Method 2: Custom Treap Implementation

See treap.py in data_structures/ for a full implementation.

Method 3: bisect module (For simple cases)

```python
import bisect

class SimpleOST:
    def __init__(self):
        self.items = []

    def insert(self, x):
        bisect.insort(self.items, x)
# ... (continued)
```

Rmq

Range Minimum Queries on an array. Returns

Time: O(|V| log |V| + Q)

data_structures/rmq.py

```
"""
Author: Johan Sannemo, pajenegod
Date: 2015-02-06
License: CC0
Source: Folklore
Description: Range Minimum Queries on an array. Returns
min(V[a], V[a + 1], ... V[b - 1]) in constant time.
Usage:
  rmq = RMQ(values)
  rmq.query(inclusive, exclusive)
Time: O(|V| log |V| + Q)
Status: stress-tested
"""

from typing import List, TypeVar, Generic

T = TypeVar('T')

class RMQ(Generic[T]):
    def __init__(self, V: List[T]):
        self.jmp = [V[:]]
        pw = 1
        k = 1
        while pw * 2 <= len(V):
            new_row = []
            for j in range(len(V) - pw * 2 + 1):
                new_row.append(min(self.jmp[k - 1][j], self.jmp[k - 1][j + pw]))
            self.jmp.append(new_row)
            pw *= 2
            k += 1

    def query(self, a: int, b: int) -> T:
        assert a < b, "Invalid range query"
        dep = (b - a).bit_length() - 1
        return min(self.jmp[dep][a], self.jmp[dep][b - (1 << dep)])
```
```

Segment Tree

Zero-indexed max-tree. Bounds are inclusive to the left and exclusive to the right.

Time: O(log N)

data_structures/segment_tree.py

```
"""
Author: Lucian Bicsi
Date: 2017-10-31
License: CC0
Source: folklore
Description: Zero-indexed max-tree. Bounds are inclusive to the
left and exclusive to the right.
Can be changed by modifying T, f and unit.
Time: O(log N)
Status: stress-tested
"""

from typing import List, Callable, TypeVar

T = TypeVar('T')

class SegmentTree:
    def __init__(self, n: int, f: Callable[[T, T], T], unit: T,
def_val: T = None):
        """
        Create segment tree with n elements.
        f = associative function to combine elements
        unit = identity element
        def_val = default value (defaults to unit)
        """
        self.n = n
        self.f = f
        self.unit = unit
        if def_val is None:
            def_val = unit
        self.s = [def_val] * (2 * n)

    def update(self, pos: int, val: T):
        """Update position pos to value val"""
        pos += self.n
        self.s[pos] = val
        while pos > 1:
            pos //= 2
            self.s[pos] = self.f(self.s[pos * 2], self.s[pos * 2
+ 1])

    def query(self, b: int, e: int) -> T:
        """Query range [b, e)"""
        ra = self.unit
        rb = self.unit
        b += self.n
        e += self.n
        while b < e:
            if b % 2:
                ra = self.f(ra, self.s[b])
                b += 1
            if e % 2:
                e -= 1
                rb = self.f(self.s[e], rb)
            b //= 2
            e //= 2
        return self.f(ra, rb)

# Example usage for max tree:
def create_max_tree(n: int) -> SegmentTree:
    """Create a max segment tree"""
    return SegmentTree(n, max, float('-inf'))

# ... (continued)
```

Sub Matrix

Calculate submatrix sums quickly, given upper-left and lower-right corners (half-open).

Time: O(N^2 + Q)

data_structures/sub_matrix.py

```
"""
Author: Johan Sannemo
Date: 2014-11-28
License: CC0
Source: Folklore
Description: Calculate submatrix sums quickly, given upper-left
and lower-right corners (half-open).
Usage:
  m = SubMatrix(matrix)
  m.sum(0, 0, 2, 2)  # top left 4 elements
Time: O(N^2 + Q)
Status: Tested on Kattis
"""

from typing import List

class SubMatrix:
    def __init__(self, v: List[List[int]]):
        """Initialize with 2D array v"""
        R = len(v)
        C = len(v[0]) if R > 0 else 0
        self.p = [[0] * (C + 1) for _ in range(R + 1)]

        for r in range(R):
            for c in range(C):
                self.p[r + 1][c + 1] = (v[r][c] + self.p[r][c +
1] +
                                        self.p[r + 1][c] -
self.p[r][c])

    def sum(self, u: int, l: int, d: int, r: int) -> int:
        """Sum of submatrix from (u,l) to (d,r) (half-open)"""
        return self.p[d][r] - self.p[d][l] - self.p[u][r] +
self.p[u][l]
```

Treap

A short self-balancing tree. It acts as a

Time: O(log N)

data_structures/treap.py

```
"""
Author: someone on Codeforces
Date: 2017-03-14
Source: folklore
Description: A short self-balancing tree. It acts as a
sequential container with log-time splits/joins, and
is easy to augment with additional data.
Time: O(log N)
Status: stress-tested
"""

import random
from typing import Optional, Tuple, Callable

class TreapNode:
    def __init__(self, val: int):
        self.l: Optional['TreapNode'] = None
        self.r: Optional['TreapNode'] = None
        self.val = val
        self.y = random.randint(0, 2**31 - 1)
        self.c = 1  # count of nodes in subtree

    def recalc(self):
        """Recalculate subtree size"""
        self.c = cnt(self.l) + cnt(self.r) + 1

def cnt(n: Optional[TreapNode]) -> int:
    """Get count of nodes in subtree"""
    return n.c if n else 0

def each(n: Optional[TreapNode], f: Callable[[int], None]):
    """Apply function f to each value in-order"""
    if n:
        each(n.l, f)
        f(n.val)
        each(n.r, f)

def split(n: Optional[TreapNode], k: int) ->
Tuple[Optional[TreapNode], Optional[TreapNode]]:
    """Split treap at position k"""
    if not n:
        return None, None

    if cnt(n.l) >= k:  # For lower_bound(k), use "n.val >= k"
        L, R = split(n.l, k)
        n.l = R
        n.recalc()
        return L, n
    else:
        L, R = split(n.r, k - cnt(n.l) - 1)  # For lower_bound,
use just "k"
        n.r = L
        n.recalc()
        return n, R

def merge(l: Optional[TreapNode], r: Optional[TreapNode]) ->
Optional[TreapNode]:
    """Merge two treaps"""
    if not l:
        return r
    if not r:
        return l

# ... (continued)
```

Union Find

Disjoint-set data structure.

Time: O(α(N))

data_structures/union_find.py

```
"""
Author: Lukas Polacek
Date: 2009-10-26
License: CC0
Source: folklore
Description: Disjoint-set data structure.
Time: O(α(N))
"""

from typing import List

class UnionFind:
    def __init__(self, n: int):
        self.e = [-1] * n

    def same_set(self, a: int, b: int) -> bool:
        """Check if a and b are in the same set"""
        return self.find(a) == self.find(b)

    def size(self, x: int) -> int:
        """Get size of set containing x"""
        return -self.e[self.find(x)]

    def find(self, x: int) -> int:
        """Find representative of set containing x"""
        if self.e[x] < 0:
            return x
        self.e[x] = self.find(self.e[x])
        return self.e[x]

    def join(self, a: int, b: int) -> bool:
        """Join sets containing a and b. Returns True if they
were different sets."""
        a = self.find(a)
        b = self.find(b)
        if a == b:
            return False
        if self.e[a] > self.e[b]:
            a, b = b, a
        self.e[a] += self.e[b]
        self.e[b] = a
        return True
```

Union Find Rollback

Disjoint-set data structure with undo.

Time: O(log(N))

data_structures/union_find_rollback.py

```
"""
Author: Lukas Polacek, Simon Lindholm
Date: 2019-12-26
License: CC0
Source: folklore
Description: Disjoint-set data structure with undo.
If undo is not needed, skip st, time() and rollback().
Usage: t = uf.time(); ...; uf.rollback(t);
Time: O(log(N))
Status: tested as part of DirectedMST.h
"""

from typing import List, Tuple

class UnionFindRollback:
    def __init__(self, n: int):
        self.e = [-1] * n
        self.st: List[Tuple[int, int]] = []

    def size(self, x: int) -> int:
        """Get size of set containing x"""
        return -self.e[self.find(x)]

    def find(self, x: int) -> int:
        """Find representative of set containing x (no path
compression)"""
        return x if self.e[x] < 0 else self.find(self.e[x])

    def time(self) -> int:
        """Get current time for rollback"""
        return len(self.st)

    def rollback(self, t: int):
        """Rollback to time t"""
        while len(self.st) > t:
            idx, val = self.st.pop()
            self.e[idx] = val

    def join(self, a: int, b: int) -> bool:
        """Join sets containing a and b. Returns True if they
were different sets."""
        a = self.find(a)
        b = self.find(b)
        if a == b:
            return False
        if self.e[a] > self.e[b]:
            a, b = b, a
        self.st.append((a, self.e[a]))
        self.st.append((b, self.e[b]))
        self.e[a] += self.e[b]
        self.e[b] = a
        return True
```

Graph Algorithms

Bellman Ford

Calculates shortest paths from s in a graph that might have negative edge weights.

Time: O(VE)

graph/bellman_ford.py

```
"""
Author: Simon Lindholm
Date: 2015-02-23
License: CC0
Source: http://en.wikipedia.org/wiki/Bellman-Ford_algorithm
Description: Calculates shortest paths from s in a graph that
might have negative edge weights.
Unreachable nodes get dist = inf; nodes reachable through
negative-weight cycles get dist = -inf.
Time: O(VE)
Status: Tested on kattis:shortestpath3
"""

from typing import List

INF = 10**18

class Edge:
    def __init__(self, a: int, b: int, w: int):
        self.a = a
        self.b = b
        self.w = w

    def s(self) -> int:
        return self.a if self.a < self.b else -self.a

class Node:
    def __init__(self):
        self.dist = INF
        self.prev = -1

def bellman_ford(nodes: List[Node], eds: List[Edge], s: int):
    """
    Find shortest paths from source s.
    nodes = list of Node objects (modified in-place)
    eds = list of Edge objects
    s = source node
    """
    nodes[s].dist = 0
    eds.sort(key=lambda ed: ed.s())

    lim = len(nodes) // 2 + 2

    for i in range(lim):
        for ed in eds:
            cur = nodes[ed.a]
            dest = nodes[ed.b]
            if abs(cur.dist) == INF:
                continue
            d = cur.dist + ed.w
            if d < dest.dist:
                dest.prev = ed.a
                dest.dist = d if i < lim - 1 else -INF

    # Propagate negative infinity
    for i in range(lim):
        for e in eds:
            if nodes[e.a].dist == -INF:
                nodes[e.b].dist = -INF
```

Biconnected Components

Finds all biconnected components in an undirected graph, and

Time: O(E + V)

graph/biconnected_components.py

```
"""
Author: Simon Lindholm
Date: 2017-04-17
License: CC0
Source: folklore
Description: Finds all biconnected components in an undirected
graph, and
runs a callback for the edges in each. In a biconnected component
there
are at least two distinct paths between any two nodes. Note that
a node can
be in several components. An edge which is not in a component is
a bridge,
i.e., not part of any cycle.
Usage:
  ed = [[] for _ in range(N)]
  eid = 0
  for each edge (a,b):
    ed[a].append((b, eid))
    ed[b].append((a, eid))
    eid += 1
  bicomps(ed, lambda edgelist: ...)
Time: O(E + V)
Status: tested during MIPT ICPC Workshop 2017
"""

from typing import List, Tuple, Callable

def biconnected_components(ed: List[List[Tuple[int, int]]],
callback: Callable[[List[int]], None]):
    """
    Find all biconnected components.
    ed[i] = list of (neighbor, edge_id) pairs
    callback is called with list of edge IDs for each biconnected
component
    """
    num = [0] * len(ed)
    st = []
    time_counter = [0]

    def dfs(at: int, par: int) -> int:
        time_counter[0] += 1
        me = num[at] = time_counter[0]
        top = me

        for y, e in ed[at]:
            if e == par:
                continue

            if num[y]:
                top = min(top, num[y])
                if num[y] < me:
                    st.append(e)
            else:
                si = len(st)
                up = dfs(y, e)
                top = min(top, up)

                if up == me:
                    st.append(e)
                    callback(st[si:])
                    del st[si:]
                elif up < me:
                    st.append(e)
                # else: e is a bridge

# ... (continued)
```

Binary Lifting

Calculate power of two jumps in a tree,

Time: construction O(N log N), queries O(log N)

graph/binary_lifting.py

```
"""
Author: Johan Sannemo
Date: 2015-02-06
License: CC0
Source: Folklore
Description: Calculate power of two jumps in a tree,
to support fast upward jumps and LCAs.
Assumes the root node points to itself.
Time: construction O(N log N), queries O(log N)
Status: Tested at Petrozavodsk, also stress-tested via LCA.cpp
"""

from typing import List

def tree_jump(P: List[int]) -> List[List[int]]:
    """
    Build binary lifting table.
    P[i] = parent of node i (root points to itself)
    Returns jump table for fast ancestor queries
    """
    on = 1
    d = 1
    while on < len(P):
        on *= 2
        d += 1

    jmp = [P[:] for _ in range(d)]
    for i in range(1, d):
        for j in range(len(P)):
            jmp[i][j] = jmp[i - 1][jmp[i - 1][j]]

    return jmp

def jump(tbl: List[List[int]], nod: int, steps: int) -> int:
    """Jump 'steps' ancestors up from node 'nod'"""
    for i in range(len(tbl)):
        if steps & (1 << i):
            nod = tbl[i][nod]
    return nod

def lca(tbl: List[List[int]], depth: List[int], a: int, b: int) -
> int:
    """Find lowest common ancestor of nodes a and b"""
    if depth[a] < depth[b]:
        a, b = b, a
    a = jump(tbl, a, depth[a] - depth[b])
    if a == b:
        return a
    for i in range(len(tbl) - 1, -1, -1):
        c = tbl[i][a]
        d = tbl[i][b]
        if c != d:
            a = c
            b = d
    return tbl[0][a]
```

Compress Tree

Given a rooted tree and a subset S of nodes, compute the minimal

Time: O(|S| log |S|)

graph/compress_tree.py

```
"""
Author: Simon Lindholm
Date: 2016-01-14
License: CC0
Description: Given a rooted tree and a subset S of nodes, compute the minimal
subtree that contains all the nodes by adding all (at most |S|-1) pairwise LCA's and compressing edges.
Returns a list of (parent, original_index) representing a tree rooted at 0.
The root points to itself.
Time: O(|S| log |S|)
Status: Tested at CodeForces
"""

from typing import List, Tuple

def compress_tree(lca_structure, subset: List[int]) -> List[Tuple[int, int]]:
    """
    Compress tree to minimal subtree containing subset.
    lca_structure = LCA object with lca() and time[] methods
    subset = list of node indices to include
    Returns list of (parent_in_compressed_tree, original_node_index)
    """
    if not subset:
        return []

    T = lca_structure.time
    li = sorted(subset, key=lambda x: T[x])

    # Add all pairwise LCAs
    m = len(li) - 1
    for i in range(m):
        a = li[i]
        b = li[i + 1]
        li.append(lca_structure.lca(a, b))

    # Remove duplicates and sort by DFS time
    li = sorted(set(li), key=lambda x: T[x])

    # Build reverse mapping
    rev = {node: i for i, node in enumerate(li)}

    # Build compressed tree
    ret = [(0, li[0])]  # Root points to itself
    for i in range(len(li) - 1):
        a = li[i]
        b = li[i + 1]
        lca_node = lca_structure.lca(a, b)
        ret.append((rev[lca_node], b))

    return ret
```

Dfs Matching

Simple bipartite matching algorithm. Graph g should be a list

Time: O(VE)

graph/dfs_matching.py

```
"""
Author: Lukas Polacek
Date: 2009-10-28
License: CC0
Description: Simple bipartite matching algorithm. Graph g should be a list
of neighbors of the left partition, and btoa should be a vector full of
-1's of the same size as the right partition. Returns the size of
the matching. btoa[i] will be the match for vertex i on the right side,
or -1 if it's not matched.
Time: O(VE)
Usage: btoa = [-1] * m; dfs_matching(g, btoa)
Status: works
"""

from typing import List

def _find(j: int, g: List[List[int]], btoa: List[int], vis: List[bool]) -> bool:
    """DFS to find augmenting path"""
    if btoa[j] == -1:
        return True
    vis[j] = True
    di = btoa[j]
    for e in g[di]:
        if not vis[e] and _find(e, g, btoa, vis):
            btoa[e] = di
            return True
    return False

def dfs_matching(g: List[List[int]], btoa: List[int]) -> int:
    """
    Find maximum bipartite matching using DFS.
    g[i] = list of neighbors of node i in left partition
    btoa = list of matches for right partition (initially all -1)
    Returns size of matching
    """
    for i in range(len(g)):
        vis = [False] * len(btoa)
        for j in g[i]:
            if _find(j, g, btoa, vis):
                btoa[j] = i
                break

    return len(btoa) - btoa.count(-1)
```

Dinic

Flow algorithm with complexity O(VE log U) where U = max |cap|.

graph/dinic.py

```
"""
Author: chilli
Date: 2019-04-26
License: CC0
Source: https://cp-algorithms.com/graph/dinic.html
Description: Flow algorithm with complexity O(VE log U) where U = max |cap|.
O(min(E^{1/2}, V^{2/3})E) if U = 1; O(√V E) for bipartite matching.
Status: Tested on SPOJ FASTFLOW and SPOJ MATCHING, stress-tested
"""

from typing import List

class Dinic:
    class Edge:
        def __init__(self, to: int, rev: int, c: int, oc: int):
            self.to = to
            self.rev = rev
            self.c = c
            self.oc = oc

        def flow(self) -> int:
            """Get current flow through this edge"""
            return max(self.oc - self.c, 0)

    def __init__(self, n: int):
        self.lvl = [0] * n
        self.ptr = [0] * n
        self.q = [0] * n
        self.adj = [[] for _ in range(n)]

    def add_edge(self, a: int, b: int, c: int, rcap: int = 0):
        """Add edge from a to b with capacity c and reverse capacity rcap"""
        self.adj[a].append(self.Edge(b, len(self.adj[b]), c, c))
        self.adj[b].append(self.Edge(a, len(self.adj[a]) - 1, rcap, rcap))

    def dfs(self, v: int, t: int, f: int) -> int:
        """DFS to push flow"""
        if v == t or f == 0:
            return f

        while self.ptr[v] < len(self.adj[v]):
            e = self.adj[v][self.ptr[v]]
            if self.lvl[e.to] == self.lvl[v] + 1:
                p = self.dfs(e.to, t, min(f, e.c))
                if p:
                    e.c -= p
                    self.adj[e.to][e.rev].c += p
                    return p
            self.ptr[v] += 1
        return 0

    def calc(self, s: int, t: int) -> int:
        """Calculate max flow from s to t"""
        flow = 0
        self.q[0] = s

        for L in range(31):
            while True:
                self.lvl = [0] * len(self.q)
                self.ptr = [0] * len(self.q)
# ... (continued)
```

Directed Mst

Finds a minimum spanning tree/arborescence of a directed graph,

Time: O(E log V)

graph/directed_mst.py

```
"""
Author: chilli, Takanori MAEHARA, Benq, Simon Lindholm
Date: 2019-05-10
License: CC0
Source: https://github.com/spaghetti-source/algorithm
Description: Finds a minimum spanning tree/arborescence of a
directed graph,
given a root node. If no MST exists, returns -1.
Time: O(E log V)
Status: Stress-tested, also tested on NWERC 2018 fastestspeedrun
"""

from typing import List, Tuple, Optional
from collections import deque
from ..data_structures.union_find_rollback import
RollbackUnionFind

class Edge:
    def __init__(self, a: int, b: int, w: int):
        self.a = a  # source
        self.b = b  # destination
        self.w = w  # weight

class HeapNode:
    """Lazy skew heap node"""
    def __init__(self, key: Edge):
        self.key = key
        self.l: Optional[HeapNode] = None
        self.r: Optional[HeapNode] = None
        self.delta = 0

    def prop(self):
        """Propagate lazy delta"""
        self.key.w += self.delta
        if self.l:
            self.l.delta += self.delta
        if self.r:
            self.r.delta += self.delta
        self.delta = 0

    def top(self) -> Edge:
        self.prop()
        return self.key

def merge_heaps(a: Optional[HeapNode], b: Optional[HeapNode]) ->
Optional[HeapNode]:
    """Merge two skew heaps"""
    if not a or not b:
        return a if a else b

    a.prop()
    b.prop()

    if a.key.w > b.key.w:
        a, b = b, a

    a.l, a.r = merge_heaps(b, a.r), a.l
    return a

def pop_heap(a: HeapNode) -> Optional[HeapNode]:
    """Pop minimum from heap"""
    a.prop()
    return merge_heaps(a.l, a.r)
# ... (continued)
```

Edge Coloring

Given a simple, undirected graph with max degree D, computes a

Time: O(NM)

graph/edge_coloring.py

```
"""
Author: Simon Lindholm
Date: 2020-10-12
License: CC0
Source:
https://en.wikipedia.org/wiki/Misra_%26_Gries_edge_coloring_algor
ithm
Description: Given a simple, undirected graph with max degree D,
computes a
(D + 1)-coloring of the edges such that no neighboring edges
share a color.
(D-coloring is NP-hard, but can be done for bipartite graphs by
repeated
matchings of max-degree nodes.)
Time: O(NM)
Status: stress-tested, tested on kattis:gamescheduling
"""

from typing import List, Tuple

def edge_coloring(N: int, edges: List[Tuple[int, int]]) ->
List[int]:
    """
    Compute edge coloring with D+1 colors where D is max degree.
    N = number of nodes
    edges = list of (u, v) edges
    Returns list of colors for each edge
    """
    # Count degrees
    cc = [0] * (N + 1)
    for u, v in edges:
        cc[u] += 1
        cc[v] += 1

    ncols = max(cc) + 1
    ret = [0] * len(edges)

    # adj[u][c] = vertex adjacent to u with color c (or -1)
    adj = [[-1] * ncols for _ in range(N)]
    free = [0] * N  # free[u] = smallest free color at u

    fan = [0] * N
    cc_list = [0] * ncols
    loc = [0] * ncols

    for edge_idx, (u, v) in enumerate(edges):
        fan[0] = v
        for i in range(ncols):
            loc[i] = 0

        c = free[u]
        ind = 0

        # Build fan
        while True:
            d = free[v]
            if loc[d]:
                break
            if adj[u][d] == -1:
                break
            v = adj[u][d]
            ind += 1
            loc[d] = ind
            cc_list[ind] = d
            fan[ind] = v

# ... (continued)
```

Edmonds Karp

Flow algorithm with guaranteed complexity O(VE^2). To get edge flow values, compare

graph/edmonds_karp.py

```
"""
Author: Chen Xing
Date: 2009-10-13
License: CC0
Source: N/A
Description: Flow algorithm with guaranteed complexity O(VE^2).
To get edge flow values, compare
capacities before and after, and take the positive values only.
Status: stress-tested
"""

from typing import List, Dict

def edmonds_karp(graph: List[Dict[int, int]], source: int, sink:
int) -> int:
    """
    Edmonds-Karp max flow algorithm.
    graph[i] is a dict mapping node j to capacity from i to j
    Returns maximum flow from source to sink
    """
    assert source != sink, "Source and sink must be different"

    flow = 0
    n = len(graph)
    par = [-1] * n
    q = [0] * n

    while True:
        # BFS to find augmenting path
        par = [-1] * n
        par[source] = source
        ptr = 1
        q[0] = source

        i = 0
        while i < ptr:
            x = q[i]
            for e_first, e_second in graph[x].items():
                if par[e_first] == -1 and e_second > 0:
                    par[e_first] = x
                    q[ptr] = e_first
                    ptr += 1
                    if e_first == sink:
                        break
            else:
                i += 1
                continue
            break

        # No augmenting path found
        if par[sink] == -1:
            return flow

        # Find minimum capacity along path
        inc = float('inf')
        y = sink
        while y != source:
            inc = min(inc, graph[par[y]][y])
            y = par[y]

        # Update flow
        flow += inc
# ... (continued)
```

Euler Walk

Eulerian undirected/directed path/cycle algorithm.

Time: O(V + E)

graph/euler_walk.py

```
"""
Author: Simon Lindholm
Date: 2019-12-31
License: CC0
Source: folklore
Description: Eulerian undirected/directed path/cycle algorithm.
Input should be a list of (dest, global edge index), where
for undirected graphs, forward/backward edges have the same
index.
Returns a list of nodes in the Eulerian path/cycle with src at
both start and end, or
empty list if no cycle/path exists.
Time: O(V + E)
Status: stress-tested
"""

from typing import List, Tuple

def euler_walk(gr: List[List[Tuple[int, int]]], nedges: int, src:
int = 0) -> List[int]:
    """
    Find Eulerian path/cycle in a graph.
    gr[i] = list of (destination, edge_index) pairs
    nedges = total number of edges
    src = starting node
    Returns list of nodes in Eulerian path, or empty list if none
exists
    """
    n = len(gr)
    D = [0] * n
    its = [0] * n
    eu = [0] * nedges
    ret = []
    s = [src]

    D[src] += 1  # to allow Euler paths, not just cycles

    while s:
        x = s[-1]
        it = its[x]
        end = len(gr[x])

        if it == end:
            ret.append(x)
            s.pop()
            continue

        y, e = gr[x][it]
        its[x] += 1

        if not eu[e]:
            D[x] -= 1
            D[y] += 1
            eu[e] = 1
            s.append(y)

    # Check if valid Eulerian path/cycle
    for x in D:
        if x < 0 or len(ret) != nedges + 1:
            return []

    return ret[::-1]
```

Floyd Warshall

Calculates all-pairs shortest path in a directed graph that might have negative edge weights.

Time: O(N^3)

graph/floyd_warshall.py

```
"""
Author: Simon Lindholm
Date: 2016-12-15
License: CC0
Source: http://en.wikipedia.org/wiki/Floyd-Warshall_algorithm
Description: Calculates all-pairs shortest path in a directed
graph that might have negative edge weights.
Input is a distance matrix m, where m[i][j] = inf if i and j are
not adjacent.
As output, m[i][j] is set to the shortest distance between i and
j, inf if no path,
or -inf if the path goes through a negative-weight cycle.
Time: O(N^3)
Status: slightly tested
"""

from typing import List

INF = 10**18

def floyd_warshall(m: List[List[int]]):
    """
    All-pairs shortest path with negative weights.
    m = adjacency matrix (modified in-place)
    m[i][j] = weight of edge i->j, or INF if no edge
    After: m[i][j] = shortest path distance, INF if unreachable,
-INF if negative cycle
    """
    n = len(m)

    # Initialize diagonal
    for i in range(n):
        m[i][i] = min(m[i][i], 0)

    # Floyd-Warshall
    for k in range(n):
        for i in range(n):
            for j in range(n):
                if m[i][k] != INF and m[k][j] != INF:
                    new_dist = max(m[i][k] + m[k][j], -INF)
                    m[i][j] = min(m[i][j], new_dist)

    # Detect and propagate negative cycles
    for k in range(n):
        if m[k][k] < 0:
            for i in range(n):
                for j in range(n):
                    if m[i][k] != INF and m[k][j] != INF:
                        m[i][j] = -INF
```

Global Min Cut

Find a global minimum cut in an undirected graph,

Time: O(V^3)

graph/global_min_cut.py

```
"""
Author: Simon Lindholm
Date: 2021-01-09
License: CC0
Source: https://en.wikipedia.org/wiki/Stoer-Wagner_algorithm
Description: Find a global minimum cut in an undirected graph,
as represented by an adjacency matrix.
Time: O(V^3)
Status: Stress-tested together with GomoryHu
"""

from typing import List, Tuple

def global_min_cut(mat: List[List[int]]) -> Tuple[int,
List[int]]:
    """
    Find global minimum cut in undirected graph.
    mat = adjacency matrix (mat[i][j] = weight of edge i-j)
    Returns (cut_weight, nodes_on_one_side)
    """
    n = len(mat)
    mat = [row[:] for row in mat]  # Copy matrix

    best_weight = float('inf')
    best_cut = []

    # co[i] = list of original nodes merged into node i
    co = [[i] for i in range(n)]

    for ph in range(1, n):
        w = mat[0][:]
        s = 0
        t = 0

        for it in range(n - ph):
            w[t] = float('-inf')
            s = t
            # Find node with maximum weight
            t = max(range(n), key=lambda i: w[i])
            # Update weights
            for i in range(n):
                w[i] += mat[t][i]

        # Check if this is a better cut
        cut_weight = w[t] - mat[t][t]
        if cut_weight < best_weight:
            best_weight = cut_weight
            best_cut = co[t][:]

        # Merge t into s
        co[s].extend(co[t])
        for i in range(n):
            mat[s][i] += mat[t][i]
            mat[i][s] = mat[s][i]
        mat[0][t] = float('-inf')

    return (int(best_weight), best_cut)
```

Gomory Hu

Given a list of edges representing an undirected flow graph,
Time: O(V) Flow Computations

graph/gomory_hu.py

```
"""
Author: chilli, Takanori MAEHARA
Date: 2020-04-03
License: CC0
Source: https://github.com/spaghetti-source/algorithm
Description: Given a list of edges representing an undirected
flow graph,
returns edges of the Gomory-Hu tree. The max flow between any
pair of
vertices is given by minimum edge weight along the Gomory-Hu tree
path.
Time: O(V) Flow Computations
Status: Tested on CERC 2015 J, stress-tested

Details: Uses Gusfield's simplified version of Gomory-Hu.
"""

from typing import List, Tuple
from .push_relabel import PushRelabel

def gomory_hu(N: int, edges: List[Tuple[int, int, int]]) ->
List[Tuple[int, int, int]]:
    """
    Construct Gomory-Hu tree.
    N = number of vertices
    edges = list of (u, v, capacity) tuples
    Returns list of tree edges (u, v, flow_value)
    """
    tree = []
    par = [0] * N

    for i in range(1, N):
        # Run max flow between i and par[i]
        flow = PushRelabel(N)
        for u, v, cap in edges:
            flow.add_edge(u, v, cap, cap)  # Undirected

        flow_value = flow.calc(i, par[i])
        tree.append((i, par[i], flow_value))

        # Update parents based on min cut
        for j in range(i + 1, N):
            if par[j] == par[i] and flow.left_of_min_cut(j):
                par[j] = i

    return tree
```

Hld

Heavy-Light Decomposition. Decomposes a tree into vertex disjoint heavy paths

Time: O((log N)^2)

graph/hld.py

```
"""
Author: Benjamin Qi, Oleksandr Kulkov, chilli
Date: 2020-01-12
License: CC0
Source: https://codeforces.com/blog/entry/53170
Description: Heavy-Light Decomposition. Decomposes a tree into
vertex disjoint heavy paths
and light edges such that the path from any leaf to the root
contains at most log(n)
light edges. Supports path and subtree queries/updates.
VALS_EDGES=True means values are stored in edges, False means
nodes.
Root must be 0.
Time: O((log N)^2)
Status: stress-tested
"""

from typing import List, Callable

class HLD:
    """Heavy-Light Decomposition with segment tree support"""

    def __init__(self, adj: List[List[int]], vals_edges: bool =
False):
        """
        Initialize HLD.
        adj = adjacency list (will be modified)
        vals_edges = True if values on edges, False if on nodes
        """
        self.N = len(adj)
        self.adj = [neighbors[:] for neighbors in adj]  # Deep
copy
        self.vals_edges = vals_edges
        self.tim = 0

        self.par = [-1] * self.N
        self.siz = [1] * self.N
        self.rt = list(range(self.N))  # Root of heavy path
        self.pos = [0] * self.N  # Position in DFS order

        # Initialize segment tree (simple array-based for max
queries)
        self.tree = [0] * (4 * self.N)
        self.lazy = [0] * (4 * self.N)

        self.dfs_sz(0)
        self.dfs_hld(0)

    def dfs_sz(self, v: int):
        """Calculate subtree sizes and reorder children"""
        for i, u in enumerate(self.adj[v]):
            # Remove back edge
            if u in self.adj[v] and v in self.adj[u]:
                self.adj[u].remove(v)

            self.par[u] = v
            self.dfs_sz(u)
            self.siz[v] += self.siz[u]

            # Move heaviest child to front
            if i == 0 or self.siz[u] > self.siz[self.adj[v][0]]:
                if i > 0:
                    self.adj[v][0], self.adj[v][i] = self.adj[v]
[i], self.adj[v][0]

    def dfs_hld(self, v: int):
        """Assign positions and heavy path roots"""
# ... (continued)
```

Hopcroft Karp

Fast bipartite matching algorithm. Graph g should be a list
Time: O(√V E)

graph/hopcroft_karp.py

```
"""
Author: Chen Xing
Date: 2009-10-13
License: CC0
Description: Fast bipartite matching algorithm. Graph g should be a list
of neighbors of the left partition, and btoa should be a vector full of
-1's of the same size as the right partition. Returns the size of
the matching. btoa[i] will be the match for vertex i on the right side,
or -1 if it's not matched.
Usage: btoa = [-1] * m; hopcroft_karp(g, btoa)
Time: O(√V E)
Status: stress-tested by MinimumVertexCover, and tested on
oldkattis.adkbipmatch and SPOJ:MATCHING
"""

from typing import List

def _dfs(a: int, L: int, g: List[List[int]], btoa: List[int], A:
List[int], B: List[int]) -> bool:
    """DFS for Hopcroft-Karp"""
    if A[a] != L:
        return False
    A[a] = -1
    for b in g[a]:
        if B[b] == L + 1:
            B[b] = 0
            if btoa[b] == -1 or _dfs(btoa[b], L + 1, g, btoa, A,
B):
                btoa[b] = a
                return True
    return False

def hopcroft_karp(g: List[List[int]], btoa: List[int]) -> int:
    """
    Fast bipartite matching using Hopcroft-Karp algorithm.
    g[i] = neighbors of node i in left partition
    btoa = matches for right partition (initially all -1)
    Returns size of matching
    """
    res = 0
    A = [0] * len(g)
    B = [0] * len(btoa)
    cur = []
    next_nodes = []

    while True:
        A = [0] * len(g)
        B = [0] * len(btoa)

        # Find starting nodes for BFS (layer 0)
        cur = []
        for a in btoa:
            if a != -1:
                A[a] = -1
        for a in range(len(g)):
            if A[a] == 0:
                cur.append(a)

        # Find all layers using BFS
        lay = 1
        while True:
            islast = False
            next_nodes = []
# ... (continued)
```

Lca

Data structure for computing lowest common ancestors in a tree
Time: O(N log N + Q)

graph/lca.py

```
"""
Author: chilli, pajenegod
Date: 2020-02-20
License: CC0
Source: Folklore
Description: Data structure for computing lowest common ancestors in a tree
(with 0 as root). C should be an adjacency list of the tree, either directed
or undirected.
Time: O(N log N + Q)
Status: stress-tested
"""

from typing import List
import sys
sys.path.append('..')
from data_structures.rmq import RMQ

class LCA:
    def __init__(self, C: List[List[int]]):
        self.T = 0
        self.time = [0] * len(C)
        self.path = []
        self.ret = []
        self.dfs(C, 0, -1)
        self.rmq = RMQ(self.ret)

    def dfs(self, C: List[List[int]], v: int, par: int):
        self.time[v] = self.T
        self.T += 1
        for y in C[v]:
            if y != par:
                self.path.append(v)
                self.ret.append(self.time[v])
                self.dfs(C, y, v)

    def lca(self, a: int, b: int) -> int:
        if a == b:
            return a
        ta = self.time[a]
        tb = self.time[b]
        if ta > tb:
            ta, tb = tb, ta
        return self.path[self.rmq.query(ta, tb)]
```

Maximal Cliques

Runs a callback for all maximal cliques in a graph (given as a
Time: O(3^{n/3}), much faster for sparse graphs

graph/maximal_cliques.py

```
"""
Author: Simon Lindholm
Date: 2018-07-18
License: CC0
Source:
https://en.wikipedia.org/wiki/Bron%E2%80%93Kerbosch_algorithm
Description: Runs a callback for all maximal cliques in a graph (given as a
symmetric adjacency matrix). Callback is given a set representing the maximal clique.
Time: O(3^{n/3}), much faster for sparse graphs
Status: stress-tested
"""

from typing import List, Set, Callable

def maximal_cliques(eds: List[Set[int]], f: Callable[[Set[int]],
None],
                    P: Set[int] = None, X: Set[int] = None, R:
Set[int] = None):
    """
    Find all maximal cliques using Bron-Kerbosch algorithm.
    eds[i] = set of neighbors of node i
    f = callback function called with each maximal clique
    """
    if P is None:
        P = set(range(len(eds)))
    if X is None:
        X = set()
    if R is None:
        R = set()

    if not P:
        if not X:
            f(R)
        return

    # Choose pivot from P U X
    pivot = next(iter(P | X))

    # Iterate through P \ N(pivot)
    cands = P - eds[pivot]
    for v in list(cands):
        maximal_cliques(eds, f, P & eds[v], X & eds[v], R | {v})
        P.remove(v)
        X.add(v)
```

Maximum Clique

Quickly finds a maximum clique of a graph (given as adjacency
Time: Runs in about 1s for n=155 and worst case random graphs (p=.90).

graph/maximum_clique.py

```
"""
Author: chilli, SJTU, Janez Konc
Date: 2019-05-10
License: GPL3+
Source: Wikipedia, https://gitlab.com/janezkonc/mcqd
Description: Quickly finds a maximum clique of a graph (given as
adjacency
matrix). Can be used to find a maximum independent set by finding
a clique
of the complement graph.
Time: Runs in about 1s for n=155 and worst case random graphs
(p=.90).
Runs faster for sparse graphs.
Status: stress-tested
"""

from typing import List

class MaxClique:
    """Find maximum clique in undirected graph"""

    def __init__(self, edges: List[List[bool]]):
        """
        Initialize with adjacency matrix.
        edges[i][j] = True if edge between i and j exists
        """
        self.limit = 0.025
        self.pk = 0
        self.e = edges
        self.n = len(edges)

        self.V = [{'i': i, 'd': 0} for i in range(self.n)]
        self.C = [[] for _ in range(self.n + 1)]
        self.qmax = []
        self.q = []
        self.S = [0] * (self.n + 1)
        self.old = [0] * (self.n + 1)

    def _init(self, R: List[dict]):
        """Initialize vertex degrees"""
        for v in R:
            v['d'] = 0

        for v in R:
            for u in R:
                if self.e[v['i']][u['i']]:
                    v['d'] += 1

        R.sort(key=lambda x: x['d'], reverse=True)

        mxD = R[0]['d'] if R else 0
        for i, v in enumerate(R):
            v['d'] = min(i, mxD) + 1

    def _expand(self, R: List[dict], lev: int = 1):
        """Expand current clique"""
        self.S[lev] += self.S[lev - 1] - self.old[lev]
        self.old[lev] = self.S[lev - 1]

        while R:
            if len(self.q) + R[-1]['d'] <= len(self.qmax):
                return

# ... (continued)
```

Min Cost Max Flow

Min-cost max-flow.

Time: O(F E log(V)) where F is max flow. O(VE) for set_pi.

graph/min_cost_max_flow.py

```
"""
Author: Stanford
Date: Unknown
Source: Stanford Notebook
Description: Min-cost max-flow.
If costs can be negative, call set_pi before maxflow, but note
that negative cost cycles are not supported.
To obtain the actual flow, look at positive values only.
Status: Tested on kattis:mincostmaxflow, stress-tested against
another implementation
Time: O(F E log(V)) where F is max flow. O(VE) for set_pi.
"""

import heapq
from typing import List, Tuple

INF = 10**18

class MinCostMaxFlow:
    class Edge:
        def __init__(self, from_node: int, to: int, rev: int,
cap: int, cost: int):
            self.from_node = from_node
            self.to = to
            self.rev = rev
            self.cap = cap
            self.cost = cost
            self.flow = 0

    def __init__(self, N: int):
        self.N = N
        self.ed = [[] for _ in range(N)]
        self.seen = [False] * N
        self.dist = [INF] * N
        self.pi = [0] * N
        self.par = [None] * N

    def add_edge(self, from_node: int, to: int, cap: int, cost:
int):
        """Add edge with capacity and cost"""
        if from_node == to:
            return
        self.ed[from_node].append(self.Edge(from_node, to,
len(self.ed[to]), cap, cost))
        self.ed[to].append(self.Edge(to, from_node,
len(self.ed[from_node]) - 1, 0, -cost))

    def path(self, s: int):
        """Find shortest path using Dijkstra with potentials"""
        self.seen = [False] * self.N
        self.dist = [INF] * self.N
        self.dist[s] = 0

        # Priority queue: (distance, node)
        pq = [(0, s)]

        while pq:
            d, u = heapq.heappop(pq)
            if self.seen[u]:
                continue
            self.seen[u] = True
            di = self.dist[u] + self.pi[u]

            for e in self.ed[u]:
                if not self.seen[e.to]:
                    val = di - self.pi[e.to] + e.cost
# ... (continued)
```

Minimum Vertex Cover

Finds a minimum vertex cover in a bipartite graph.

graph/minimum_vertex_cover.py

```
"""
Author: Johan Sannemo, Simon Lindholm
Date: 2016-12-15
License: CC0
Description: Finds a minimum vertex cover in a bipartite graph.
The size is the same as the size of a maximum matching, and
the complement is a maximum independent set.
Status: stress-tested
"""

from typing import List
from .dfs_matching import dfs_matching

def minimum_vertex_cover(g: List[List[int]], n: int, m: int) ->
List[int]:
    """
    Find minimum vertex cover in bipartite graph.
    g = adjacency list for left partition (size n)
    m = size of right partition
    Returns list of vertices in the cover (left vertices as 0..n-
1, right as n..n+m-1)
    """
    match = [-1] * m
    res = dfs_matching(g, match)

    lfound = [True] * n
    seen = [False] * m

    for it in match:
        if it != -1:
            lfound[it] = False

    q = [i for i in range(n) if lfound[i]]

    while q:
        i = q.pop()
        lfound[i] = True
        for e in g[i]:
            if not seen[e] and match[e] != -1:
                seen[e] = True
                q.append(match[e])

    cover = []
    for i in range(n):
        if not lfound[i]:
            cover.append(i)
    for i in range(m):
        if seen[i]:
            cover.append(n + i)

    assert len(cover) == res
    return cover
```

Push Relabel

Push-relabel using the highest label selection rule and the gap heuristic. Quite fast in practice.

Time: O(V^2√E)

graph/push_relabel.py

```
"""
Author: Simon Lindholm
Date: 2015-02-24
License: CC0
Source: Wikipedia, tinyKACTL
Description: Push-relabel using the highest label selection rule
and the gap heuristic. Quite fast in practice.
To obtain the actual flow, look at positive values only.
Time: O(V^2√E)
Status: Tested on Kattis and SPOJ, and stress-tested
"""

from typing import List

class PushRelabel:
    class Edge:
        def __init__(self, dest: int, back: int, c: int):
            self.dest = dest
            self.back = back
            self.f = 0
            self.c = c

    def __init__(self, n: int):
        self.g = [[] for _ in range(n)]
        self.ec = [0] * n
        self.cur = [0] * n
        self.hs = [[] for _ in range(2 * n)]
        self.H = [0] * n

    def add_edge(self, s: int, t: int, cap: int, rcap: int = 0):
        """Add edge with capacity and optional reverse
capacity"""
        if s == t:
            return
        self.g[s].append(self.Edge(t, len(self.g[t]), cap))
        self.g[t].append(self.Edge(s, len(self.g[s]) - 1, rcap))

    def add_flow(self, e: Edge, f: int):
        """Push flow through edge"""
        back = self.g[e.dest][e.back]
        if not self.ec[e.dest] and f:
            self.hs[self.H[e.dest]].append(e.dest)
        e.f += f
        e.c -= f
        self.ec[e.dest] += f
        back.f -= f
        back.c += f
        self.ec[back.dest] -= f

    def calc(self, s: int, t: int) -> int:
        """Calculate max flow from s to t"""
        v = len(self.g)
        self.H[s] = v
        self.ec[t] = 1
        co = [0] * (2 * v)
        co[0] = v - 1

        for i in range(v):
            self.cur[i] = 0

        for e in self.g[s]:
            self.add_flow(e, e.c)
# ... (continued)
```

Scc

Finds strongly connected components in a

Time: O(E + V)

graph/scc.py

```
"""
Author: Lukas Polacek
Date: 2009-10-28
License: CC0
Source: Czech graph algorithms book, by Demel. (Tarjan's
algorithm)
Description: Finds strongly connected components in a
directed graph. If vertices u, v belong to the same component,
we can reach u from v and vice versa.
Usage: scc(graph, callback) visits all components
in reverse topological order. comp[i] holds the component
index of a node (a component only has edges to components with
lower index). Returns (comp, ncomps).
Time: O(E + V)
Status: Bruteforce-tested for N <= 5
"""

from typing import List, Callable

class SCCState:
    def __init__(self, n: int):
        self.val = [0] * n
        self.comp = [-1] * n
        self.z = []
        self.cont = []
        self.Time = 0
        self.ncomps = 0

def _dfs(j: int, g: List[List[int]], f: Callable, state:
SCCState) -> int:
    """DFS helper for SCC"""
    state.Time += 1
    low = state.val[j] = state.Time
    state.z.append(j)

    for e in g[j]:
        if state.comp[e] < 0:
            low = min(low, state.val[e] if state.val[e] else
_dfs(e, g, f, state))

    if low == state.val[j]:
        while True:
            x = state.z.pop()
            state.comp[x] = state.ncomps
            state.cont.append(x)
            if x == j:
                break
        f(state.cont)
        state.cont = []
        state.ncomps += 1

    state.val[j] = low
    return low

def scc(g: List[List[int]], f: Callable):
    """Find strongly connected components"""
    n = len(g)
    state = SCCState(n)

    for i in range(n):
        if state.comp[i] < 0:
            _dfs(i, g, f, state)

# ... (continued)
```

Topo Sort

Topological sorting. Given is an oriented graph.

Time: O(|V|+|E|)

graph/topo_sort.py

```
"""
Author: Unknown
Date: 2002-09-13
Source: predates tinyKACTL
Description: Topological sorting. Given is an oriented graph.
Output is an ordering of vertices, such that there are edges only
from left to right.
If there are cycles, the returned list will have size smaller
than n -- nodes reachable
from cycles will not be returned.
Time: O(|V|+|E|)
Status: stress-tested
"""

from typing import List

def topo_sort(gr: List[List[int]]) -> List[int]:
    """
    Topological sort using Kahn's algorithm.
    gr[i] = list of neighbors of node i
    Returns topologically sorted list (empty or partial if cycle
exists)
    """
    indeg = [0] * len(gr)

    # Calculate in-degrees
    for neighbors in gr:
        for x in neighbors:
            indeg[x] += 1

    # Start with nodes having in-degree 0
    q = [i for i in range(len(gr)) if indeg[i] == 0]

    # Process queue
    j = 0
    while j < len(q):
        for x in gr[q[j]]:
            indeg[x] -= 1
            if indeg[x] == 0:
                q.append(x)
        j += 1

    return q
```

Two Sat

Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem,

Time: O(N+E), where N is the number of boolean variables, and E is the number of clauses.

graph/two_sat.py

```
"""
Author: Emil Lenngren, Simon Lindholm
Date: 2011-11-29
License: CC0
Source: folklore
Description: Calculates a valid assignment to boolean variables
a, b, c,... to a 2-SAT problem,
so that an expression of the type (a||b)&&(!a||c)&&(d||!b)&&...
becomes true, or reports that it is unsatisfiable.
Negated variables are represented by bit-inversions (~x).
Usage:
  ts = TwoSat(number of boolean variables)
  ts.either(0, ~3)  # Var 0 is true or var 3 is false
  ts.set_value(2)  # Var 2 is true
  ts.at_most_one([0,~1,2])  # <= 1 of vars 0, ~1 and 2 are true
  ts.solve()  # Returns true iff it is solvable
  ts.values[0..N-1] holds the assigned values to the vars
Time: O(N+E), where N is the number of boolean variables, and E
is the number of clauses.
Status: stress-tested
"""

from typing import List

class TwoSat:
    def __init__(self, n: int = 0):
        self.N = n
        self.gr = [[] for _ in range(2 * n)]
        self.values = []

    def add_var(self) -> int:
        """Add a new boolean variable (optional)"""
        self.gr.append([])
        self.gr.append([])
        self.N += 1
        return self.N - 1

    def either(self, f: int, j: int):
        """Add clause: f OR j must be true"""
        # Convert negative numbers to bit notation
        f = max(2 * f, -1 - 2 * f)
        j = max(2 * j, -1 - 2 * j)
        self.gr[f].append(j ^ 1)
        self.gr[j].append(f ^ 1)

    def set_value(self, x: int):
        """Force variable x to be true"""
        self.either(x, x)

    def at_most_one(self, li: List[int]):
        """At most one of the variables in li can be true
(optional)"""
        if len(li) <= 1:
            return
        cur = ~li[0]
        for i in range(2, len(li)):
            next_var = self.add_var()
            self.either(cur, ~li[i])
            self.either(cur, next_var)
            self.either(~li[i], next_var)
            cur = ~next_var
        self.either(cur, ~li[1])

# ... (continued)
```

Weighted Matching

Given a weighted bipartite graph, matches every node on

Time: O(N^2 M)

graph/weighted_matching.py

```
"""
Author: Benjamin Qi, chilli
Date: 2020-04-04
License: CC0
Source:
https://github.com/bqi343/USACO/blob/master/Implementations/conte
nt/graphs%20(12)/Matching/Hungarian.h
Description: Given a weighted bipartite graph, matches every node
on
the left with a node on the right such that no
nodes are in two matchings and the sum of the edge weights is
minimal. Takes
cost[N][M], where cost[i][j] = cost for L[i] to be matched with
R[j] and
returns (min cost, match), where L[i] is matched with
R[match[i]]. Negate costs for max cost. Requires N <= M.
Time: O(N^2 M)
Status: Tested on kattis:cordonbleu, stress-tested
"""

from typing import List, Tuple

def hungarian(a: List[List[int]]) -> Tuple[int, List[int]]:
    """
    Hungarian algorithm for minimum cost bipartite matching.
    a[i][j] = cost for L[i] to be matched with R[j]
    Returns (min_cost, match) where L[i] is matched with
R[match[i]]
    """
    if not a:
        return (0, [])

    n = len(a) + 1
    m = len(a[0]) + 1
    u = [0] * n
    v = [0] * m
    p = [0] * m
    ans = [0] * (n - 1)

    for i in range(1, n):
        p[0] = i
        j0 = 0  # add "dummy" worker 0
        dist = [float('inf')] * m
        pre = [-1] * m
        done = [False] * (m + 1)

        while True:  # Dijkstra
            done[j0] = True
            i0 = p[j0]
            delta = float('inf')
            j1 = 0

            for j in range(1, m):
                if not done[j]:
                    cur = a[i0 - 1][j - 1] - u[i0] - v[j]
                    if cur < dist[j]:
                        dist[j] = cur
                        pre[j] = j0
                    if dist[j] < delta:
                        delta = dist[j]
                        j1 = j

            for j in range(m):
                if done[j]:
                    u[p[j]] += delta
                    v[j] -= delta
# ... (continued)
```

String Algorithms

Aho Corasick

Aho-Corasick automaton, used for multiple pattern matching.

Time: construction takes O(26N), where N = sum of length of patterns.

strings/aho_corasick.py

```
"""
Author: Simon Lindholm
Date: 2015-02-18
License: CC0
Source: marian's (TC) code
Description: Aho-Corasick automaton, used for multiple pattern
matching.
Initialize with AhoCorasick(patterns); the automaton start node
will be at index 0.
find(word) returns for each position the index of the longest
word that ends there, or -1 if none.
findAll(patterns, word) finds all words (up to N√N many if no
duplicate patterns)
that start at each position (shortest first).
Duplicate patterns are allowed; empty patterns are not.
Time: construction takes O(26N), where N = sum of length of
patterns.
find(x) is O(N), where N = length of x. findAll is O(NM).
Status: stress-tested
"""

from typing import List
from collections import deque

class AhoCorasick:
    ALPHA = 26
    FIRST = ord('A')  # Change this for different alphabets

    class Node:
        def __init__(self):
            self.back = 0
            self.next = [-1] * AhoCorasick.ALPHA
            self.start = -1
            self.end = -1
            self.nmatches = 0

    def __init__(self, patterns: List[str]):
        self.N = [self.Node()]
        self.backp = []

        # Insert all patterns
        for j, pattern in enumerate(patterns):
            self._insert(pattern, j)

        # Build failure links
        self.N[0].back = len(self.N)
        self.N.append(self.Node())
        self.N[-1].back = 0

        q = deque([0])
        while q:
            n = q.popleft()
            prev = self.N[n].back

            for i in range(self.ALPHA):
                ed = self.N[n].next[i]
                y = self.N[prev].next[i]

                if ed == -1:
                    self.N[n].next[i] = y
                else:
                    self.N[ed].back = y
                    if self.N[ed].end == -1:
                        self.N[ed].end = self.N[y].end
                    else:
# ... (continued)
```

Hashing

Self-explanatory methods for string hashing.

strings/hashing.py

```
"""
Author: Simon Lindholm
Date: 2015-03-15
License: CC0
Source: own work
Description: Self-explanatory methods for string hashing.
Status: stress-tested
"""

from typing import List

# Using simple modulo for Python version
# In Python, we can use native integers which handle large
numbers well
MOD = (1 << 64) - 1  # 2^64 - 1
C = int(1e11) + 3  # base for hashing

class H:
    """Hash value with arithmetic mod 2^64-1"""
    def __init__(self, x: int = 0):
        self.x = x % MOD

    def __add__(self, o):
        result = (self.x + o.x) % MOD
        return H(result)

    def __sub__(self, o):
        result = (self.x - o.x) % MOD
        return H(result)

    def __mul__(self, o):
        result = (self.x * o.x) % MOD
        return H(result)

    def get(self) -> int:
        return self.x

    def __eq__(self, o):
        return self.get() == o.get()

    def __lt__(self, o):
        return self.get() < o.get()

    def __hash__(self):
        return hash(self.x)

class HashInterval:
    """Compute hashes for all prefixes of a string"""
    def __init__(self, s: str):
        n = len(s)
        self.ha = [H(0) for _ in range(n + 1)]
        self.pw = [H(0) for _ in range(n + 1)]
        self.pw[0] = H(1)

        for i in range(n):
            self.ha[i + 1] = self.ha[i] * H(C) + H(ord(s[i]))
            self.pw[i + 1] = self.pw[i] * H(C)

    def hash_interval(self, a: int, b: int) -> H:
        """Hash substring [a, b)"""
        return self.ha[b] - self.ha[a] * self.pw[b - a]
# ... (continued)
```

Hashing Codeforces

Various methods for string hashing with dual-modulo approach.

strings/hashing_codeforces.py

```
"""
Author: Simon Lindholm
Date: 2015-03-15
License: CC0
Source: own work
Description: Various methods for string hashing with dual-modulo
approach.
Use on Codeforces, which lacks 64-bit support and where solutions
can be hacked.
Status: stress-tested
"""

import time

# Initialize random constant C (should be done once at startup)
C = int(time.time() * 1000000) % 1000000007

class Hash:
    """Dual-modulo hash for anti-hack protection"""
    MOD1 = 1000000007
    MOD2 = 1000000009

    def __init__(self, x=0, y=0):
        self.x = x % self.MOD1
        self.y = y % self.MOD2

    def __add__(self, other):
        x = (self.x + other.x) % self.MOD1
        y = (self.y + other.y) % self.MOD2
        return Hash(x, y)

    def __sub__(self, other):
        x = (self.x - other.x + self.MOD1) % self.MOD1
        y = (self.y - other.y + self.MOD2) % self.MOD2
        return Hash(x, y)

    def __mul__(self, other):
        x = (self.x * other.x) % self.MOD1
        y = (self.y * other.y) % self.MOD2
        return Hash(x, y)

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __hash__(self):
        return self.x ^ (self.y << 21)

    def __lt__(self, other):
        return (self.x, self.y) < (other.x, other.y)

    def __repr__(self):
        return f"Hash({self.x}, {self.y})"

class HashInterval:
    """Compute hash of any substring in O(1) after O(n)
preprocessing"""

    def __init__(self, s: str):
        n = len(s)
        self.ha = [Hash(0, 0)] * (n + 1)
        self.pw = [Hash(0, 0)] * (n + 1)
        self.pw[0] = Hash(1, 1)

# ... (continued)
```

Kmp

pi[x] computes the length of the longest prefix of s that ends at x,

Time: O(n)

strings/kmp.py

```
"""
Author: Johan Sannemo
Date: 2016-12-15
License: CC0
Description: pi[x] computes the length of the longest prefix of s
that ends at x,
other than s[0...x] itself (abacaba -> 0010123).
Can be used to find all occurrences of a string.
Time: O(n)
Status: Tested on kattis:stringmatching
"""

from typing import List

def pi(s: str) -> List[int]:
    """Compute KMP prefix function"""
    p = [0] * len(s)
    for i in range(1, len(s)):
        g = p[i - 1]
        while g and s[i] != s[g]:
            g = p[g - 1]
        p[i] = g + (1 if s[i] == s[g] else 0)
    return p

def match(s: str, pat: str) -> List[int]:
    """Find all occurrences of pat in s"""
    p = pi(pat + '\0' + s)
    res = []
    for i in range(len(p) - len(s), len(p)):
        if p[i] == len(pat):
            res.append(i - 2 * len(pat))
    return res
```

Manacher

For each position in a string, computes p[0][i] = half length of

Time: O(N)

strings/manacher.py

```
"""
Author: User adamant on CodeForces
Source: http://codeforces.com/blog/entry/12143
Description: For each position in a string, computes p[0][i] =
half length of
longest even palindrome around pos i, p[1][i] = longest odd (half
rounded down).
Time: O(N)
Status: Stress-tested
"""

from typing import List

def manacher(s: str) -> List[List[int]]:
    """
    Returns [even_palindromes, odd_palindromes]
    even_palindromes[i] = half length of longest even palindrome
centered at position i
    odd_palindromes[i] = half length (rounded down) of longest
odd palindrome centered at position i
    """
    n = len(s)
    p = [[0] * (n + 1), [0] * n]

    for z in range(2):
        i = 0
        l = 0
        r = 0
        while i < n:
            t = r - i + (1 if z == 0 else 0)
            if i < r:
                p[z][i] = min(t, p[z][l + t])

            L = i - p[z][i]
            R = i + p[z][i] - (1 if z == 0 else 0)

            while L >= 1 and R + 1 < n and s[L - 1] == s[R + 1]:
                p[z][i] += 1
                L -= 1
                R += 1

            if R > r:
                l = L
                r = R

            i += 1

    return p
```

Min Rotation

Finds the lexicographically smallest rotation of a string.

Time: O(N)

strings/min_rotation.py

```
"""
Author: Stjepan Glavina
License: Unlicense
Source:
https://github.com/stjepang/snippets/blob/master/min_rotation.cpp
Description: Finds the lexicographically smallest rotation of a
string.
Time: O(N)
Status: Stress-tested
"""

def min_rotation(s: str) -> int:
    """Find index of lexicographically smallest rotation"""
    a = 0
    N = len(s)
    s = s + s
    for b in range(N):
        for k in range(N):
            if a + k == b or s[a + k] < s[b + k]:
                b += max(0, k - 1)
                break
            if s[a + k] > s[b + k]:
                a = b
                break
    return a
```

Suffix Array

Builds suffix array for a string.

Time: O(n log n)

strings/suffix_array.py

```
"""
Author: 罗穗骞, chilli
Date: 2019-04-11
License: Unknown
Source: Suffix array - a powerful tool for dealing with strings
(Chinese IOI National team training paper, 2009)
Description: Builds suffix array for a string.
sa[i] is the starting index of the suffix which is i'th in the
sorted suffix array.
The returned vector is of size n+1, and sa[0] = n.
The lcp array contains longest common prefixes for neighbouring
strings in the suffix array:
lcp[i] = lcp(sa[i], sa[i-1]), lcp[0] = 0.
The input string must not contain any nul chars.
Time: O(n log n)
Status: stress-tested
"""

from typing import List

class SuffixArray:
    def __init__(self, s: str, lim: int = 256):
        # Convert string to list and append null terminator
        s_list = [ord(c) for c in s] + [0]
        n = len(s_list)
        k = 0

        x = s_list[:]
        y = [0] * n
        ws = [0] * max(n, lim)

        self.sa = list(range(n))
        self.lcp = [0] * n

        j = 0
        p = 0
        while p < n:
            # Radix sort
            p = j
            y[p:p + n - j] = list(range(n - j, n))
            p += n - j
            for i in range(n):
                if self.sa[i] >= j:
                    y[p] = self.sa[i] - j
                    p += 1

            ws[:lim] = [0] * lim
            for i in range(n):
                ws[x[i]] += 1
            for i in range(1, lim):
                ws[i] += ws[i - 1]
            for i in range(n - 1, -1, -1):
                ws[x[y[i]]] -= 1
                self.sa[ws[x[y[i]]]] = y[i]

            x, y = y, x
            p = 1
            x[self.sa[0]] = 0
            for i in range(1, n):
                a = self.sa[i - 1]
                b = self.sa[i]
                if y[a] == y[b] and a + j < n and b + j < n and
y[a + j] == y[b + j]:
# ... (continued)
```

Suffix Tree

Ukkonen's algorithm for online suffix tree construction.

Time: O(26N) for alphabet size 26

strings/suffix_tree.py

```
"""
Author: Unknown
Date: 2017-05-15
Source: https://e-maxx.ru/algo/ukkonen
Description: Ukkonen's algorithm for online suffix tree
construction.
Each node contains indices [l, r) into the string, and a list of
child nodes.
Suffixes are given by traversals of this tree, joining [l, r)
substrings.
The root is 0 (has l = -1, r = 0), non-existent children are -1.
To get a complete tree, append a dummy symbol -- otherwise it may
contain
an incomplete path (still useful for substring matching).
Time: O(26N) for alphabet size 26
Status: stress-tested a bit
"""

class SuffixTree:
    """Ukkonen's suffix tree for pattern matching"""

    def __init__(self, s: str, alpha_size: int = 26):
        """
        Build suffix tree for string s.
        alpha_size = alphabet size (26 for lowercase letters)
        """
        N = len(s) * 2 + 10
        self.s = s
        self.alpha_size = alpha_size

        # Arrays for suffix tree structure
        self.t = [[-1] * alpha_size for _ in range(N)]  #
transitions
        self.l = [0] * N  # left bound of substring
        self.r = [len(s)] * N  # right bound
        self.p = [0] * N  # parent
        self.s_link = [0] * N  # suffix link

        self.v = 0  # current node
        self.q = 0  # current position in edge
        self.m = 2  # next free node

        # Initialize root
        self.l[0] = self.l[1] = -1
        self.r[0] = self.r[1] = 0
        self.p[0] = self.p[1] = 0
        for c in range(alpha_size):
            self.t[1][c] = 0
        self.s_link[0] = 1

        # Build tree character by character
        for i, ch in enumerate(s):
            self._ukkonen_add(i, ord(ch) - ord('a'))

    def _ukkonen_add(self, i: int, c: int):
        """Add character at position i"""
        while True:  # suff label
            if self.r[self.v] <= self.q:
                if self.t[self.v][c] == -1:
                    self.t[self.v][c] = self.m
                    self.l[self.m] = i
                    self.p[self.m] = self.v
                    self.m += 1
                    self.v = self.s_link[self.v]
                    self.q = self.r[self.v]
# ... (continued)
```

Zfunc

z[i] computes the length of the longest common prefix of s[i:] and s,

Time: O(n)

strings/zfunc.py

```
"""
Author: chilli
License: CC0
Description: z[i] computes the length of the longest common
prefix of s[i:] and s,
except z[0] = 0. (abacaba -> 0010301)
Time: O(n)
Status: stress-tested
"""

from typing import List

def Z(S: str) -> List[int]:
    """Compute Z-function for string S"""
    z = [0] * len(S)
    l = -1
    r = -1
    for i in range(1, len(S)):
        z[i] = 0 if i >= r else min(r - i, z[i - l])
        while i + z[i] < len(S) and S[i + z[i]] == S[z[i]]:
            z[i] += 1
        if i + z[i] > r:
            l = i
            r = i + z[i]
    return z
```

Geometry

Angle

A class for ordering angles (as represented by int points and

geometry/angle.py

```
"""
Author: Simon Lindholm
Date: 2015-01-31
License: CC0
Source: me
Description: A class for ordering angles (as represented by int
points and
a number of rotations around the origin). Useful for rotational
sweeping.
Sometimes also represents points or vectors.
Status: Used, works well
"""

from typing import Tuple

class Angle:
    def __init__(self, x: int, y: int, t: int = 0):
        self.x = x
        self.y = y
        self.t = t  # number of rotations

    def __sub__(self, b: 'Angle') -> 'Angle':
        return Angle(self.x - b.x, self.y - b.y, self.t)

    def half(self) -> int:
        """Which half-plane is this angle in?"""
        assert self.x or self.y
        return 1 if self.y < 0 or (self.y == 0 and self.x < 0)
else 0

    def t90(self) -> 'Angle':
        """Rotate 90 degrees"""
        return Angle(-self.y, self.x, self.t + (1 if self.half()
and self.x >= 0 else 0))

    def t180(self) -> 'Angle':
        """Rotate 180 degrees"""
        return Angle(-self.x, -self.y, self.t + self.half())

    def t360(self) -> 'Angle':
        """Rotate 360 degrees"""
        return Angle(self.x, self.y, self.t + 1)

    def __lt__(self, b: 'Angle') -> bool:
        return (self.t, self.half(), self.y * b.x) < (b.t,
b.half(), self.x * b.y)

    def __add__(self, b: 'Angle') -> 'Angle':
        """Add point a + vector b"""
        r = Angle(self.x + b.x, self.y + b.y, self.t)
        if self.t180() < r:
            r.t -= 1
        return r.t360() if r.t180() < self else r

    def dist2(self) -> int:
        """Squared distance from origin"""
        return self.x * self.x + self.y * self.y

def segment_angles(a: Angle, b: Angle) -> Tuple[Angle, Angle]:
    """
    Given two points, calculate the smallest angle between them,
    i.e., the angle that covers the defined line segment.
    """
    if b < a:
        a, b = b, a
# ... (continued)
```

Circle Intersection

Computes the pair of points at which two circles intersect.

geometry/circle_intersection.py

```
"""
Author: Simon Lindholm
Date: 2015-09-01
License: CC0
Description: Computes the pair of points at which two circles
intersect.
Returns None in case of no intersection.
Status: stress-tested
"""

import math
from typing import Optional, Tuple
from .point import Point

def circle_intersection(a: Point, b: Point, r1: float, r2: float)
-> Optional[Tuple[Point, Point]]:
    """
    Find intersection points of two circles.
    a, b = centers of circles
    r1, r2 = radii
    Returns (point1, point2) or None if no intersection
    """
    if a == b:
        assert r1 != r2
        return None

    vec = b - a
    d2 = vec.dist2()
    sum_r = r1 + r2
    dif_r = r1 - r2

    if sum_r * sum_r < d2 or dif_r * dif_r > d2:
        return None

    p = (d2 + r1 * r1 - r2 * r2) / (d2 * 2)
    h2 = r1 * r1 - p * p * d2

    mid = a + vec * p
    per = vec.perp() * math.sqrt(max(0, h2) / d2)

    return (mid + per, mid - per)
```

Circle Line

Finds the intersection between a circle and a line.

geometry/circle_line.py

```
"""
Author: Victor Lecomte, chilli
Date: 2019-10-29
License: CC0
Source: https://vlecomte.github.io/cp-geo.pdf
Description: Finds the intersection between a circle and a line.
Returns a list of either 0, 1, or 2 intersection points.
Status: unit tested
"""

import math
from typing import List
from .point import Point

def circle_line(c: Point, r: float, a: Point, b: Point) ->
List[Point]:
    """
    Find intersection of circle with center c and radius r with
line through a and b.
    Returns list of intersection points (0, 1, or 2 points).
    """
    ab = b - a
    p = a + ab * (c - a).dot(ab) / ab.dist2()
    s = a.cross(b, c)
    h2 = r * r - s * s / ab.dist2()

    if h2 < 0:
        return []
    if h2 == 0:
        return [p]

    h = ab.unit() * math.sqrt(h2)
    return [p - h, p + h]
```

Circle Polygon Intersection

Returns the area of the intersection of a circle with a ccw polygon.

Time: O(n)

geometry/circle_polygon_intersection.py

```
"""
Author: chilli, Takanori MAEHARA
Date: 2019-10-31
License: CC0
Source: https://github.com/spaghetti-source/algorithm
Description: Returns the area of the intersection of a circle
with a ccw polygon.
Time: O(n)
Status: Tested on GNYR 2019 Gerrymandering, stress-tested
"""

import math
from typing import List
from .point import Point

def circle_polygon_intersection(c: Point, r: float, ps:
List[Point]) -> float:
    """
    Calculate area of intersection between circle and polygon.
    c = circle center
    r = circle radius
    ps = polygon vertices (CCW order)
    """
    def arg(p: Point, q: Point) -> float:
        """Angle between vectors"""
        return math.atan2(p.cross(q), p.dot(q))

    def tri(p: Point, q: Point) -> float:
        """Area contribution of triangle/circular segment"""
        r2 = r * r / 2
        d = q - p
        a = d.dot(p) / d.dist2()
        b = (p.dist2() - r * r) / d.dist2()
        det = a * a - b

        if det <= 0:
            return arg(p, q) * r2

        s = max(0.0, -a - math.sqrt(det))
        t = min(1.0, -a + math.sqrt(det))

        if t < 0 or 1 <= s:
            return arg(p, q) * r2

        u = p + d * s
        v = p + d * (t - 1)
        return arg(p, u) * r2 + u.cross(v) / 2 + arg(v, q) * r2

    total = 0.0
    for i in range(len(ps)):
        total += tri(ps[i] - c, ps[(i + 1) % len(ps)] - c)

    return total
```

Circle Tangents

Finds the external tangents of two circles, or internal if r2 is negated.

geometry/circle_tangents.py

```
"""
Author: Victor Lecomte, chilli
Date: 2019-10-31
License: CC0
Source: https://vlecomte.github.io/cp-geo.pdf
Description: Finds the external tangents of two circles, or
internal if r2 is negated.
Can return 0, 1, or 2 tangents.
.first and .second give the tangency points at circle 1 and 2
respectively.
To find the tangents of a circle with a point set r2 to 0.
Status: tested
"""

import math
from typing import List, Tuple
from .point import Point

def circle_tangents(c1: Point, r1: float, c2: Point, r2: float) -
> List[Tuple[Point, Point]]:
    """
    Find tangent lines of two circles.
    Returns list of (point_on_c1, point_on_c2) pairs.
    For internal tangents, negate r2.
    For tangents from point to circle, set r2=0.
    """
    d = c2 - c1
    dr = r1 - r2
    d2 = d.dist2()
    h2 = d2 - dr * dr

    if d2 == 0 or h2 < 0:
        return []

    out = []
    for sign in [-1, 1]:
        v = (d * dr + d.perp() * math.sqrt(h2) * sign) / d2
        out.append((c1 + v * r1, c2 + v * r2))

    if h2 == 0:
        out.pop()

    return out
```

Circumcircle

geometry/circumcircle.py

```
"""
Author: Ulf Lundstrom
Date: 2009-04-11
License: CC0
Source: http://en.wikipedia.org/wiki/Circumcircle
Description:
The circumcircle of a triangle is the circle intersecting all
three vertices.
cc_radius returns the radius and cc_center returns the center.
Status: tested
"""

from .point import Point

def cc_radius(A: Point, B: Point, C: Point) -> float:
    """Calculate radius of circumcircle of triangle ABC"""
    return ((B - A).dist() * (C - B).dist() * (A - C).dist() /
            abs((B - A).cross(C - A)) / 2)

def cc_center(A: Point, B: Point, C: Point) -> Point:
    """Calculate center of circumcircle of triangle ABC"""
    b = C - A
    c = B - A
    return A + (b * c.dist2() - c * b.dist2()).perp() /
b.cross(c) / 2
```

Closest Pair

Finds the closest pair of points.

Time: O(n log n)

geometry/closest_pair.py

```
"""
Author: Simon Lindholm
Date: 2019-04-17
License: CC0
Source: https://codeforces.com/blog/entry/58747
Description: Finds the closest pair of points.
Time: O(n log n)
Status: stress-tested
"""

import math
from typing import List, Tuple
from .point import Point

def closest_pair(v: List[Point]) -> Tuple[Point, Point]:
    """
    Find the closest pair of points.
    Returns (point1, point2)
    """
    assert len(v) > 1

    S = set()
    v = sorted(v, key=lambda p: p.y)

    ret_dist2 = float('inf')
    ret_points = (Point(0, 0), Point(0, 0))
    j = 0

    for p in v:
        d_x = 1 + int(math.sqrt(ret_dist2))

        # Remove points that are too far below
        while v[j].y <= p.y - d_x:
            S.discard((v[j].x, v[j].y))
            j += 1

        # Check nearby points
        for x, y in sorted(S):
            q = Point(x, y)
            if abs(q.x - p.x) <= d_x and abs(q.y - p.y) <= d_x:
                dist2 = (q - p).dist2()
                if dist2 < ret_dist2:
                    ret_dist2 = dist2
                    ret_points = (q, p)

        S.add((p.x, p.y))

    return ret_points
```

Convex Hull

Returns a vector of the points of the convex hull in counter-clockwise order.

Time: O(n log n)

geometry/convex_hull.py

```
"""
Author: Stjepan Glavina, chilli
Date: 2019-05-05
License: Unlicense
Source:
https://github.com/stjepang/snippets/blob/master/convex_hull.cpp
Description: Returns a vector of the points of the convex hull in
counter-clockwise order.
Points on the edge of the hull between two other points are not
considered part of the hull.
Time: O(n log n)
Status: stress-tested, tested with kattis:convexhull
"""

from typing import List
from .point import Point

def convex_hull(pts: List[Point]) -> List[Point]:
    """Compute convex hull of points"""
    if len(pts) <= 1:
        return pts

    pts = sorted(pts)
    h = [None] * (len(pts) + 1)
    s = 0
    t = 0

    for it in range(2):
        for p in pts:
            while t >= s + 2 and h[t - 2].cross(h[t - 1], p) <=
0:
                t -= 1
            h[t] = p
            t += 1
        s = t - 1
        t -= 1
        pts.reverse()

    # Remove duplicate if hull has only 2 points and they're the
same
    end = t - (1 if t == 2 and h[0] == h[1] else 0)
    return h[:end]
```

Delaunay

Computes the Delaunay triangulation of a set of points.

Time: O(n^2)

geometry/delaunay.py

```
"""
Author: Mattias de Zalenski
Date: Unknown
Source: Geometry in C
Description: Computes the Delaunay triangulation of a set of
points.
Each circumcircle contains none of the input points.
If any three points are collinear or any four are on the same
circle,
behavior is undefined.
Time: O(n^2)
Status: stress-tested
"""

from typing import List, Callable
from .point import Point
from .point_3d import Point3D
from .hull_3d import hull_3d

def delaunay_triangulation(ps: List[Point], trifun:
Callable[[int, int, int], None]):
    """
    Compute Delaunay triangulation and call trifun for each
triangle.
    ps = list of 2D points
    trifun = callback function(a, b, c) for each triangle with
vertices a,b,c
    """
    n = len(ps)

    if n == 3:
        # Special case: exactly 3 points
        d = 1 if ps[0].cross(ps[1], ps[2]) < 0 else 0
        trifun(0, 1 + d, 2 - d)
        return

    if n < 3:
        return

    # Lift points to paraboloid: (x, y) -> (x, y, x^2 + y^2)
    p3 = []
    for p in ps:
        p3.append(Point3D(p.x, p.y, p.dist2()))

    # Compute 3D convex hull
    if n > 3:
        faces = hull_3d(p3)

        # Extract lower hull faces (those visible from below, z =
-infinity)
        for face in faces:
            v1 = p3[face.b] - p3[face.a]
            v2 = p3[face.c] - p3[face.a]
            normal = v1.cross(v2)

            # If normal points down (negative z component), it's
part of lower hull
            if normal.z < 0:
                trifun(face.a, face.c, face.b)

def get_delaunay_triangles(ps: List[Point]) -> List[tuple]:
    """
    Compute Delaunay triangulation and return list of triangles.
    ps = list of 2D points
    Returns list of (a, b, c) tuples representing triangle vertex
indices
    """
    triangles = []
# ... (continued)
```

Fast Delaunay

Fast Delaunay triangulation using divide-and-conquer.

Time: O(n log n)

geometry/fast_delaunay.py

```
"""
Author: Philippe Legault
Date: 2016
License: MIT
Source: https://github.com/Bathlamos/delaunay-triangulation/
Description: Fast Delaunay triangulation using divide-and-
conquer.
Each circumcircle contains none of the input points.
There must be no duplicate points.
If all points are on a line, no triangles will be returned.
Returns triangles as flat list: [t0_p0, t0_p1, t0_p2, t1_p0,
...], all counter-clockwise.
Time: O(n log n)
Status: stress-tested

Note: This is a complex algorithm. For simpler cases, use
delaunay.py
which is based on 3D convex hull and is easier to understand.
"""

from typing import List, Optional, Tuple
from .point import Point

class QuadEdge:
    """Quad-edge data structure for Delaunay triangulation"""

    def __init__(self):
        self.rot: Optional[QuadEdge] = None
        self.o: Optional[QuadEdge] = None
        self.p: Optional[Point] = None
        self.mark: bool = False

    def sym(self) -> 'QuadEdge':
        """Symmetric edge"""
        return self.rot.rot

    def onext(self) -> 'QuadEdge':
        """Next edge around origin"""
        return self.o

    def oprev(self) -> 'QuadEdge':
        """Previous edge around origin"""
        return self.rot.o.rot

    def dest(self) -> Point:
        """Destination vertex"""
        return self.sym().p

class FastDelaunay:
    """Fast Delaunay triangulation using quad-edge structure"""

    def __init__(self):
        self.free_list: Optional[QuadEdge] = None

    def in_circle(self, p: Point, a: Point, b: Point, c: Point) -
> bool:
        """Check if p is inside circumcircle of triangle abc"""
        p2 = p.dist2()
        A = a.dist2() - p2
        B = b.dist2() - p2
        C = c.dist2() - p2

        return (p.cross(a, b) * C +
                p.cross(b, c) * A +
# ... (continued)
```

Hull 3D

Computes all faces of the 3-dimension hull of a point set.

Time: O(n^2)

geometry/hull_3d.py

```
"""
Author: Johan Sannemo
Date: 2017-04-18
Source: derived from https://gist.github.com/msg555/4963794 by
Mark Gordon
Description: Computes all faces of the 3-dimension hull of a
point set.
*No four points must be coplanar*, or else random results will be
returned.
All faces will point outwards.
Time: O(n^2)
Status: tested on SPOJ CH3D
"""

from typing import List, Tuple
from .point_3d import Point3D

class Face:
    def __init__(self, q: Point3D, a: int, b: int, c: int):
        self.q = q  # Normal vector
        self.a = a  # Vertex indices
        self.b = b
        self.c = c

def hull_3d(A: List[Point3D]) -> List[Face]:
    """
    Compute 3D convex hull.
    A = list of 3D points (at least 4, no 4 coplanar)
    Returns list of Face objects
    """
    assert len(A) >= 4, "Need at least 4 points"

    n = len(A)
    # E[i][j] = pair of faces adjacent to edge i-j
    E = [[[-1, -1] for _ in range(n)] for _ in range(n)]

    def E_ins(i, j, x):
        """Insert face x into edge i-j"""
        if E[i][j][0] == -1:
            E[i][j][0] = x
        else:
            E[i][j][1] = x

    def E_rem(i, j, x):
        """Remove face x from edge i-j"""
        if E[i][j][0] == x:
            E[i][j][0] = -1
        else:
            E[i][j][1] = -1

    def E_cnt(i, j):
        """Count faces on edge i-j"""
        return (E[i][j][0] != -1) + (E[i][j][1] != -1)

    FS = []  # List of faces

    def make_face(i, j, k, l):
        """Create face from points i, j, k with l as reference"""
        q = (A[j] - A[i]).cross(A[k] - A[i])
        if q.dot(A[l]) > q.dot(A[i]):
            q = q * -1

        f = Face(q, i, j, k)
# ... (continued)
```

Hull Diameter

Returns the two points with max distance on a convex hull (ccw,

Time: O(n)

geometry/hull_diameter.py

```
"""
Author: Oleksandr Bacherikov, chilli
Date: 2019-05-05
License: Boost Software License
Source: https://codeforces.com/blog/entry/48868
Description: Returns the two points with max distance on a convex
hull (ccw,
no duplicate/collinear points).
Status: stress-tested, tested on kattis:roberthood
Time: O(n)
"""

from typing import List, Tuple
from .point import Point

def hull_diameter(S: List[Point]) -> Tuple[Point, Point]:
    """
    Find diameter of convex hull (furthest pair of points).
    Hull must be CCW with no duplicates or collinear points.
    Returns (point1, point2)
    """
    n = len(S)
    if n < 2:
        return (S[0], S[0])

    j = 1
    res_dist2 = 0
    res_points = (S[0], S[0])

    for i in range(j):
        while True:
            dist2 = (S[i] - S[j]).dist2()
            if dist2 > res_dist2:
                res_dist2 = dist2
                res_points = (S[i], S[j])

            if (S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >=
0:
                break
            j = (j + 1) % n

    return res_points
```

Inside Polygon

Returns true if p lies within the polygon. If strict is true,
Time: O(n)

geometry/inside_polygon.py

```
"""
Author: Victor Lecomte, chilli
Date: 2019-04-26
License: CC0
Source: https://vlecomte.github.io/cp-geo.pdf
Description: Returns true if p lies within the polygon. If strict is true,
it returns false for points on the boundary. The algorithm uses
products in intermediate steps so watch out for overflow.
Time: O(n)
Status: stress-tested and tested on kattis:pointinpolygon
"""

from typing import List
from .point import Point
from .on_segment import on_segment

def inside_polygon(p: List[Point], a: Point, strict: bool = True)
-> bool:
    """
    Check if point a is inside polygon p.
    strict=True: excludes boundary points
    strict=False: includes boundary points
    """
    cnt = 0
    n = len(p)

    for i in range(n):
        q = p[(i + 1) % n]
        if on_segment(p[i], q, a):
            return not strict
        cnt ^= ((a.y < p[i].y) - (a.y < q.y)) * a.cross(p[i], q)
> 0

    return bool(cnt)
```

Kd Tree

KD-tree for nearest neighbor search (2d, can be extended to 3d)

geometry/kd_tree.py

```
"""
Author: Stanford
Date: Unknown
Source: Stanford Notebook
Description: KD-tree for nearest neighbor search (2d, can be
extended to 3d)
Status: Tested on excellentengineers
"""

from typing import List, Tuple, Optional
from .point import Point

class KDNode:
    """Node in KD-tree"""

    def __init__(self, points: List[Point]):
        self.pt = points[0]
        self.x0 = min(p.x for p in points)
        self.x1 = max(p.x for p in points)
        self.y0 = min(p.y for p in points)
        self.y1 = max(p.y for p in points)
        self.first = None
        self.second = None

        if len(points) > 1:
            # Split on x if width >= height
            if self.x1 - self.x0 >= self.y1 - self.y0:
                points.sort(key=lambda p: p.x)
            else:
                points.sort(key=lambda p: p.y)

            half = len(points) // 2
            self.first = KDNode(points[:half])
            self.second = KDNode(points[half:])

    def distance(self, p: Point) -> float:
        """Minimum squared distance from point p to this bounding
box"""
        x = p.x if self.x0 <= p.x <= self.x1 else (self.x0 if p.x
< self.x0 else self.x1)
        y = p.y if self.y0 <= p.y <= self.y1 else (self.y0 if p.y
< self.y0 else self.y1)
        return (Point(x, y) - p).dist2()

class KDTree:
    """KD-tree for efficient nearest neighbor queries"""

    def __init__(self, points: List[Point]):
        """Build KD-tree from list of points"""
        self.root = KDNode(points[:])

    def _search(self, node: KDNode, p: Point) -> Tuple[float,
Point]:
        """Recursively search for nearest point"""
        if node.first is None:
            # Leaf node
            return ((p - node.pt).dist2(), node.pt)

        # Search closer child first
        f, s = node.first, node.second
        bfirst = f.distance(p)
        bsec = s.distance(p)

        if bfirst > bsec:
            f, s = s, f
# ... (continued)
```

Line Distance

geometry/line_distance.py

```
"""
Author: Ulf Lundstrom
Date: 2009-03-21
License: CC0
Source: Basic math
Description:
Returns the signed distance between point p and the line
containing points a and b.
Positive value on left side and negative on right as seen from a
towards b. a==b gives nan.
It uses products in intermediate steps so watch out for overflow.
Status: tested
"""

from .point import Point

def line_dist(a: Point, b: Point, p: Point) -> float:
    """
    Signed distance from point p to line through a and b.
    Positive on left, negative on right (as seen from a to b).
    """
    return (b - a).cross(p - a) / (b - a).dist()
```

Line Hull Intersection

Line-convex polygon intersection. The polygon must be ccw and have no collinear points.

Time: O(log n)

geometry/line_hull_intersection.py

```
"""
Author: Oleksandr Bacherikov, chilli
Date: 2019-05-07
License: Boost Software License
Source: https://github.com/AlCash07/ACTL
Description: Line-convex polygon intersection. The polygon must
be ccw and have no collinear points.
line_hull returns a pair describing the intersection:
  (-1, -1) if no collision
  (i, -1) if touching corner i
  (i, i) if along side (i, i+1)
  (i, j) if crossing sides (i, i+1) and (j, j+1)
Time: O(log n)
Status: stress-tested
"""

from typing import List, Tuple
from .point import Point
from .side_of import sgn

def extr_vertex(poly: List[Point], dir: Point) -> int:
    """Find extreme vertex in direction dir"""
    n = len(poly)

    def cmp(i, j):
        return sgn(dir.perp().cross(poly[i % n] - poly[j % n]))

    def extr(i):
        return cmp(i + 1, i) >= 0 and cmp(i, i - 1 + n) < 0

    if extr(0):
        return 0

    lo = 0
    hi = n
    while lo + 1 < hi:
        m = (lo + hi) // 2
        if extr(m):
            return m
        ls = cmp(lo + 1, lo)
        ms = cmp(m + 1, m)
        if ls < ms or (ls == ms and ls == cmp(lo, m)):
            hi = m
        else:
            lo = m

    return lo

def line_hull(a: Point, b: Point, poly: List[Point]) ->
Tuple[int, int]:
    """
    Find intersection of line through a,b with convex polygon.
    Returns (i, j) as described in docstring.
    """
    n = len(poly)

    def cmpL(i):
        return sgn(a.cross(poly[i], b))

    endA = extr_vertex(poly, (a - b).perp())
    endB = extr_vertex(poly, (b - a).perp())

# ... (continued)
```

Line Intersection

geometry/line_intersection.py

```
"""
Author: Victor Lecomte, chilli
Date: 2019-05-05
License: CC0
Source: https://vlecomte.github.io/cp-geo.pdf
Description:
If a unique intersection point of the lines going through s1,e1
and s2,e2 exists {1, point} is returned.
If no intersection point exists {0, (0,0)} is returned and if
infinitely many exists {-1, (0,0)} is returned.
The wrong position will be returned if P is Point with integer
coordinates and the intersection point does not have integer
coordinates.
Products of three coordinates are used in intermediate steps so
watch out for overflow.
Status: stress-tested, and tested through half-plane tests
"""

from typing import Tuple
from .point import Point

def line_intersection(s1: Point, e1: Point, s2: Point, e2: Point)
-> Tuple[int, Point]:
    """
    Find intersection of two lines.
    Returns (status, point) where:
      status = 1: unique intersection at point
      status = 0: no intersection (parallel lines)
      status = -1: infinitely many intersections (same line)
    """
    d = (e1 - s1).cross(e2 - s2)
    if d == 0:  # parallel
        status = -1 if s1.cross(e1, s2) == 0 else 0
        return (status, Point(0, 0))

    p = s2.cross(e1, e2)
    q = s2.cross(e2, s1)
    return (1, (s1 * p + e1 * q) / d)
```

Line Projection

Projects point p onto line ab. Set refl=True to get reflection

geometry/line_projection.py

```
"""
Author: Victor Lecomte, chilli
Date: 2019-10-29
License: CC0
Source: https://vlecomte.github.io/cp-geo.pdf
Description: Projects point p onto line ab. Set refl=True to get
reflection
of point p across line ab instead.
Products of three coordinates are used in intermediate steps so
watch out for overflow.
Status: stress-tested
"""

from .point import Point

def line_proj(a: Point, b: Point, p: Point, refl: bool = False) -
> Point:
    """
    Project point p onto line through a and b.
    If refl=True, returns reflection instead of projection.
    """
    v = b - a
    return p - v.perp() * (1 + refl) * v.cross(p - a) / v.dist2()
```

Linear Transformation

Apply the linear transformation (translation, rotation and scaling)

geometry/linear_transformation.py

```
"""
Author: Per Austrin, Ulf Lundstrom
Date: 2009-04-09
License: CC0
Description: Apply the linear transformation (translation,
rotation and scaling)
which takes line p0-p1 to line q0-q1 to point r.
Status: not tested
"""

from .point import Point

def linear_transformation(p0: Point, p1: Point, q0: Point, q1:
Point, r: Point) -> Point:
    """
    Transform point r by the linear transformation that maps line
p0-p1 to q0-q1.
    p0, p1 = source line endpoints
    q0, q1 = destination line endpoints
    r = point to transform
    Returns transformed point
    """
    dp = p1 - p0
    dq = q1 - q0

    # Compute complex number representing transformation
    num = Point(dp.cross(dq), dp.dot(dq))

    # Apply transformation
    r_rel = r - p0
    transformed = Point(r_rel.cross(num), r_rel.dot(num))

    return q0 + transformed / dp.dist2()

# Example usage
if __name__ == "__main__":
    # Transform a point from one coordinate system to another
    p0 = Point(0.0, 0.0)
    p1 = Point(1.0, 0.0)
    q0 = Point(0.0, 0.0)
    q1 = Point(0.0, 1.0)  # 90 degree rotation
    r = Point(0.5, 0.0)

    result = linear_transformation(p0, p1, q0, q1, r)
    print(f"Transformed point: ({result.x}, {result.y})")
```

Manhattan Mst

Given N points, returns up to 4*N edges, which are guaranteed

Time: O(N log N)

geometry/manhattan_mst.py

```
"""
Author: chilli, Takanori MAEHARA
Date: 2019-11-02
License: CC0
Source: https://github.com/spaghetti-source/algorithm
Description: Given N points, returns up to 4*N edges, which are
guaranteed
to contain a minimum spanning tree for the graph with edge
weights w(p, q) =
|p.x - q.x| + |p.y - q.y|. Edges are in the form (distance, src,
dst). Use a
standard MST algorithm on the result to find the final MST.
Time: O(N log N)
Status: Stress-tested
"""

from typing import List, Tuple
from .point import Point

def manhattan_mst(ps: List[Point]) -> List[Tuple[int, int, int]]:
    """
    Find candidate edges for Manhattan MST.
    ps = list of points
    Returns list of (distance, src_idx, dst_idx) edges
    """
    n = len(ps)
    ps = ps[:]  # Copy
    id_list = list(range(n))
    edges = []

    for k in range(4):
        # Sort by x - y
        id_list.sort(key=lambda i: ps[i].x - ps[i].y)

        sweep = {}  # y -> point index
        for i in id_list:
            # Find points that could form edges
            keys_to_remove = []
            for y_key in sorted(sweep.keys()):
                if y_key < -ps[i].y:
                    continue

                j = sweep[y_key]
                d = ps[i] - ps[j]
                if d.y > d.x:
                    break
                edges.append((d.y + d.x, i, j))
                keys_to_remove.append(y_key)

            for key in keys_to_remove:
                del sweep[key]

            sweep[-ps[i].y] = i

        # Transform points for next iteration
        if k & 1:
            for p in ps:
                p.x = -p.x
        else:
            for p in ps:
                p.x, p.y = p.y, p.x

    return edges
# ... (continued)
```

Minimum Enclosing Circle

Computes the minimum circle that encloses a set of points.

Time: expected O(n)

geometry/minimum_enclosing_circle.py

```
"""
Author: Andrew He, chilli
Date: 2019-05-07
License: CC0
Source: folklore
Description: Computes the minimum circle that encloses a set of
points.
Time: expected O(n)
Status: stress-tested
"""

import random
from typing import List, Tuple
from .point import Point
from .circumcircle import cc_center

def minimum_enclosing_circle(ps: List[Point]) -> Tuple[Point,
float]:
    """
    Find minimum enclosing circle for a set of points.
    Returns (center, radius)
    """
    ps = ps[:]
    random.shuffle(ps)

    o = ps[0]
    r = 0.0
    EPS = 1 + 1e-8

    for i in range(len(ps)):
        if (o - ps[i]).dist() > r * EPS:
            o = ps[i]
            r = 0.0
            for j in range(i):
                if (o - ps[j]).dist() > r * EPS:
                    o = (ps[i] + ps[j]) / 2
                    r = (o - ps[i]).dist()
                    for k in range(j):
                        if (o - ps[k]).dist() > r * EPS:
                            o = cc_center(ps[i], ps[j], ps[k])
                            r = (o - ps[i]).dist()

    return (o, r)
```

On Segment

Returns true iff p lies on the line segment from s to e.

geometry/on_segment.py

```
"""
Author: Victor Lecomte, chilli
Date: 2019-04-26
License: CC0
Source: https://vlecomte.github.io/cp-geo.pdf
Description: Returns true iff p lies on the line segment from s
to e.
Use (seg_dist(s,e,p)<=epsilon) instead when using floating point.
Status: tested
"""

from .point import Point

def on_segment(s: Point, e: Point, p: Point) -> bool:
    """Check if point p lies on line segment from s to e"""
    return p.cross(s, e) == 0 and (s - p).dot(e - p) <= 0
```

Point

Class to handle points in the plane.

geometry/point.py

```
"""
Author: Ulf Lundstrom
Date: 2009-02-26
License: CC0
Source: My head with inspiration from tinyKACTL
Description: Class to handle points in the plane.
Status: Works fine, used a lot
"""

import math
from typing import Tuple

def sgn(x):
    """Return sign of x: 1 if positive, -1 if negative, 0 if
zero"""
    return (x > 0) - (x < 0)

class Point:
    """2D Point class"""
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def __lt__(self, p):
        return (self.x, self.y) < (p.x, p.y)

    def __eq__(self, p):
        return (self.x, self.y) == (p.x, p.y)

    def __add__(self, p):
        return Point(self.x + p.x, self.y + p.y)

    def __sub__(self, p):
        return Point(self.x - p.x, self.y - p.y)

    def __mul__(self, d):
        return Point(self.x * d, self.y * d)

    def __truediv__(self, d):
        return Point(self.x / d, self.y / d)

    def dot(self, p):
        """Dot product"""
        return self.x * p.x + self.y * p.y

    def cross(self, p, b=None):
        """Cross product. If b is given, compute cross product of
vectors (p-self) and (b-self)"""
        if b is None:
            return self.x * p.y - self.y * p.x
        else:
            return (p - self).cross(b - self)

    def dist2(self):
        """Squared distance from origin"""
        return self.x * self.x + self.y * self.y

    def dist(self):
        """Distance from origin"""
        return math.sqrt(self.dist2())

    def angle(self):
# ... (continued)
```

Point 3D

Class to handle points in 3D space.

geometry/point_3d.py

```
"""
Author: Ulf Lundstrom with inspiration from tinyKACTL
Date: 2009-04-14
License: CC0
Description: Class to handle points in 3D space.
Usage:
Status: tested, except for phi and theta
"""

import math
from typing import Tuple

class Point3D:
    def __init__(self, x: float = 0, y: float = 0, z: float = 0):
        self.x = x
        self.y = y
        self.z = z

    def __lt__(self, p: 'Point3D') -> bool:
        return (self.x, self.y, self.z) < (p.x, p.y, p.z)

    def __eq__(self, p: 'Point3D') -> bool:
        return (self.x, self.y, self.z) == (p.x, p.y, p.z)

    def __add__(self, p: 'Point3D') -> 'Point3D':
        return Point3D(self.x + p.x, self.y + p.y, self.z + p.z)

    def __sub__(self, p: 'Point3D') -> 'Point3D':
        return Point3D(self.x - p.x, self.y - p.y, self.z - p.z)

    def __mul__(self, d: float) -> 'Point3D':
        return Point3D(self.x * d, self.y * d, self.z * d)

    def __truediv__(self, d: float) -> 'Point3D':
        return Point3D(self.x / d, self.y / d, self.z / d)

    def dot(self, p: 'Point3D') -> float:
        """Dot product"""
        return self.x * p.x + self.y * p.y + self.z * p.z

    def cross(self, p: 'Point3D') -> 'Point3D':
        """Cross product"""
        return Point3D(
            self.y * p.z - self.z * p.y,
            self.z * p.x - self.x * p.z,
            self.x * p.y - self.y * p.x
        )

    def dist2(self) -> float:
        """Squared distance from origin"""
        return self.x * self.x + self.y * self.y + self.z *
self.z

    def dist(self) -> float:
        """Distance from origin"""
        return math.sqrt(self.dist2())

    def phi(self) -> float:
        """Azimuthal angle (longitude) to x-axis in interval [-
pi, pi]"""
        return math.atan2(self.y, self.x)

# ... (continued)
```

Point Inside Hull

Determine whether a point t lies inside a convex hull (CCW

Time: O(log N)

geometry/point_inside_hull.py

```
"""
Author: chilli
Date: 2019-05-17
License: CC0
Source: https://github.com/ngthanhtrung23/ACM_Notebook_new
Description: Determine whether a point t lies inside a convex
hull (CCW
order, with no collinear points). Returns true if point lies
within
the hull. If strict is true, points on the boundary aren't
included.
Time: O(log N)
Status: stress-tested
"""

from typing import List
from .point import Point
from .side_of import side_of, sgn
from .on_segment import on_segment

def point_inside_hull(l: List[Point], p: Point, strict: bool =
True) -> bool:
    """
    Check if point p is inside convex hull l.
    Hull must be in CCW order with no collinear points.
    strict=True: excludes boundary
    strict=False: includes boundary
    """
    n = len(l)
    r = 0 if strict else 1

    if n < 3:
        return r and on_segment(l[0], l[-1], p)

    a = 1
    b = n - 1

    if side_of(l[0], l[a], l[b]) > 0:
        a, b = b, a

    if side_of(l[0], l[a], p) >= r or side_of(l[0], l[b], p) <= -
r:
        return False

    while abs(a - b) > 1:
        c = (a + b) // 2
        if side_of(l[0], l[c], p) > 0:
            b = c
        else:
            a = c

    return sgn(l[a].cross(l[b], p)) < r
```

Polygon Area

Returns twice the signed area of a polygon.

geometry/polygon_area.py

```
"""
Author: Ulf Lundstrom
Date: 2009-03-21
License: CC0
Source: tinyKACTL
Description: Returns twice the signed area of a polygon.
Clockwise enumeration gives negative area. Watch out for overflow
if using int!
Status: Stress-tested and tested on kattis:polygonarea
"""

from typing import List
from .point import Point

def polygon_area2(v: List[Point]):
    """Compute twice the signed area of polygon"""
    a = v[-1].cross(v[0])
    for i in range(len(v) - 1):
        a += v[i].cross(v[i + 1])
    return a
```

Polygon Center

Returns the center of mass for a polygon.

Time: O(n)

geometry/polygon_center.py

```
"""
Author: Ulf Lundstrom
Date: 2009-04-08
License: CC0
Description: Returns the center of mass for a polygon.
Time: O(n)
Status: Tested
"""

from typing import List
from .point import Point

def polygon_center(v: List[Point]) -> Point:
    """Calculate center of mass of polygon"""
    res = Point(0, 0)
    A = 0.0

    j = len(v) - 1
    for i in range(len(v)):
        res = res + (v[i] + v[j]) * v[j].cross(v[i])
        A += v[j].cross(v[i])
        j = i

    return res / A / 3
```

Polygon Cut

geometry/polygon_cut.py

```
"""
Author: Ulf Lundstrom
Date: 2009-03-21
License: CC0
Description:
Returns a vector with the vertices of a polygon with everything
to the left of the line going from s to e cut away.
Status: tested but not extensively
"""

from typing import List
from .point import Point

def polygon_cut(poly: List[Point], s: Point, e: Point) ->
List[Point]:
    """
    Cut polygon with line from s to e.
    Returns vertices of polygon with everything to the left of
line cut away.
    """
    res = []

    for i in range(len(poly)):
        cur = poly[i]
        prev = poly[i - 1] if i else poly[-1]
        a = s.cross(e, cur)
        b = s.cross(e, prev)

        if (a < 0) != (b < 0):
            res.append(cur + (prev - cur) * (a / (a - b)))
        if a < 0:
            res.append(cur)

    return res
```

Polygon Union

Calculates the area of the union of n polygons (not necessarily

Time: O(N^2), where N is the total number of points

geometry/polygon_union.py

```
"""
Author: black_horse2014, chilli
Date: 2019-10-29
License: Unknown
Source: https://codeforces.com/gym/101673/submission/50481926
Description: Calculates the area of the union of n polygons (not
necessarily
convex). The points within each polygon must be given in CCW
order.
Time: O(N^2), where N is the total number of points
Status: stress-tested, Submitted on ECNA 2017 Problem A
"""

from typing import List
from .point import Point
from .side_of import side_of, sgn

def rat(a: Point, b: Point) -> float:
    """Ratio helper function"""
    return a.x / b.x if sgn(b.x) else a.y / b.y

def polygon_union(poly: List[List[Point]]) -> float:
    """
    Calculate area of union of polygons.
    poly = list of polygons (each polygon is a list of points in
CCW order)
    """
    ret = 0.0

    for i in range(len(poly)):
        for v in range(len(poly[i])):
            A = poly[i][v]
            B = poly[i][(v + 1) % len(poly[i])]
            segs = [(0.0, 0), (1.0, 0)]

            for j in range(len(poly)):
                if i != j:
                    for u in range(len(poly[j])):
                        C = poly[j][u]
                        D = poly[j][(u + 1) % len(poly[j])]
                        sc = side_of(A, B, C)
                        sd = side_of(A, B, D)

                        if sc != sd:
                            sa = C.cross(D, A)
                            sb = C.cross(D, B)
                            if min(sc, sd) < 0:
                                segs.append((sa / (sa - sb),
sgn(sc - sd)))
                        elif not sc and not sd and j < i and
sgn((B - A).dot(D - C)) > 0:
                            segs.append((rat(C - A, B - A), 1))
                            segs.append((rat(D - A, B - A), -1))

            segs.sort()
            for k in range(len(segs)):
                segs[k] = (min(max(segs[k][0], 0.0), 1.0),
segs[k][1])

            total = 0.0
            cnt = segs[0][1]
            for j in range(1, len(segs)):
                if not cnt:
                    total += segs[j][0] - segs[j - 1][0]
                cnt += segs[j][1]

# ... (continued)
```

Polyhedron Volume

Magic formula for the volume of a polyhedron. Faces should point outwards.

geometry/polyhedron_volume.py

```
"""
Author: Mattias de Zalenski
Date: 2002-11-04
Description: Magic formula for the volume of a polyhedron. Faces
should point outwards.
Status: tested
"""

from typing import List, Any
from .point_3d import Point3D

def signed_poly_volume(p: List[Point3D], trilist: List[Any]) ->
float:
    """
    Calculate volume of polyhedron.
    p = list of 3D points
    trilist = list of triangles (each with attributes a, b, c as
indices into p)
    """
    v = 0.0
    for tri in trilist:
        v += p[tri.a].cross(p[tri.b]).dot(p[tri.c])
    return v / 6
```

Segment Distance

geometry/segment_distance.py

```
"""
Author: Ulf Lundstrom
Date: 2009-03-21
License: CC0
Description:
Returns the shortest distance between point p and the line
segment from point s to e.
Status: tested
"""

from .point import Point

def segment_dist(s: Point, e: Point, p: Point) -> float:
    """
    Shortest distance from point p to line segment from s to e.
    """
    if s == e:
        return (p - s).dist()
    d = (e - s).dist2()
    t = min(d, max(0.0, (p - s).dot(e - s)))
    return ((p - s) * d - (e - s) * t).dist() / d
```

Segment Intersection

geometry/segment_intersection.py

```
"""
Author: Victor Lecomte, chilli
Date: 2019-04-27
License: CC0
Source: https://vlecomte.github.io/cp-geo.pdf
Description:
If a unique intersection point between the line segments going
from s1 to e1 and from s2 to e2 exists then it is returned.
If no intersection point exists an empty list is returned.
If infinitely many exist a list with 2 elements is returned,
containing the endpoints of the common line segment.
Status: stress-tested, tested on kattis:intersection
"""

from typing import List
from .point import Point
from .on_segment import on_segment
from .side_of import sgn

def segment_intersection(a: Point, b: Point, c: Point, d: Point)
-> List[Point]:
    """
    Find intersection of line segments ab and cd.
    Returns list of intersection points (0, 1, or 2 points).
    """
    oa = c.cross(d, a)
    ob = c.cross(d, b)
    oc = a.cross(b, c)
    od = a.cross(b, d)

    # Check if intersection is single non-endpoint point
    if sgn(oa) * sgn(ob) < 0 and sgn(oc) * sgn(od) < 0:
        return [(a * ob - b * oa) / (ob - oa)]

    # Check endpoints
    s = set()
    if on_segment(c, d, a):
        s.add((a.x, a.y))
    if on_segment(c, d, b):
        s.add((b.x, b.y))
    if on_segment(a, b, c):
        s.add((c.x, c.y))
    if on_segment(a, b, d):
        s.add((d.x, d.y))

    return [Point(x, y) for x, y in sorted(s)]
```

Side Of

Returns where p is as seen from s towards e. 1/0/-1 <=> left/on line/right.

geometry/side_of.py

```
"""
Author: Ulf Lundstrom
Date: 2009-03-21
License: CC0
Description: Returns where p is as seen from s towards e. 1/0/-1
<=> left/on line/right.
If the optional argument eps is given 0 is returned if p is
within distance eps from the line.
It uses products in intermediate steps so watch out for overflow.
Status: tested
"""

from .point import Point

def sgn(x: float) -> int:
    """Sign function"""
    return (x > 0) - (x < 0)

def side_of(s: Point, e: Point, p: Point, eps: float = None) ->
int:
    """
    Check which side of line (s->e) point p is on.
    Returns 1 (left), 0 (on line), -1 (right)
    If eps is given, returns 0 if p is within distance eps from
line
    """
    if eps is None:
        return sgn(s.cross(e, p))
    else:
        a = (e - s).cross(p - s)
        l = (e - s).dist() * eps
        return (1 if a > l else 0) - (1 if a < -l else 0)
```

Spherical Distance

Returns the shortest distance on the sphere with radius between points

geometry/spherical_distance.py

```
"""
Author: Ulf Lundstrom
Date: 2009-04-07
License: CC0
Source: My geometric reasoning
Description: Returns the shortest distance on the sphere with
radius between points
with azimuthal angles (longitude) f1, f2 from x axis and zenith
angles
(latitude) t1, t2 from z axis (0 = north pole). All angles in
radians.
Status: tested on kattis:airlinehub
"""

import math

def spherical_distance(f1: float, t1: float, f2: float, t2:
float, radius: float) -> float:
    """
    Calculate shortest distance on sphere between two points.
    f1, f2 = azimuthal angles (longitude) in radians
    t1, t2 = zenith angles (latitude) in radians (0 = north pole)
    radius = sphere radius
    """
    dx = math.sin(t2) * math.cos(f2) - math.sin(t1) *
math.cos(f1)
    dy = math.sin(t2) * math.sin(f2) - math.sin(t1) *
math.sin(f1)
    dz = math.cos(t2) - math.cos(t1)
    d = math.sqrt(dx * dx + dy * dy + dz * dz)
    return radius * 2 * math.asin(d / 2)
```

Numerical Methods

Berlekamp Massey

Recovers any n-order linear recurrence relation from the first

Time: O(N^2)

numerical/berlekamp_massey.py

```
"""
Author: Lucian Bicsi
Date: 2017-10-31
License: CC0
Source: Wikipedia
Description: Recovers any n-order linear recurrence relation from
the first
2n terms of the recurrence.
Useful for guessing linear recurrences after brute-forcing the
first terms.
Should work on any field, but numerical stability for floats is
not guaranteed.
Output will have size <= n.
Usage: berlekamp_massey([0, 1, 1, 3, 5, 11], mod=1000000007) #
[1, 2]
Time: O(N^2)
Status: bruteforce-tested mod 5 for n <= 5 and all s
"""

from typing import List

def modpow(base: int, exp: int, mod: int) -> int:
    """Modular exponentiation"""
    result = 1
    base %= mod
    while exp > 0:
        if exp & 1:
            result = (result * base) % mod
        base = (base * base) % mod
        exp >>= 1
    return result

def berlekamp_massey(s: List[int], mod: int = 10**9 + 7) ->
List[int]:
    """
    Find minimal linear recurrence for sequence s.
    Returns coefficients [c1, c2, ...] where s[i] = c1*s[i-1] +
c2*s[i-2] + ...
    """
    n = len(s)
    L = 0
    m = 0
    C = [0] * n
    B = [0] * n
    C[0] = B[0] = 1

    b = 1
    for i in range(n):
        m += 1
        d = s[i] % mod
        for j in range(1, L + 1):
            d = (d + C[j] * s[i - j]) % mod
        if d == 0:
            continue
        T = C[:]
        coef = d * modpow(b, mod - 2, mod) % mod
        for j in range(m, n):
            C[j] = (C[j] - coef * B[j - m]) % mod
        if 2 * L > i:
            continue
        L = i + 1 - L
        B = T
        b = d
        m = 0

    C = C[:L + 1]
# ... (continued)
```

Determinant

Calculates determinant of a matrix. Destroys the matrix.

Time: O(N^3)

numerical/determinant.py

```
"""
Author: Simon Lindholm
Date: 2016-09-06
License: CC0
Source: folklore
Description: Calculates determinant of a matrix. Destroys the
matrix.
Time: O(N^3)
Status: somewhat tested
"""

from typing import List

def determinant(a: List[List[float]]) -> float:
    """
    Calculate determinant of matrix a.
    Warning: modifies input matrix
    """
    n = len(a)
    res = 1.0

    for i in range(n):
        # Find pivot
        b = i
        for j in range(i + 1, n):
            if abs(a[j][i]) > abs(a[b][i]):
                b = j

        if i != b:
            a[i], a[b] = a[b], a[i]
            res *= -1

        res *= a[i][i]
        if res == 0:
            return 0

        # Eliminate column
        for j in range(i + 1, n):
            v = a[j][i] / a[i][i]
            if v != 0:
                for k in range(i + 1, n):
                    a[j][k] -= v * a[i][k]

    return res
```

Fast Subset Transform

Transform to a basis with fast convolutions of the form

Time: O(N log N)

numerical/fast_subset_transform.py

```
"""
Author: Lucian Bicsi
Date: 2015-06-25
License: GNU Free Documentation License 1.2
Source: csacademy
Description: Transform to a basis with fast convolutions of the
form
c[z] = sum_{z = x op y} a[x] * b[y],
where op is one of AND, OR, XOR. The size of a must be a power of
two.
Time: O(N log N)
Status: stress-tested
"""

from typing import List

def fst_and(a: List[int], inv: bool = False) -> List[int]:
    """Fast Subset Transform for AND operation"""
    n = len(a)
    a = a[:]
    step = 1
    while step < n:
        for i in range(0, n, 2 * step):
            for j in range(i, i + step):
                u, v = a[j], a[j + step]
                if inv:
                    a[j], a[j + step] = v - u, u
                else:
                    a[j], a[j + step] = v, u + v
        step *= 2
    return a

def fst_or(a: List[int], inv: bool = False) -> List[int]:
    """Fast Subset Transform for OR operation"""
    n = len(a)
    a = a[:]
    step = 1
    while step < n:
        for i in range(0, n, 2 * step):
            for j in range(i, i + step):
                u, v = a[j], a[j + step]
                if inv:
                    a[j], a[j + step] = v, u - v
                else:
                    a[j], a[j + step] = u + v, u
        step *= 2
    return a

def fst_xor(a: List[int], inv: bool = False) -> List[int]:
    """Fast Subset Transform for XOR operation"""
    n = len(a)
    a = a[:]
    step = 1
    while step < n:
        for i in range(0, n, 2 * step):
            for j in range(i, i + step):
                u, v = a[j], a[j + step]
                a[j], a[j + step] = u + v, u - v
        step *= 2
    if inv:
        for i in range(n):
            a[i] //= n
# ... (continued)
```

Fft

fft(a) computes FFT. Useful for convolution.

Time: O(N log N) with N = |A|+|B|

numerical/fft.py

```
"""
Author: Ludo Pulles, chilli, Simon Lindholm
Date: 2019-01-09
License: CC0
Source: http://neerc.ifmo.ru/trains/toulouse/2017/fft2.pdf
Description: fft(a) computes FFT. Useful for convolution.
conv(a, b) = c, where c[x] = sum a[i]*b[x-i].
Rounding is safe if (sum a_i^2 + sum b_i^2)*log2(N) < 9*10^14.
Time: O(N log N) with N = |A|+|B|
Status: somewhat tested
"""

import cmath
from typing import List

def fft(a: List[complex], inv: bool = False) -> List[complex]:
    """FFT in-place"""
    n = len(a)
    if n <= 1:
        return a

    L = n.bit_length() - 1

    # Bit-reverse permutation
    rev = [0] * n
    for i in range(n):
        rev[i] = (rev[i // 2] | (i & 1) << L) // 2
    for i in range(n):
        if i < rev[i]:
            a[i], a[rev[i]] = a[rev[i]], a[i]

    # FFT computation
    k = 1
    while k < n:
        angle = (-2 if inv else 2) * cmath.pi / (2 * k)
        w_len = complex(cmath.cos(angle), cmath.sin(angle))
        for i in range(0, n, 2 * k):
            w = complex(1, 0)
            for j in range(k):
                u = a[i + j]
                v = a[i + j + k] * w
                a[i + j] = u + v
                a[i + j + k] = u - v
                w *= w_len
        k *= 2

    if inv:
        for i in range(n):
            a[i] /= n

    return a

def conv(a: List[float], b: List[float]) -> List[float]:
    """Convolution of two arrays"""
    if not a or not b:
        return []

    res_len = len(a) + len(b) - 1
    n = 1 << (res_len - 1).bit_length()

# ... (continued)
```

Golden Section Search

Finds the argument minimizing the function f in the interval [a,b]

Time: O(log((b-a) / epsilon))

numerical/golden_section_search.py

```
"""
Author: Ulf Lundstrom
Date: 2009-04-17
License: CC0
Source: Numeriska algoritmer med matlab, Gerd Eriksson, NADA, KTH
Description: Finds the argument minimizing the function f in the
interval [a,b]
assuming f is unimodal on the interval, i.e. has only one local
minimum and no
local maximum. The maximum error in the result is eps. Works
equally well for
maximization with a small change in the code.
Time: O(log((b-a) / epsilon))
Status: tested
"""

import math
from typing import Callable

def golden_section_search(a: float, b: float, f:
Callable[[float], float],
                          eps: float = 1e-7, maximize: bool =
False) -> float:
    """
    Find minimum (or maximum) of unimodal function f on [a, b].
    a, b = interval boundaries
    f = function to minimize/maximize
    eps = precision
    maximize = if True, find maximum instead of minimum
    Returns x value that minimizes/maximizes f
    """
    # Golden ratio constant
    r = (math.sqrt(5) - 1) / 2

    x1 = b - r * (b - a)
    x2 = a + r * (b - a)
    f1 = f(x1)
    f2 = f(x2)

    while b - a > eps:
        if (f1 < f2) if not maximize else (f1 > f2):
            b = x2
            x2 = x1
            f2 = f1
            x1 = b - r * (b - a)
            f1 = f(x1)
        else:
            a = x1
            x1 = x2
            f1 = f2
            x2 = a + r * (b - a)
            f2 = f(x2)

    return a

# Example usage
if __name__ == "__main__":
    # Find minimum of f(x) = 4 + x + 0.3*x^2
    def func(x):
        return 4 + x + 0.3 * x * x

    xmin = golden_section_search(-1000, 1000, func)
    print(f"Minimum at x = {xmin:.6f}, f(x) = {func(xmin):.6f}")

    # Find maximum of f(x) = -(x-2)^2
# ... (continued)
```

Hill Climbing

Poor man's optimization for unimodal functions.

numerical/hill_climbing.py

```
"""
Author: Simon Lindholm
Date: 2015-02-04
License: CC0
Source: Johan Sannemo
Description: Poor man's optimization for unimodal functions.
Status: used with great success
"""

from typing import Tuple, Callable, List

def hill_climb(start: List[float], f: Callable[[List[float]],
float]) -> Tuple[float, List[float]]:
    """
    Hill climbing optimization for unimodal functions.
    start = starting point [x, y]
    f = function to minimize
    Returns (min_value, [x, y])
    """
    cur_val = f(start)
    cur_point = start[:]

    jmp = 1e9
    while jmp > 1e-20:
        for _ in range(100):
            for dx in [-1, 0, 1]:
                for dy in [-1, 0, 1]:
                    p = [cur_point[0] + dx * jmp, cur_point[1] +
dy * jmp]
                    val = f(p)
                    if val < cur_val:
                        cur_val = val
                        cur_point = p
        jmp /= 2

    return (cur_val, cur_point)
```

Int Determinant

Calculates determinant using modular arithmetics.

Time: O(N^3)

numerical/int_determinant.py

```
"""
Author: Unknown
Date: 2014-11-27
Source: somewhere on github
Description: Calculates determinant using modular arithmetics.
Modulos can also be removed to get a pure-integer version.
Time: O(N^3)
Status: bruteforce-tested for N <= 3, mod <= 7
"""

from typing import List

def int_determinant(a: List[List[int]], mod: int = 10**9 + 7) ->
int:
    """
    Calculate determinant of integer matrix modulo mod.
    Warning: modifies input matrix
    """
    n = len(a)
    ans = 1

    for i in range(n):
        for j in range(i + 1, n):
            # GCD step
            while a[j][i] != 0:
                t = a[i][i] // a[j][i]
                if t:
                    for k in range(i, n):
                        a[i][k] = (a[i][k] - a[j][k] * t) % mod
                a[i], a[j] = a[j], a[i]
                ans *= -1
        ans = ans * a[i][i] % mod
        if not ans:
            return 0

    return (ans + mod) % mod
```

Integrate

Simple integration of a function over an interval using

numerical/integrate.py

```
"""
Author: Simon Lindholm
Date: 2015-02-11
License: CC0
Source: Wikipedia
Description: Simple integration of a function over an interval
using
Simpson's rule. The error should be proportional to h^4, although
in
practice you will want to verify that the result is stable to
desired
precision when epsilon changes.
Status: mostly untested
"""

from typing import Callable

def integrate(a: float, b: float, f: Callable[[float], float], n:
int = 1000) -> float:
    """
    Integrate function f from a to b using Simpson's rule.
    n = number of intervals (higher = more accurate)
    """
    h = (b - a) / 2 / n
    v = f(a) + f(b)
    for i in range(1, n * 2):
        v += f(a + i * h) * (4 if i & 1 else 2)
    return v * h / 3
```

Integrate Adaptive

Fast integration using an adaptive Simpson's rule.

numerical/integrate_adaptive.py

```
"""
Author: Simon Lindholm
Date: 2015-02-11
License: CC0
Source: Wikipedia
Description: Fast integration using an adaptive Simpson's rule.
Status: mostly untested
"""

from typing import Callable

def integrate_adaptive(a: float, b: float, f: Callable[[float], float],
                       eps: float = 1e-8) -> float:
    """
    Adaptive Simpson's rule integration.
    a, b = integration bounds
    f = function to integrate
    eps = desired precision
    """

    def simpson(x, y):
        """Simpson's rule estimate"""
        mid = (x + y) / 2
        return (f(x) + 4 * f(mid) + f(y)) * (y - x) / 6

    def rec(x, y, eps_local, S):
        """Recursive adaptive integration"""
        mid = (x + y) / 2
        S1 = simpson(x, mid)
        S2 = simpson(mid, y)
        T = S1 + S2

        if abs(T - S) <= 15 * eps_local or y - x < 1e-10:
            return T + (T - S) / 15

        return (rec(x, mid, eps_local / 2, S1) +
                rec(mid, y, eps_local / 2, S2))

    return rec(a, b, eps, simpson(a, b))
```

Linear Recurrence

Generates the k'th term of an n-order

Time: O(n^2 log k)

numerical/linear_recurrence.py

```
"""
Author: Lucian Bicsi
Date: 2018-02-14
License: CC0
Source: Chinese material
Description: Generates the k'th term of an n-order
linear recurrence S[i] = sum_j S[i-j-1]*tr[j],
given S[0 ... >= n-1] and tr[0 ... n-1].
Faster than matrix multiplication.
Useful together with Berlekamp-Massey.
Usage: linear_recurrence([0, 1], [1, 1], k, mod) # k'th Fibonacci number
Time: O(n^2 log k)
Status: bruteforce-tested mod 5 for n <= 5
"""

from typing import List

def linear_recurrence(S: List[int], tr: List[int], k: int, mod:
int = 10**9 + 7) -> int:
    """
    Compute k'th term of linear recurrence.
    S = initial terms
    tr = transition coefficients
    k = index to compute
    mod = modulus
    """
    n = len(tr)

    def combine(a: List[int], b: List[int]) -> List[int]:
        """Combine two polynomials"""
        res = [0] * (n * 2 + 1)
        for i in range(n + 1):
            for j in range(n + 1):
                res[i + j] = (res[i + j] + a[i] * b[j]) % mod
        for i in range(2 * n, n, -1):
            for j in range(n):
                res[i - 1 - j] = (res[i - 1 - j] + res[i] *
tr[j]) % mod
        return res[:n + 1]

    pol = [0] * (n + 1)
    e = [0] * (n + 1)
    pol[0] = 1
    e[1] = 1

    k += 1
    while k:
        if k % 2:
            pol = combine(pol, e)
        e = combine(e, e)
        k //= 2

    res = 0
    for i in range(n):
        res = (res + pol[i + 1] * S[i]) % mod
    return res
```

Matrix Inverse

Invert matrix A. Returns rank; result is stored in A unless singular (rank < n).

Time: O(n^3)

numerical/matrix_inverse.py

```
"""
Author: Max Bennedich
Date: 2004-02-08
Description: Invert matrix A. Returns rank; result is stored in A
unless singular (rank < n).
Can easily be extended to prime moduli; for prime powers,
repeatedly
set A^{-1} = A^{-1} (2I - AA^{-1}) (mod p^k) where A^{-1} starts
as
the inverse of A mod p, and k is doubled in each step.
Time: O(n^3)
Status: Slightly tested
"""

from typing import List

def matrix_inverse(A: List[List[float]]) -> int:
    """
    Invert matrix A in place using Gaussian elimination.
    A = square matrix (will be modified to contain inverse)
    Returns rank of matrix (n if invertible, < n if singular)
    """
    n = len(A)
    col = list(range(n))

    # Initialize identity matrix
    tmp = [[0.0] * n for _ in range(n)]
    for i in range(n):
        tmp[i][i] = 1.0

    for i in range(n):
        # Find pivot
        r, c = i, i
        for j in range(i, n):
            for k in range(i, n):
                if abs(A[j][k]) > abs(A[r][c]):
                    r, c = j, k

        # Check if matrix is singular
        if abs(A[r][c]) < 1e-12:
            return i

        # Swap rows
        A[i], A[r] = A[r], A[i]
        tmp[i], tmp[r] = tmp[r], tmp[i]

        # Swap columns
        for j in range(n):
            A[j][i], A[j][c] = A[j][c], A[j][i]
            tmp[j][i], tmp[j][c] = tmp[j][c], tmp[j][i]
        col[i], col[c] = col[c], col[i]

        # Eliminate
        v = A[i][i]
        for j in range(i + 1, n):
            f = A[j][i] / v
            A[j][i] = 0
            for k in range(i + 1, n):
                A[j][k] -= f * A[i][k]
            for k in range(n):
                tmp[j][k] -= f * tmp[i][k]

        # Scale row
# ... (continued)
```

Matrix Inverse Mod

Invert matrix A modulo a prime.

Time: O(n^3)

numerical/matrix_inverse_mod.py

```
"""
Author: Simon Lindholm
Date: 2016-12-08
Source: The regular matrix inverse code
Description: Invert matrix A modulo a prime.
Returns rank; result is stored in A unless singular (rank < n).
Time: O(n^3)
Status: Slightly tested
"""

from typing import List

def modpow(base: int, exp: int, mod: int) -> int:
    """Modular exponentiation"""
    result = 1
    base %= mod
    while exp > 0:
        if exp & 1:
            result = (result * base) % mod
        base = (base * base) % mod
        exp >>= 1
    return result

def matrix_inverse_mod(A: List[List[int]], mod: int) -> int:
    """
    Compute matrix inverse modulo a prime.
    A = matrix to invert (modified in-place to contain inverse)
    mod = prime modulus
    Returns rank (n if successful, < n if singular)
    """
    n = len(A)
    col = list(range(n))
    tmp = [[0] * n for _ in range(n)]

    # Initialize tmp as identity
    for i in range(n):
        tmp[i][i] = 1

    # Forward elimination
    for i in range(n):
        # Find pivot
        r = i
        c = i
        found = False
        for j in range(i, n):
            for k in range(i, n):
                if A[j][k] != 0:
                    r = j
                    c = k
                    found = True
                    break
            if found:
                break

        if not found:
            return i  # Singular matrix

        # Swap rows
        A[i], A[r] = A[r], A[i]
        tmp[i], tmp[r] = tmp[r], tmp[i]
# ... (continued)
```

Ntt

ntt(a) computes Number Theoretic Transform for mod = 998244353.

Time: O(N log N)

numerical/ntt.py

```
"""
Author: chilli
Date: 2019-04-16
License: CC0
Source: based on KACTL's FFT
Description: ntt(a) computes Number Theoretic Transform for mod =
998244353.
Useful for convolution modulo specific nice primes.
conv(a, b) = c, where c[x] = sum a[i]*b[x-i] (mod 998244353).
Inputs must be in [0, mod).
Time: O(N log N)
Status: stress-tested
"""

from typing import List

MOD = 998244353
ROOT = 62

def modpow(base: int, exp: int, mod: int) -> int:
    """Modular exponentiation"""
    result = 1
    base %= mod
    while exp > 0:
        if exp & 1:
            result = (result * base) % mod
        base = (base * base) % mod
        exp >>= 1
    return result

def ntt(a: List[int], inv: bool = False) -> List[int]:
    """Number Theoretic Transform in-place"""
    n = len(a)
    if n <= 1:
        return a

    L = n.bit_length() - 1

    # Precompute roots
    rt = [0] * n
    rt[0] = 1
    k = 1
    s = 1
    while k < n:
        if k == 1:
            z = modpow(ROOT, MOD >> (s + 1), MOD)
        else:
            z = rt[k // 2] if k & 1 == 0 else (rt[k // 2] *
modpow(ROOT, MOD >> (s + 1), MOD)) % MOD
        rt[k] = z if k % 2 == 0 else (rt[k // 2] * modpow(ROOT,
MOD >> (s + 1), MOD)) % MOD
        k += 1
        if k & (k - 1) == 0:
            s += 1

    # Bit-reverse permutation
    rev = [0] * n
    for i in range(n):
        rev[i] = (rev[i // 2] | (i & 1) << L) // 2
    for i in range(n):
        if i < rev[i]:
            a[i], a[rev[i]] = a[rev[i]], a[i]

# ... (continued)
```

Poly Interpolate

Given n points (x[i], y[i]), computes an n-1-degree polynomial p that

Time: O(n^2)

numerical/poly_interpolate.py

```
"""
Author: Simon Lindholm
Date: 2017-05-10
License: CC0
Source: Wikipedia
Description: Given n points (x[i], y[i]), computes an n-1-degree
polynomial p that
passes through them: p(x) = a[0]*x^0 + ... + a[n-1]*x^{n-1}.
For numerical precision, pick x[k] = c*cos(k/(n-1)*pi), k=0 ...
n-1.
Time: O(n^2)
"""

from typing import List

def poly_interpolate(x: List[float], y: List[float], n: int) ->
List[float]:
    """
    Interpolate polynomial through n points.
    x, y = lists of coordinates
    Returns coefficients [a0, a1, ..., a_{n-1}]
    """
    res = [0.0] * n
    temp = [0.0] * n
    y = y[:]  # Copy to avoid modifying input

    for k in range(n - 1):
        for i in range(k + 1, n):
            y[i] = (y[i] - y[k]) / (x[i] - x[k])

    last = 0.0
    temp[0] = 1.0

    for k in range(n):
        for i in range(n):
            res[i] += y[k] * temp[i]
            last, temp[i] = temp[i], temp[i] - last * x[k]

    return res
```

Poly Roots

Finds the real roots to a polynomial using recursive bisection.

Time: O(n^2 log(1/epsilon))

numerical/poly_roots.py

```
"""
Author: Per Austrin
Date: 2004-02-08
License: CC0
Description: Finds the real roots to a polynomial using recursive
bisection.
Time: O(n^2 log(1/epsilon))
"""

from typing import List
from .polynomial import Polynomial

def poly_roots(p: Polynomial, xmin: float, xmax: float, eps:
float = 1e-8) -> List[float]:
    """
    Find all real roots of polynomial p in interval [xmin, xmax].
    p = polynomial
    xmin, xmax = search interval
    eps = precision
    Returns list of roots
    """
    # Base case: linear polynomial
    if len(p.a) == 2:
        if abs(p.a[1]) > 1e-10:
            root = -p.a[0] / p.a[1]
            if xmin <= root <= xmax:
                return [root]
        return []

    # Find roots of derivative
    der = Polynomial(p.a[:])
    der.diff()
    dr = poly_roots(der, xmin, xmax, eps)

    # Add boundary points
    dr.append(xmin - 1)
    dr.append(xmax + 1)
    dr.sort()

    ret = []

    # Check each interval between critical points
    for i in range(len(dr) - 1):
        l = dr[i]
        h = dr[i + 1]

        # Check if both endpoints are in valid range
        if h < xmin or l > xmax:
            continue

        l = max(l, xmin)
        h = min(h, xmax)

        # Check if there's a sign change (root exists)
        fl = p(l)
        fh = p(h)

        if (fl > 0) != (fh > 0):
            # Binary search for root
            for _ in range(60):  # Enough iterations for machine
precision
                m = (l + h) / 2
                fm = p(m)
# ... (continued)
```

Polynomial

Basic polynomial operations.

numerical/polynomial.py

```
"""
Author: David Rydh, Per Austrin
Date: 2003-03-16
Description: Basic polynomial operations.
"""

from typing import List

class Polynomial:
    def __init__(self, a: List[float]):
        self.a = a[:]

    def __call__(self, x: float) -> float:
        """Evaluate polynomial at x using Horner's method"""
        val = 0.0
        for i in range(len(self.a) - 1, -1, -1):
            val = val * x + self.a[i]
        return val

    def diff(self):
        """Differentiate polynomial in-place"""
        for i in range(1, len(self.a)):
            self.a[i - 1] = i * self.a[i]
        if self.a:
            self.a.pop()

    def divroot(self, x0: float):
        """Divide polynomial by (x - x0) using synthetic
division"""
        if not self.a:
            return
        b = self.a[-1]
        self.a[-1] = 0
        for i in range(len(self.a) - 2, -1, -1):
            c = self.a[i]
            self.a[i] = self.a[i + 1] * x0 + b
            b = c
        self.a.pop()
```

Simplex

Solves a general linear maximization problem: maximize c^T x subject to Ax <= b, x >= 0.

Time: O(NM * #pivots), where a pivot may be e.g. an edge relaxation. O(2^n) in the general case.

numerical/simplex.py

```
"""
Author: Stanford
Source: Stanford Notebook
License: MIT
Description: Solves a general linear maximization problem:
maximize c^T x subject to Ax <= b, x >= 0.
Returns -inf if there is no solution, inf if there are
arbitrarily good solutions, or the maximum value of c^T x
otherwise.
The input vector is set to an optimal x (or in the unbounded
case, an arbitrary solution fulfilling the constraints).
Numerical stability is not guaranteed. For better performance,
define variables such that x = 0 is viable.
Time: O(NM * #pivots), where a pivot may be e.g. an edge
relaxation. O(2^n) in the general case.
Status: seems to work?
"""

from typing import List

EPS = 1e-8

class LPSolver:
    def __init__(self, A: List[List[float]], b: List[float], c:
List[float]):
        """
        Initialize LP solver.
        A = constraint matrix
        b = constraint RHS
        c = objective function coefficients
        """
        self.m = len(b)
        self.n = len(c)
        self.N = list(range(self.n + 1))
        self.B = [self.n + i for i in range(self.m)]
        self.D = [[0.0] * (self.n + 2) for _ in range(self.m +
2)]

        for i in range(self.m):
            for j in range(self.n):
                self.D[i][j] = A[i][j]
            self.D[i][self.n] = -1
            self.D[i][self.n + 1] = b[i]

        for j in range(self.n):
            self.D[self.m][j] = -c[j]

        self.N[self.n] = -1
        self.D[self.m + 1][self.n] = 1

    def pivot(self, r: int, s: int):
        """Perform pivot operation"""
        inv = 1.0 / self.D[r][s]

        for i in range(self.m + 2):
            if i != r and abs(self.D[i][s]) > EPS:
                inv2 = self.D[i][s] * inv
                for j in range(self.n + 2):
                    self.D[i][j] -= self.D[r][j] * inv2
                self.D[i][s] = self.D[r][s] * inv2

        for j in range(self.n + 2):
            if j != s:
                self.D[r][j] *= inv

        for i in range(self.m + 2):
            if i != r:
                self.D[i][s] *= -inv
# ... (continued)
```

Solve Linear

Solves A * x = b. If there are multiple solutions, an arbitrary one is returned.

Time: O(n^2 m)

numerical/solve_linear.py

```
"""
Author: Per Austrin, Simon Lindholm
Date: 2004-02-08
License: CC0
Description: Solves A * x = b. If there are multiple solutions,
an arbitrary one is returned.
Returns rank, or -1 if no solutions. Data in A and b is lost.
Time: O(n^2 m)
Status: tested on kattis:equationsolver, and bruteforce-tested
mod 3 and 5 for n,m <= 3
"""

from typing import List

EPS = 1e-12

def solve_linear(A: List[List[float]], b: List[float], x:
List[float]) -> int:
    """
    Solve linear system Ax = b.
    A = coefficient matrix (modified)
    b = right hand side (modified)
    x = solution vector (output, should be initialized to correct
size)
    Returns rank, or -1 if no solution
    """
    n = len(A)
    m = len(x)
    if n:
        assert len(A[0]) == m
    col = list(range(m))
    rank = 0

    for i in range(n):
        # Find pivot
        bv = 0.0
        br = bc = i
        for r in range(i, n):
            for c in range(i, m):
                v = abs(A[r][c])
                if v > bv:
                    br, bc, bv = r, c, v

        if bv <= EPS:
            # Check if any remaining equations are inconsistent
            for j in range(i, n):
                if abs(b[j]) > EPS:
                    return -1
            break

        # Swap rows and columns
        A[i], A[br] = A[br], A[i]
        b[i], b[br] = b[br], b[i]
        col[i], col[bc] = col[bc], col[i]
        for j in range(n):
            A[j][i], A[j][bc] = A[j][bc], A[j][i]

        # Eliminate
        bv = 1.0 / A[i][i]
        for j in range(i + 1, n):
            fac = A[j][i] * bv
            b[j] -= fac * b[i]
            for k in range(i + 1, m):
                A[j][k] -= fac * A[i][k]
# ... (continued)
```

Solve Linear Binary

Solves Ax = b over F_2 (binary field).

Time: O(n^2 m)

numerical/solve_linear_binary.py

```
"""
Author: Simon Lindholm
Date: 2016-08-27
License: CC0
Source: own work
Description: Solves Ax = b over F_2 (binary field).
If there are multiple solutions, one is returned arbitrarily.
Returns rank, or -1 if no solutions.
Time: O(n^2 m)
Status: bruteforce-tested for n, m <= 4
"""

from typing import List

def solve_linear_binary(A: List[List[int]], b: List[int], m: int)
-> tuple:
    """
    Solve linear system over F_2 (binary field).
    A = coefficient matrix (modified)
    b = RHS vector (modified)
    m = number of variables
    Returns (rank, solution) or (-1, None) if no solution
    solution is a list of length m with 0/1 values
    """
    n = len(A)
    rank = 0
    col = list(range(m))

    for i in range(n):
        # Find pivot
        br = -1
        for row in range(i, n):
            if any(A[row]):
                br = row
                break

        if br == -1:
            # Check if remaining equations are consistent
            for j in range(i, n):
                if b[j]:
                    return (-1, None)
            break

        # Find first nonzero column
        bc = -1
        for c in range(i, m):
            if A[br][c]:
                bc = c
                break

        # Swap rows and columns
        A[i], A[br] = A[br], A[i]
        b[i], b[br] = b[br], b[i]
        col[i], col[bc] = col[bc], col[i]

        for j in range(n):
            if A[j][i] != A[j][bc]:
                A[j][i] ^= 1
                A[j][bc] ^= 1

        # Eliminate
# ... (continued)
```

Tridiagonal

x = tridiagonal(d, p, q, b) solves a tridiagonal system.

Time: O(N)

numerical/tridiagonal.py

```
"""
Author: Ulf Lundstrom, Simon Lindholm
Date: 2009-08-15
License: CC0
Source:
https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm
Description: x = tridiagonal(d, p, q, b) solves a tridiagonal
system.
Useful for solving problems of the form a_i = b_i * a_{i-1} + c_i
* a_{i+1} + d_i.
Fails if the solution is not unique.
If |d_i| > |p_i| + |q_{i-1}| for all i, or |d_i| > |p_{i-1}| +
|q_i|,
or the matrix is positive definite, the algorithm is numerically
stable.
Time: O(N)
Status: Brute-force tested mod 5 and 7 and stress-tested for real
matrices
"""

from typing import List

def tridiagonal(diag: List[float], super_diag: List[float],
                sub_diag: List[float], b: List[float]) ->
List[float]:
    """
    Solve tridiagonal system.
    diag = main diagonal
    super_diag = super diagonal (above main)
    sub_diag = sub diagonal (below main)
    b = right hand side
    Returns solution vector
    """
    n = len(b)
    tr = [0] * n
    diag = diag[:]  # Copy to avoid modifying input
    b = b[:]

    for i in range(n - 1):
        if abs(diag[i]) < 1e-9 * abs(super_diag[i]):  # diag[i]
== 0
            b[i + 1] -= b[i] * diag[i + 1] / super_diag[i]
            if i + 2 < n:
                b[i + 2] -= b[i] * sub_diag[i + 1] /
super_diag[i]
            diag[i + 1] = sub_diag[i]
            i += 1
            tr[i] = 1
        else:
            diag[i + 1] -= super_diag[i] * sub_diag[i] / diag[i]
            b[i + 1] -= b[i] * sub_diag[i] / diag[i]

    for i in range(n - 1, -1, -1):
        if tr[i]:
            b[i], b[i - 1] = b[i - 1], b[i]
            diag[i - 1] = diag[i]
            b[i] /= super_diag[i - 1]
        else:
            b[i] /= diag[i]
            if i:
                b[i - 1] -= b[i] * super_diag[i - 1]

    return b
```

Various Algorithms

Bump Allocator Note

various/bump_allocator_note.py

```
"""
Bump Allocator in Python
========================

The C++ BumpAllocator overrides the new operator to allocate from a
pre-allocated buffer, avoiding per-allocation overhead.

C++ BumpAllocator Pattern:
--------------------------
```cpp
static char buf[450 << 20];
void* operator new(size_t s) {
    static size_t i = sizeof buf;
    assert(s < i);
    return (void*)&buf[i -= s];
}
void operator delete(void*) {}
```

Why This Doesn't Apply to Python:
----
1. Python manages memory automatically with reference counting and GC
2. No manual new/delete operators
3. Object allocation overhead is already optimized by the interpreter
4. Python's memory allocator is sophisticated (uses arenas, pools, etc.)

Python Equivalents:
----

1. Object Pooling (Closest Equivalent)

```python
class BumpAllocator:
    '''Pre-allocate objects and reuse them'''

    def __init__(self, factory, size=10000):
        self.pool = [factory() for _ in range(size)]
        self.index = 0

    def alloc(self):
        if self.index >= len(self.pool):
            raise MemoryError("Pool exhausted")
        obj = self.pool[self.index]
        self.index += 1
        return obj

    def reset(self):
        '''Reset allocator to reuse all objects'''
        self.index = 0

# Usage
class Node:
    def __init__(self):
        self.value = 0
        self.left = None
        self.right = None

allocator = BumpAllocator(Node, 1000000)

# Allocate many nodes quickly
# ... (continued)
```

Constant Intervals

Split a monotone function on [from, to) into a minimal set of half-open intervals on which it has the same value.
Time: O(k * log(n/k))

various/constant_intervals.py

```
"""
Author: Simon Lindholm
Date: 2015-03-20
License: CC0
Source: me
Description: Split a monotone function on [from, to) into a
minimal set of half-open intervals on which it has the same
value.
Runs a callback g for each such interval.
Time: O(k * log(n/k))
Status: tested
"""

from typing import Callable, Any

def constant_intervals(from_val: int, to_val: int,
                       f: Callable[[int], Any],
                       g: Callable[[int, int, Any], None]):
    """
    Split monotone function into constant intervals.
    from_val, to_val = range [from_val, to_val)
    f = monotone function
    g = callback(lo, hi, val) for each constant interval [lo, hi)
    """
    if to_val <= from_val:
        return

    i = from_val
    p = f(i)
    q = f(to_val - 1)

    def rec(from_idx, to_idx, i_ref, p_ref, q_val):
        """Recursive helper"""
        if p_ref == q_val:
            return i_ref, p_ref

        if from_idx == to_idx:
            g(i_ref, to_idx, p_ref)
            return to_idx, q_val
        else:
            mid = (from_idx + to_idx) >> 1
            i_ref, p_ref = rec(from_idx, mid, i_ref, p_ref,
f(mid))
            i_ref, p_ref = rec(mid + 1, to_idx, i_ref, p_ref,
q_val)
            return i_ref, p_ref

    i, p = rec(from_val, to_val - 1, i, p, q)
    g(i, to_val, q)
```

Divide And Conquer Dp

Given a[i] = min_{lo(i) <= k < hi(i)}(f(i, k)) where the (minimal)
Time: O((N + (hi-lo)) log N)

various/divide_and_conquer_dp.py

```
"""
Author: Simon Lindholm
License: CC0
Source: Codeforces
Description: Given a[i] = min_{lo(i) <= k < hi(i)}(f(i, k)) where
the (minimal)
optimal k increases with i, computes a[i] for i = L..R-1.
Time: O((N + (hi-lo)) log N)
Status: tested on http://codeforces.com/contest/321/problem/E
"""

from typing import Callable, Any, List

class DivideAndConquerDP:
    """
    Divide and Conquer DP optimization.
    Use when optimal k increases monotonically with i.
    """

    def __init__(self,
                 lo_func: Callable[[int], int],
                 hi_func: Callable[[int], int],
                 cost_func: Callable[[int, int], int]):
        """
        lo_func(i) = minimum k to consider for state i
        hi_func(i) = maximum k to consider for state i
        cost_func(i, k) = cost of transition from k to i
        """
        self.lo = lo_func
        self.hi = hi_func
        self.f = cost_func
        self.result = {}

    def rec(self, L: int, R: int, LO: int, HI: int):
        """Recursively solve subproblem"""
        if L >= R:
            return

        mid = (L + R) >> 1
        best_val = float('inf')
        best_k = LO

        for k in range(max(LO, self.lo(mid)), min(HI,
self.hi(mid))):
            val = self.f(mid, k)
            if val < best_val:
                best_val = val
                best_k = k

        self.result[mid] = (best_k, best_val)

        self.rec(L, mid, LO, best_k + 1)
        self.rec(mid + 1, R, best_k, HI)

    def solve(self, L: int, R: int) -> dict:
        """
        Solve for range [L, R).
        Returns dict mapping i to (optimal_k, optimal_value)
        """
        self.result = {}
        self.rec(L, R, -(10**9), 10**9)
        return self.result
# ... (continued)
```
```

Fast Input Note

various/fast_input_note.py

```
"""
Fast Input in Python
====================

The C++ FastInput is a low-level optimization that doesn't
translate directly to Python.
However, Python has several ways to optimize input reading:

Method 1: sys.stdin (Fastest for competitive programming)
---------------------------------------------------------
```python
import sys
input = sys.stdin.readline

# Read single integer
n = int(input())

# Read list of integers
arr = list(map(int, input().split()))

# Read multiple values
a, b, c = map(int, input().split())
```

Method 2: sys.stdin.buffer (Even faster for bulk reading)

```python
import sys
data = sys.stdin.buffer.read().decode().split()
ptr = 0

def read_int():
    global ptr
    ptr += 1
    return int(data[ptr - 1])

# Usage
n = read_int()
arr = [read_int() for _ in range(n)]
```

Method 3: Pre-read all input (Fastest for known input size)

```python
import sys
lines = sys.stdin.read().split('\\n')
idx = 0

def next_line():
    global idx
    result = lines[idx]
    idx += 1
    return result

# Usage
n = int(next_line())
for _ in range(n):
    a, b = map(int, next_line().split())
```

Performance Comparison:
... (continued)
```

Fast Knapsack

Given N non-negative integer weights w and a non-negative target t,

Time: O(N * max(w_i))

various/fast_knapsack.py

```
"""
Author: Mårten Wiman
License: CC0
Source: Pisinger 1999, "Linear Time Algorithms for Knapsack
Problems with Bounded Weights"
Description: Given N non-negative integer weights w and a non-
negative target t,
computes the maximum S <= t such that S is the sum of some subset
of the weights.
Time: O(N * max(w_i))
Status: Tested on kattis:eavesdropperevasion, stress-tested
"""

from typing import List

def fast_knapsack(w: List[int], t: int) -> int:
    """
    Fast subset sum / knapsack solver.
    w = list of weights
    t = target sum
    Returns maximum sum <= t that can be achieved
    """
    a = 0
    b = 0

    # Greedy phase
    while b < len(w) and a + w[b] <= t:
        a += w[b]
        b += 1

    if b == len(w):
        return a

    # DP phase
    m = max(w)
    v = [-1] * (2 * m)
    v[a + m - t] = b

    for i in range(b, len(w)):
        u = v[:]
        for x in range(m):
            if v[x + w[i]] < u[x]:
                v[x + w[i]] = u[x]

        x = 2 * m - 1
        while x > m:
            if u[x] >= 0:
                for j in range(max(0, u[x]), min(len(w), v[x])):
                    if v[x - w[j]] < j:
                        v[x - w[j]] = j
            x -= 1

    # Find maximum achievable sum
    a = t
    while a >= 0 and v[a + m - t] < 0:
        a -= 1

    return a
```

Fast Mod Note

various/fast_mod_note.py

```
"""
Fast Modulo in Python
=====================

The C++ FastMod uses Barrett reduction with __uint128_t for
ultra-fast modulo
operations. Python doesn't have this low-level optimization, but
here are alternatives:

Method 1: Native Python (Simple, usually sufficient)
----------------------------------------------------
```python
# Python's modulo is already quite optimized
result = a % b
```

Method 2: Barrett Reduction (For repeated mods by same constant)

```python
class FastMod:
    '''Barrett reduction for fast modulo with constant divisor'''

    def __init__(self, b: int):
        self.b = b
        # Precompute m = floor(2^64 / b)
        self.m = (1 << 64) // b

    def reduce(self, a: int) -> int:
        '''Compute a % b quickly'''
        # q = floor(a * m / 2^64)
        q = (a * self.m) >> 64
        # a - q * b is in range [0, 2*b)
        r = a - q * self.b
        return r if r < self.b else r - self.b

# Usage
fm = FastMod(1000000007)
result = fm.reduce(12345678901234567890)
```

Method 3: Precomputed Powers (For modular exponentiation)

```python
def mod_pow(base: int, exp: int, mod: int) -> int:
    '''Fast modular exponentiation'''
    result = 1
    base %= mod
    while exp > 0:
        if exp & 1:
            result = (result * base) % mod
        base = (base * base) % mod
        exp >>= 1
    return result
```

Method 4: NumPy (For bulk operations)

```python
import numpy as np

# For arrays of numbers
arr = np.array([1000000, 2000000, 3000000])
# ... (continued)
```
```

Interval Container

Add and remove intervals from a set of disjoint intervals.

Time: O(log N)

various/interval_container.py

```
"""
Author: Simon Lindholm
License: CC0
Description: Add and remove intervals from a set of disjoint
intervals.
Will merge the added interval with any overlapping intervals in
the set when adding.
Intervals are [inclusive, exclusive).
Time: O(log N)
Status: stress-tested
"""

from sortedcontainers import SortedSet
from typing import Tuple

class IntervalContainer:
    """Maintain a set of disjoint intervals"""
    def __init__(self):
        self.intervals = SortedSet()

    def add_interval(self, L: int, R: int):
        """Add interval [L, R) to the set, merging with
overlapping intervals"""
        if L == R:
            return

        # Find overlapping intervals
        to_remove = []
        for interval in self.intervals:
            if interval[0] > R:
                break
            if interval[1] >= L:
                L = min(L, interval[0])
                R = max(R, interval[1])
                to_remove.append(interval)

        # Remove overlapping intervals
        for interval in to_remove:
            self.intervals.remove(interval)

        # Add merged interval
        self.intervals.add((L, R))

    def remove_interval(self, L: int, R: int):
        """Remove interval [L, R) from the set"""
        if L == R:
            return

        # Find intervals that overlap with [L, R)
        to_remove = []
        to_add = []

        for interval in self.intervals:
            if interval[0] >= R:
                break
            if interval[1] <= L:
                continue

            to_remove.append(interval)

            # Add parts that don't overlap
            if interval[0] < L:
                to_add.append((interval[0], L))
# ... (continued)
```

Interval Cover

Compute indices of smallest set of intervals covering another interval.

Time: O(N log N)

various/interval_cover.py

```
"""
Author: Johan Sannemo
License: CC0
Description: Compute indices of smallest set of intervals
covering another interval.
Intervals should be [inclusive, exclusive).
Returns empty set on failure (or if G is empty).
Time: O(N log N)
Status: Tested on kattis:intervalcover
"""

from typing import List, Tuple, TypeVar

T = TypeVar('T')

def interval_cover(G: Tuple[T, T], I: List[Tuple[T, T]]) ->
List[int]:
    """
    Find minimum set of intervals from I that cover interval G.
    G: interval to cover (inclusive, exclusive)
    I: list of available intervals (inclusive, exclusive)
    Returns: indices of intervals that form the cover, or [] if
impossible
    """
    S = sorted(range(len(I)), key=lambda i: I[i])
    R = []
    cur = G[0]
    at = 0

    while cur < G[1]:
        mx = (cur, -1)
        while at < len(I) and I[S[at]][0] <= cur:
            mx = max(mx, (I[S[at]][1], S[at]))
            at += 1

        if mx[1] == -1:
            return []
        cur = mx[0]
        R.append(mx[1])

    return R
```

Lis

Compute indices for the longest increasing subsequence.

Time: O(N log N)

various/lis.py

```
"""
Author: Johan Sannemo
License: CC0
Description: Compute indices for the longest increasing
subsequence.
Time: O(N log N)
Status: Tested on kattis:longincsubseq, stress-tested
"""

from typing import List, TypeVar
import bisect

T = TypeVar('T')

def lis(S: List[T]) -> List[int]:
    """
    Find longest increasing subsequence.
    Returns list of indices in S forming the LIS.
    For longest non-decreasing, use bisect_right instead of
bisect_left.
    """
    if not S:
        return []

    prev = [0] * len(S)
    res = []  # list of (value, index) pairs

    for i in range(len(S)):
        # For longest non-decreasing, change bisect_left to
bisect_right
        pos = bisect.bisect_left(res, (S[i], 0))

        if pos == len(res):
            res.append((S[i], i))
        else:
            res[pos] = (S[i], i)

        prev[i] = 0 if pos == 0 else res[pos - 1][1]

    L = len(res)
    cur = res[-1][1]
    ans = [0] * L
    while L > 0:
        L -= 1
        ans[L] = cur
        cur = prev[cur]

    return ans
```

Loop Unrolling Note

various/loop_unrolling_note.py

````
"""
Loop Unrolling in Python
========================

The C++ Unrolling pattern manually unrolls loops for performance.
Python handles this differently:

C++ Unrolling Pattern:
----------------------
```cpp
#define F {...; ++i;}
int i = from;
while (i&3 && i < to) F // for alignment
while (i + 4 <= to) { F F F F }
while (i < to) F
```

Python Equivalents:
-------------------

1. NumPy (Automatically Optimized)
----------------------------------
```python
import numpy as np

# Python loops are slow, but NumPy is fast (internally unrolled)
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
result = arr * 2 + 1  # Automatically vectorized and unrolled
```

2. Manual Unrolling (Rarely Needed)
-----------------------------------
```python
def process_unrolled(data, func):
    '''Manually unroll loop by factor of 4'''
    i = 0
    n = len(data)

    # Process 4 at a time
    while i + 4 <= n:
        func(data[i])
        func(data[i + 1])
        func(data[i + 2])
        func(data[i + 3])
        i += 4

    # Handle remainder
    while i < n:
        func(data[i])
        i += 1

# Example
total = 0
def add_to_total(x):
    global total
    total += x

process_unrolled([1, 2, 3, 4, 5, 6, 7], add_to_total)
```

# ... (continued)
````

Memory Optimization Note

various/memory_optimization_note.py

````
"""
Memory Optimization in Python
=============================

C++ has BumpAllocator, SmallPtr, and other memory optimization
techniques.
Python manages memory differently, but here are equivalent
optimizations:

1. Object Pooling (Similar to BumpAllocator)
--------------------------------------------
```python
class ObjectPool:
    '''Reuse objects instead of allocating new ones'''
    def __init__(self, factory, initial_size=100):
        self.factory = factory
        self.pool = [factory() for _ in range(initial_size)]
        self.in_use = []

    def acquire(self):
        if self.pool:
            obj = self.pool.pop()
        else:
            obj = self.factory()
        self.in_use.append(obj)
        return obj

    def release(self, obj):
        self.in_use.remove(obj)
        self.pool.append(obj)

    def release_all(self):
        self.pool.extend(self.in_use)
        self.in_use.clear()

# Usage
class Node:
    def __init__(self):
        self.value = 0
        self.left = None
        self.right = None

pool = ObjectPool(Node, 1000)
node1 = pool.acquire()
node2 = pool.acquire()
# ... use nodes ...
pool.release(node1)
pool.release(node2)
```

2. __slots__ (Reduce Memory per Object)
---------------------------------------
```python
# Without __slots__: each object has a __dict__
class NodeNormal:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

# With __slots__: ~40% less memory
class NodeOptimized:
# ... (continued)
````

Simd Note

various/simd_note.py

````
"""
SIMD (Single Instruction Multiple Data) in Python
=================================================

The C++ SIMD code uses Intel SSE/AVX intrinsics for vectorized
operations.
Python doesn't have direct access to these low-level
instructions, but there
are several alternatives:

Method 1: NumPy (Recommended for numerical computing)
-----------------------------------------------------
NumPy uses SIMD under the hood and is highly optimized.

```python
import numpy as np

# Vectorized operations (automatically use SIMD when available)
a = np.array([1, 2, 3, 4, 5, 6, 7, 8], dtype=np.int32)
b = np.array([8, 7, 6, 5, 4, 3, 2, 1], dtype=np.int32)

# Element-wise operations (SIMD optimized)
c = a + b
d = a * b
e = np.maximum(a, b)
f = np.sum(a)

# Dot product (highly optimized)
dot_product = np.dot(a, b)
```

Method 2: Numba (JIT compilation with SIMD)
-------------------------------------------
Numba can auto-vectorize loops and generate SIMD instructions.

```python
from numba import jit, vectorize, int32

@jit(nopython=True, fastmath=True)
def filtered_dot_product(a, b):
    '''Compute sum of a[i] * b[i] where a[i] < b[i]'''
    result = 0
    for i in range(len(a)):
        if a[i] < b[i]:
            result += a[i] * b[i]
    return result

# Numba will vectorize this if possible
a = np.array([1, 2, 3, 4], dtype=np.int32)
b = np.array([4, 3, 2, 1], dtype=np.int32)
result = filtered_dot_product(a, b)
```

Method 3: Direct SIMD with python-simd (experimental)
-----------------------------------------------------
```python
# Requires: pip install simd
import simd

# This is experimental and platform-specific
```

# ... (continued)
````
````

Ternary Search

Time: O(log(b-a))

various/ternary_search.py

```
"""
Author: Simon Lindholm
Date: 2015-05-12
License: CC0
Source: own work
Description:
Find the smallest i in [a,b] that maximizes f(i), assuming that
f(a) < ... < f(i) >= ... >= f(b).
To minimize f, change < to >.
Time: O(log(b-a))
Status: tested
"""

from typing import Callable

def ternary_search(a: int, b: int, f: Callable[[int], float]) ->
int:
    """
    Find the index that maximizes f in range [a, b].
    Assumes f is unimodal (increases then decreases).
    """
    assert a <= b
    while b - a >= 5:
        mid = (a + b) // 2
        if f(mid) < f(mid + 1):
            a = mid
        else:
            b = mid + 1

    # Linear search in remaining range
    best = a
    for i in range(a + 1, b + 1):
        if f(best) < f(i):
            best = i

    return best
```