Team notebook

University of Bern — Aarebitrary

November 21, 2025

Contents

1 Data Structures

1.1 Disjoint Set Union

```python
"""Disjoint Set Union (Union-Find) with union
    by size and path compression.

Usage example:
    >>> dsu = DSU(5)
    >>> dsu.unite(0,1)
    True
    >>> dsu.same(0,2)
    False

This structure supports: find(x), unite(a,b),
    same(a,b).
"""

class DSU:
    def __init__(self, n):
        self.p = list(range(n))
        self.sz = [1] * n

    def find(self, x):
        while x != self.p[x]:
            self.p[x] = self.p[self.p[x]]
            x = self.p[x]
        return x

    def unite(self, a, b):
        a, b = self.find(a), self.find(b)
        if a == b: return False
```

```
        if self.sz[a] < self.sz[b]:
            a, b = b, a
        self.p[b] = a
        self.sz[a] += self.sz[b]
        return True

    def same(self, a, b):
        return self.find(a) == self.find(b)
```

1.2 Fenwick Tree

```
"""Fenwick Tree (Binary Indexed Tree)
    supports point add and prefix sums.

Usage example:
    >>> fw = Fenwick(5)
    >>> fw.add(2, 3)
    >>> fw.sum(3)
    3

This implementation is 0-indexed and provides
    `add`, `sum` (prefix [0, pos)),
`range_sum(l, r)` (sum over [l, r)), and
    `lower_bound`.
"""

class Fenwick:
    # 0-indexed, supports prefix sums on [0, i)
    def __init__(self, n):
        self.s = [0] * n

    def add(self, pos, delta):
        # a[pos] += delta
        n = len(self.s)
        while pos < n:
            self.s[pos] += delta
            pos |= pos + 1

    def sum(self, pos):
        # sum of [0, pos)
        res = 0
        while pos > 0:
            res += self.s[pos - 1]
            pos &= pos - 1
        return res

    def range_sum(self, l, r): # [l, r)
        return self.sum(r) - self.sum(l)

    def lower_bound(self, target):
        # min pos s.t. sum[0..pos] >= target,
            returns n if none
        if target <= 0: return -1
        n = len(self.s)
        pos = 0
        pw = 1 << (n.bit_length())
        while pw:
            nxt = pos + pw
            if nxt <= n and self.s[nxt - 1] <
                target:
                target -= self.s[nxt - 1]
                pos = nxt
            pw >>= 1
        return pos # in [0..n]
```

1.3 Segment Tree Lazy

```
"""Segment tree with lazy propagation: range
    add & range sum (0-indexed).

Usage example:
        >>> st = SegTreeLazy(8)
        >>> st.add(1,4,5)      # add 5 to
            indices [1,4)
        >>> st.range_sum(0,5)
        15

Methods:
    - `add(L, R, val)`: add `val` to [L, R)
    - `range_sum(L, R)`: return sum over [L, R)
"""

class SegTreeLazy:
    def __init__(self, n):
        self.n = 1
        while self.n < n:
            self.n <<= 1
        self.sum = [0] * (2 * self.n)
        self.lazy = [0] * (2 * self.n)

    def _apply(self, k, l, r, val):
        self.sum[k] += val * (r - l)
        self.lazy[k] += val

    def _push(self, k, l, r):
        if self.lazy[k] != 0 and k < self.n:
            m = (l + r) // 2
            self._apply(k * 2, l, m,
                self.lazy[k])
            self._apply(k * 2 + 1, m, r,
                self.lazy[k])
            self.lazy[k] = 0

    def add(self, L, R, val, k=1, l=0, r=None):
        if r is None:
            r = self.n
        if R <= l or r <= L:
            return
        if L <= l and r <= R:
            self._apply(k, l, r, val)
            return
        self._push(k, l, r)
        m = (l + r) // 2
        self.add(L, R, val, k * 2, l, m)
        self.add(L, R, val, k * 2 + 1, m, r)
        self.sum[k] = self.sum[k * 2] +
            self.sum[k * 2 + 1]

    def range_sum(self, L, R, k=1, l=0,
        r=None):
        if r is None:
            r = self.n
        if R <= l or r <= L:
            return 0
        if L <= l and r <= R:
            return self.sum[k]
        self._push(k, l, r)
        m = (l + r) // 2
        return self.range_sum(L, R, k * 2, l,
            m) + self.range_sum(L, R, k * 2 +
            1, m, r)
```

1.4 Segment Tree

```python
"""Segment Tree supporting an associative
    operation (default: min).

Usage example:
    >>> st = SegTree(8)
    >>> st.build([5,2,7,1,3])
    >>> st.query(1,4) # min on [1,4)
    1

Construct with `SegTree(n, func, unit)`, then
    `build`, `update`, and `query`.
"""

class SegTree:
    # supports any associative op, default: min
    def __init__(self, n, func=min, unit=INF):
        self.N = 1
        while self.N < n:
            self.N <<= 1
        self.f = func
        self.unit = unit
        self.st = [unit] * (2 * self.N)

    def build(self, arr):
        for i, v in enumerate(arr):
            self.st[self.N + i] = v
        for i in range(self.N - 1, 0, -1):
            self.st[i] = self.f(self.st[2 * i],
                self.st[2 * i + 1])

    def update(self, pos, val):
        i = self.N + pos
        self.st[i] = val
        i >>= 1
        while i:
            self.st[i] = self.f(self.st[2 * i],
                self.st[2 * i + 1])
            i >>= 1

    def query(self, l, r):
        # [l, r)
        resl = self.unit
        resr = self.unit
        l += self.N
        r += self.N
        while l < r:
            if l & 1:
                resl = self.f(resl, self.st[l])
                l += 1
            if r & 1:
                r -= 1
                resr = self.f(self.st[r], resr)
            l >>= 1
            r >>= 1
        return self.f(resl, resr)
```

1.5 Sparse Table

```python
"""Sparse Table for static range queries with
    idempotent ops (min/max/gcd).

Usage example:
    >>> st = SparseTable([1,5,2,4,3], func=min)
    >>> st.query(1,4) # min on [1,4)
    2

Build with `SparseTable(arr, func)`, then
    query with `query(l,r)`.
"""

class SparseTable:
    # for static array, idempotent op like
        min/max/gcd
    def __init__(self, arr, func=min):
        self.f = func
        n = len(arr)
        self.log = [0]*(n+1)
        for i in range(2, n+1):
            self.log[i] = self.log[i//2] + 1
        K = self.log[n] + 1
        st = [arr[:]]
        j = 1
        while (1 << j) <= n:
            prev = st[j-1]
            cur = []
            step = 1 << j
            half = step >> 1
            for i in range(0, n - step + 1):
                cur.append(self.f(prev[i],
                    prev[i + half]))
            st.append(cur)
            j += 1
        self.st = st

    def query(self, l, r):
        # [l, r) (like KACTL RMQ)
        length = r - l
        k = self.log[length]
        return self.f(self.st[k][l],
            self.st[k][r - (1 << k)])
```

2 Geometry

2.1 2D Basics

```python
def cross(o, a, b):
    """2D cross product (OA x OB). >0 if OAB
        is counter-clockwise."""
    return (a[0]-o[0])*(b[1]-o[1]) -
        (a[1]-o[1])*(b[0]-o[0])

def convex_hull(points):
    """Monotone chain; returns hull in CCW
        order (no repeated first point)."""
    points = sorted(set(points))
    if len(points) <= 1:
        return points
    lower = []
    for p in points:
        while len(lower) >= 2 and
            cross(lower[-2], lower[-1], p) <=
            0:
            lower.pop()
        lower.append(p)
    upper = []
    for p in reversed(points):
        while len(upper) >= 2 and
            cross(upper[-2], upper[-1], p) <=
```

```
        0:
            upper.pop()
        upper.append(p)
    return lower[:-1] + upper[:-1]
```

3 Graphs

3.1 BFS and DFS

```
"""Breadth-First Search (unweighted
    distances) and Depth-First Search helpers.

Usage example:
    >>> g = [[1,2],[0,3],[0],[1]]
    >>> bfs(0, g) # distances from node 0
    [0,1,1,2]

Note: this BFS assumes an unweighted graph
    (neighbors listed directly).
For weighted graphs use Dijkstra.
"""

from collections import deque

##############################
# Iterative BFS
##############################
def bfs(start, n, adj):
    """
    Iterative BFS from 'start'.
    Returns distance array; -1 means
        unreachable.
    """
    dist = [-1] * n
    dist[start] = 0
    q = deque([start])

    while q:
        u = q.popleft()
        for v in adj[u]:
            if dist[v] == -1:
                dist[v] = dist[u] + 1
                q.append(v)

    return dist


##############################
# recursive BFS
##############################
def bfs(start, g):
    n = len(g)
    dist = [INF]*n
    dist[start] = 0
    dq = deque([start])
    while dq:
        u = dq.popleft()
        for v in g[u]:
            if dist[v] == INF:
                dist[v] = dist[u] + 1
                dq.append(v)
    return dist


##############################
# Iterative DFS using a stack
##############################
def dfs_iter(start, n, adj):
    """
    Iterative DFS from 'start' using an
        explicit stack.
    Returns visit order (optional; you can
        instead do some processing).
    """
    visited = [False] * n
    order = []

    stack = [start]
    while stack:
        u = stack.pop()
        if visited[u]:
            continue
        visited[u] = True
        order.append(u)
        # For the same order as recursive DFS,
            push neighbors in reverse
        for v in reversed(adj[u]):
            if not visited[v]:
                stack.append(v)

    return order

##############################
# recursive DFS
##############################
def dfs(u, g, vis):
    vis[u] = True
    for v in g[u]:
        if not vis[v]:
            dfs(v, g, vis)
```

3.2 Dijkstra

```
"""Dijkstra's algorithm for single-source
    shortest paths on non-negative weighted
    graphs.

Graph representation: `g[u] = [(v, w), ...]`

Usage example:
    >>> g = [[(1,2),(2,5)],[(2,1)],[]]
    >>> dijkstra(0, g)
    [0,2,3]
"""

def dijkstra(start, g):
    n = len(g)
    dist = [INF]*n
    dist[start] = 0
    pq = [(0, start)]
    while pq:
        d, u = heapq.heappop(pq)
        if d != dist[u]:
            continue
        for v, w in g[u]:
            nd = d + w
            if nd < dist[v]:
```

```
            dist[v] = nd
            heapq.heappush(pq, (nd, v))
    return dist
```

3.3 Dinic

```
from collections import deque


class Dinic:
    class Edge:
        __slots__ = ("to", "rev", "cap")

        def __init__(self, to, rev, cap):
            self.to = to
            self.rev = rev
            self.cap = cap

    def __init__(self, n):
        self.n = n
        self.g = [[] for _ in range(n)]

    def add_edge(self, u, v, cap):
        self.g[u].append(self.Edge(v,
            len(self.g[v]), cap))
        self.g[v].append(self.Edge(u,
            len(self.g[u]) - 1, 0))

    def max_flow(self, s, t):
        flow = 0
        INF = 10 ** 18
        while True:
            level = [-1] * self.n
            q = deque([s])
            level[s] = 0
            while q:
                u = q.popleft()
                for e in self.g[u]:
                    if e.cap > 0 and
                        level[e.to] < 0:
                        level[e.to] = level[u] +
                            1
                        q.append(e.to)
            if level[t] < 0:
                break
            it = [0] * self.n

            def dfs(u, f):
                if u == t:
                    return f
                for i in range(it[u],
                     len(self.g[u])):
                    it[u] = i
                    e = self.g[u][i]
                    if e.cap > 0 and
                        level[e.to] == level[u]
                        + 1:
                        pushed = dfs(e.to,
                            min(f, e.cap))
                        if pushed:
                            e.cap -= pushed
                            self.g[e.to][e.rev].cap
                                += pushed
                            return pushed
                return 0

            while True:
                pushed = dfs(s, INF)
                if not pushed:
                    break
                flow += pushed
        return flow
```

3.4 Kruskal MST

```
"""Kruskal's algorithm to compute Minimum
    Spanning Tree (MST).

Edges should be a list of `(w, u, v)` tuples.
    Returns `(total_weight, used_edges)`.

Usage example:
    >>> edges = [(1,0,1),(2,1,2),(3,0,2)]
    >>> kruskal(3, edges)
    (3, [(0, 1, 1), (1, 2, 2)])
"""

def kruskal(n, edges):
    # edges: (w, u, v)
    dsu = DSU(n)
    edges.sort()
    total = 0
    used = []
    for w, u, v in edges:
        if dsu.unite(u, v):
            total += w
            used.append((u, v, w))
    return total, used
```

3.5 Strongly Connected Components

```
"""Kosaraju's algorithm to compute Strongly
    Connected Components (SCCs).

Returns `(comp, cid)` where `comp[v]` is
    component id for vertex `v` in
`[0..cid-1]`.

Usage example:
    >>> g = [[1],[2],[0,3],[4],[]]
    >>> scc(g)[1]
    5
"""

def scc(graph):
    n = len(graph)
    rg = [[] for _ in range(n)]
    for u in range(n):
        for v in graph[u]:
            rg[v].append(u)

    vis = [False]*n
    order = []

    def dfs1(u):
        vis[u] = True
        for v in graph[u]:
            if not vis[v]:
```

```
            dfs1(v)
        order.append(u)

    for i in range(n):
        if not vis[i]:
            dfs1(i)

    comp = [-1]*n
    cid = 0

    def dfs2(u, cid):
        comp[u] = cid
        for v in rg[u]:
            if comp[v] == -1:
                dfs2(v, cid)

    for u in reversed(order):
        if comp[u] == -1:
            dfs2(u, cid)
            cid += 1

    return comp, cid # comp[i] in [0..cid-1]
```

3.6 Topo Sort

```
"""Topological sort for a DAG. Returns a
    topological ordering of nodes.

Usage example:
    >>> g = [[1],[2],[]]
    >>> topo_sort(g)
    [0,1,2]

If a cycle exists the returned list will have
    length < n.
"""

def topo_sort(g):
    n = len(g)
    indeg = [0]*n
    for u in range(n):
        for v in g[u]:
            indeg[v] += 1
    q = deque([i for i in range(n) if indeg[i]
        == 0])
    order = []
    while q:
        u = q.popleft()
        order.append(u)
        for v in g[u]:
            indeg[v] -= 1
            if indeg[v] == 0:
                q.append(v)
    return order # len < n if cycle
```

4 Libs Usages

4.1 Array

```
##############################
# 10. array & others (less common but handy)
##############################
from array import array

# Memory-compact numeric array
a = array('i', [0]) * n   # signed int
a[i] = 5

# operator: function versions of +, -, *, etc.
import operator as op
op.add(x, y)
op.mul(x, y)
```

4.2 Bisect

```
##############################
# 3. bisect (binary search on sorted lists)
##############################
import bisect

a = [1, 2, 4, 4, 5]
i = bisect.bisect_left(a, x) # first index >=
    x
j = bisect.bisect_right(a, x) # first index >
    x
# or:
i = bisect.bisect(a, x)     # alias of
    bisect_right

# Insert while keeping sorted
bisect.insort_left(a, x)
bisect.insort_right(a, x)

# Typical pattern: check existence / counts
exists = (i < len(a) and a[i] == x)
count_x = bisect.bisect_right(a, x) -
    bisect.bisect_left(a, x)
```

4.3 Collections

```
##############################
# 1. collections
##############################
from collections import deque, defaultdict,
    Counter, namedtuple

# deque: queue / stack with O(1) push/pop on
    both ends
dq = deque()
dq.append(x)     # push right
dq.appendleft(x) # push left
dq.pop()         # pop right
dq.popleft()     # pop left
dq[0], dq[-1]    # front, back

# defaultdict: auto-create missing keys (e.g.
    list, int)
g = defaultdict(list)
g[u].append(v)
cnt = defaultdict(int)
cnt[key] += 1    # starts from 0

# Counter: frequency map, multiset operations
c = Counter(a_list)
```

```
c[key]            # count
c.most_common(1) # [(value, freq)]
c1 + c2           # add multisets
c1 & c2           # intersection (min counts)

# namedtuple: lightweight struct-like objects
Point = namedtuple('Point', ['x', 'y'])
p = Point(3, 4)
p.x, p.y
```

4.4 Fast IO

```
##############################
# 0. FAST I/O & SETUP
##############################
import sys
input = sys.stdin.readline # faster input
print = sys.stdout.write # optional: manual
    '\n'

# For deep recursion (DFS on big trees)
sys.setrecursionlimit(10**7)
```

4.5 Fraction and Decimal

```
##############################
# 7. fractions / decimal (exact / high
    precision)
##############################
from fractions import Fraction

f = Fraction(1, 3) + Fraction(2, 5) # exact
    rational arithmetic
f.numerator, f.denominator

# decimal (if you really need precise
    decimals; slower than float)
from decimal import Decimal, getcontext
getcontext().prec = 50
x = Decimal('0.1') + Decimal('0.2')
```

4.6 Functools

```
##############################
# 6. functools (lru_cache, reduce)
##############################
from functools import lru_cache, reduce

# Memoized recursion (DP)
@lru_cache(maxsize=None)
def f(i, j):
    ...
    return ans

# reduce: fold (e.g. xor of list)
import operator as op
from functools import reduce
xor_all = reduce(op.xor, arr, 0)
```

4.7 Heapq

```
##############################
# 2. heapq (priority queue)
##############################
import heapq

# Min-heap (default)
h = []
heapq.heappush(h, (dist, node))
d, u = heapq.heappop(h)

# Initialize from list
h = [5, 1, 7]
heapq.heapify(h)
heapq.heappush(h, 3)
x = heapq.heappop(h) # smallest element

# Max-heap trick: store negatives
h = []
heapq.heappush(h, -value)
max_val = -heapq.heappop(h)
```

4.8 Itertools

```
##############################
# 4. itertools (combinatorics & sequences)
##############################
import itertools as it

# permutations, combinations, product
for p in it.permutations(arr, r):   # r-length
    permutations
    ...
for c in it.combinations(arr, r):   #
    combinations (no repeat)
    ...
for c in
    it.combinations_with_replacement(arr, r):
    ...
for prod in it.product(A, B, repeat=2):
    ...

# accumulate (prefix sums)
from itertools import accumulate
pref = list(accumulate(a))          # pref[i]
    = sum(a[:i+1])

# groupby (group consecutive equal keys)
for key, group_iter in
    it.groupby(sorted_pairs, key=lambda x:
    x[0]):
    group = list(group_iter)

# infinite iterators
it.count(start=0, step=1)           #
    0,1,2,3,...
it.cycle([0, 1])                    #
    0,1,0,1,...
it.repeat(x, times)                 # x,x,x,...
```

4.9 Math

```
##############################
# 5. math (number theory / geometry)
##############################
```

```python
import math

math.gcd(a, b)
math.lcm(a, b)        # Python 3.9+
math.isqrt(n)         # integer sqrt
math.sqrt(x)          # float sqrt
math.factorial(n)
math.comb(n, k)       # n choose k (exact
    integer)
math.perm(n, k)       # permutations (3.8+)
math.hypot(x, y)      # sqrt(x*x + y*y)
math.pi, math.tau, math.e

# Angle <-> radians
math.radians(deg)
math.degrees(rad)

# Useful for EPS in geometry
EPS = 1e-9
```

4.10 Random

```python
##############################
# 8. random (randomized algorithms)
##############################
import random

random.seed(0)            # fix seed
r = random.randint(a, b)  # a <= r <= b
r = random.randrange(n)   # 0 <= r < n
random.shuffle(a_list)
random.choice(a_list)
```

4.11 String

```python
##############################
# 9. string (character sets)
##############################
import string

lower = string.ascii_lowercase    #
    'abcdefghijklmnopqrstuvwxyz'
upper = string.ascii_uppercase
digits = string.digits            #
    '0123456789'
alpha = string.ascii_letters      # upper +
    lower
```

5 Math

5.1 Number Theory

```python
"""Common number theory utilities: gcd, lcm,
    extended gcd, modular inverse, pow.

Usage examples:
   >>> gcd(6,8)
   2
   >>> modinv(3, 11)
   4
"""

def gcd(a, b):
   while b:
      a, b = b, a % b
   return abs(a)

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

def extgcd(a, b):
   if b == 0:
      return a, 1, 0
   g, x1, y1 = extgcd(b, a % b)
   return g, y1, x1 - (a // b) * y1

def modinv(a, m=MOD):
   g, x, _ = extgcd(a, m)
   if g != 1:
      return None
   return x % m

def modpow(a, e, m=MOD):
   r = 1
   a %= m
   while e:
      if e & 1:
         r = r * a % m
      a = a * a % m
      e >>= 1
   return r
```

5.2 nCr

```python
"""Precompute factorials and inverse
    factorials to compute nCr modulo MOD.

Usage example:
   >>> nCr(5,2)
   10

Adjust 'NMAX' if you need larger
    precomputation limits.
"""

# Precompute up to N
NMAX = 2 * 10**5
fact = [1] * (NMAX + 1)
invfact = [1] * (NMAX + 1)
for i in range(1, NMAX + 1):
   fact[i] = fact[i - 1] * i % MOD
invfact[NMAX] = modpow(fact[NMAX], MOD - 2)
for i in range(NMAX, 0, -1):
   invfact[i - 1] = invfact[i] * i % MOD

def nCr(n, r):
   if r < 0 or r > n: return 0
   return fact[n] * invfact[r] % MOD *
      invfact[n - r] % MOD
```

5.3 primality

```python
"""Primality testing (deterministic
    Miller-Rabin for 64-bit) and Pollard Rho
factorization utilities.

Usage examples:
    >>> is_prime(101)
    True
    >>> factorize(91)
    {7:1,13:1}
"""

import random

def sieve_primes(n):
    """Returns (primes, is_prime[0..n])."""
    is_prime = [True] * (n+1)
    is_prime[0] = is_prime[1] = False
    primes = []
    for i in range(2, n+1):
        if is_prime[i]:
            primes.append(i)
            step = i
            start = i * i
            if start > n:
                continue
            for j in range(start, n+1, step):
                is_prime[j] = False
    return primes, is_prime

def _is_prime_small(n):
    if n < 2:
        return False
    small_primes = [2, 3, 5, 7, 11, 13, 17,
        19, 23, 29]
    for p in small_primes:
        if n % p == 0:
            return n == p
    return None


def is_prime(n):
    """Deterministic Miller-Rabin for 64-bit
        integers."""
    sp = _is_prime_small(n)
    if sp is not None:
        return sp
    d = n - 1
    s = 0
    while d % 2 == 0:
        d //= 2
        s += 1
    # bases for deterministic up to 2^64
    for a in [2, 325, 9375, 28178, 450775,
        9780504, 1795265022]:
        if a % n == 0:
            continue
        x = pow(a, d, n)
        if x == 1 or x == n - 1:
            continue
        composite = True
        for _ in range(s - 1):
            x = (x * x) % n
            if x == n - 1:
                composite = False
                break
        if composite:
            return False
    return True


def pollards_rho(n):
    if n % 2 == 0:
        return 2
    if n % 3 == 0:
        return 3
    while True:
        c = random.randrange(1, n - 1)
        x = random.randrange(2, n - 1)
        y = x
        d = 1
        while d == 1:
            x = (x * x + c) % n
            y = (y * y + c) % n
            y = (y * y + c) % n
            d = math_gcd(abs(x - y), n)
            if d == n:
                break
        if d > 1 and d < n:
            return d

def factorize(n):
    """Return prime factors as a dict {prime:
        exponent}.
    Uses Pollard Rho + Miller-Rabin.
    """
    if n == 1:
        return {}
    if is_prime(n):
        return {n: 1}
    d = pollards_rho(n)
    while d is None:
        d = pollards_rho(n)
    a = factorize(d)
    b = factorize(n // d)
    for k, v in b.items():
        a[k] = a.get(k, 0) + v
    return a


def math_gcd(a, b):
    while b:
        a, b = b, a % b
    return abs(a)
```

6 Matrice

6.1 Determinant of Matrix

```python
from math import fabs

# ========================
#  Determinant (double)
# ========================

def det_double(a):
    """
    Determinant of a square matrix of floats.
    Destroys 'a' in-place (Gaussian
        elimination with partial pivoting).
    """
    n = len(a)
```

```
    res = 1.0
    for i in range(n):
        # find pivot row b
        b = i
        for j in range(i + 1, n):
            if fabs(a[j][i]) > fabs(a[b][i]):
                b = j
        if i != b:
            a[i], a[b] = a[b], a[i]
            res *= -1.0
        res *= a[i][i]
        if res == 0:
            return 0.0
        # eliminate below
        for j in range(i + 1, n):
            v = a[j][i] / a[i][i]
            if v != 0.0:
                for k in range(i + 1, n):
                    a[j][k] -= v * a[i][k]
    return res


# =========================
#  IntDeterminant (modular)
# =========================

MOD_DEFAULT = 12345 # same as in KACTL

def det_int(a, mod=MOD_DEFAULT):
    """
    Determinant of an integer matrix modulo
        'mod'.
    Destroys 'a' in-place.
    """
    n = len(a)
    ans = 1 % mod
    for i in range(n):
        for j in range(i + 1, n):
            # gcd-like elimination step
            while a[j][i] % mod != 0:
                # integer division like C++
                    (floor towards 0)
                t = a[i][i] // a[j][i]
                if t != 0:
                    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 = -ans
        ans = (ans * (a[i][i] % mod)) % mod
        if ans == 0:
            return 0
    return (ans + mod) % mod
```

6.2 Matrix Inverse

```
# =========================
#  MatrixInverse (double)
# =========================

def mat_inv(A, eps=1e-12):
    """
    In-place inversion of a square matrix A
        (double).
    On success: A becomes A^{-1}, return rank
        (= n).
    If singular: returns rank < n and A is
        undefined for inversion.
    """
    n = len(A)
    col = list(range(n))
    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 with max abs(A[j][k]) in
            submatrix
        r = c = i
        for j in range(i, n):
            for k in range(i, n):
                if fabs(A[j][k]) >
                    fabs(A[r][c]):
                    r, c = j, k
        if fabs(A[r][c]) < eps:
            return i # rank i < n

        # swap row r <-> i in both A and tmp
        A[i], A[r] = A[r], A[i]
        tmp[i], tmp[r] = tmp[r], tmp[i]

        # swap columns c <-> i in both A and
            tmp
        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]

        v = A[i][i]
        # eliminate below
        for j in range(i + 1, n):
            f = A[j][i] / v
            A[j][i] = 0.0
            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]

        # normalize row i
        for j in range(i + 1, n):
            A[i][j] /= v
        for j in range(n):
            tmp[i][j] /= v
        A[i][i] = 1.0

    # eliminate above
    for i in range(n - 1, 0, -1):
        for j in range(i):
            v = A[j][i]
            for k in range(n):
                tmp[j][k] -= v * tmp[i][k]

    # reorder columns back according to col[]
    res = [[0.0] * n for _ in range(n)]
    for i in range(n):
        for j in range(n):
            res[col[i]][col[j]] = tmp[i][j]

    # copy back into A
    for i in range(n):
        for j in range(n):
            A[i][j] = res[i][j]
```

```
    return n # full rank
```

6.3 Solve Linear

```
# ========================
#  SolveLinear (double)
# ========================

def solve_linear(A, b, eps=1e-12):
    """
    Solve A * x = b (double).
    A: n x m list of lists (will be destroyed).
    b: length-n list (will be destroyed).
    Returns (rank, x), where rank = -1 if no
        solution.
    If multiple solutions, returns one
        arbitrary solution.
    """
    n = len(A)
    m = len(A[0]) if n else 0
    rank = 0
    col = list(range(m)) # column permutation

    for i in range(n):
        # find pivot with max abs value in
            submatrix
        br = bc = i
        bv = 0.0
        for r in range(i, n):
            for c in range(i, m):
                v = fabs(A[r][c])
                if v > bv:
                    bv, br, bc = v, r, c

        if bv <= eps:
            # check for inconsistency
            for j in range(i, n):
                if fabs(b[j]) > eps:
                    return -1, None # no
                        solution
            break

        # swap rows and columns to put pivot
            at (i, i) (after col perm)
        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 below
        pivot_inv = 1.0 / A[i][i]
        for j in range(i + 1, n):
            fac = A[j][i] * pivot_inv
            b[j] -= fac * b[i]
            for k in range(i + 1, m):
                A[j][k] -= fac * A[i][k]

        rank += 1

    # back substitution
    x = [0.0] * m
    for i in reversed(range(rank)):
        b[i] /= A[i][i]
        x[col[i]] = b[i]
        for j in range(i):
            b[j] -= A[j][i] * b[i]

    return rank, x


# ========================
#  SolveLinear2 (uniquely determined values)
# ========================

def solve_linear_unique(A, b, eps=1e-12):
    """
    Variant of solve_linear: only returns
        values that are uniquely determined.
    Undetermined variables get None.
    A, b are destroyed.
    Returns (rank, x_unique) where x_unique[j]
        is either a float or None.
    """
    n = len(A)
    m = len(A[0]) if n else 0
    rank = 0
    col = list(range(m))

    # same pivoting as solve_linear, but
        eliminate against ALL rows
    for i in range(n):
        br = bc = i
        bv = 0.0
        for r in range(i, n):
            for c in range(i, m):
                v = fabs(A[r][c])
                if v > bv:
                    bv, br, bc = v, r, c

        if bv <= eps:
            for j in range(i, n):
                if fabs(b[j]) > eps:
                    return -1, None
            break

        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]

        pivot_inv = 1.0 / A[i][i]
        # eliminate in ALL other rows (j != i)
        for j in range(n):
            if j == i:
                continue
            fac = A[j][i] * pivot_inv
            b[j] -= fac * b[i]
            for k in range(i + 1, m):
                A[j][k] -= fac * A[i][k]

        rank += 1

    # Now A is almost diagonal in pivot
        columns; detect uniquely determined
        vars
    x = [None] * m
    for i in range(rank):
```

```
        # If any free variable (column >=
            rank) appears in row i, it's not
            unique
        if any(fabs(A[i][j]) > eps for j in
            range(rank, m)):
            continue
        pivot_col = col[i]
        x[pivot_col] = b[i] / A[i][i]

    return rank, x


# =========================
#  SolveLinearBinary (over F2)
# =========================

def _first_set_bit_at_or_after(mask, start,
    m):
    """Return index of first set bit >= start,
        or m if none."""
    for i in range(start, m):
        if (mask >> i) & 1:
            return i
    return m

def solve_linear_binary(A, b, m):
    """
    Solve A x = b over F2.
    A: list of ints, each int's bits represent
        a row of length m (0/1).
    b: list of ints (0 or 1).
    Returns (rank, x_mask) where x_mask is an
        int with bits of solution.
    Returns (-1, None) if no solution.
    Destroys A and b.
    """
    n = len(A)
    rank = 0
    col = list(range(m))

    i = 0
    while i < n:
        # find row with any nonzero entry
            among remaining rows
        br = i
        while br < n and A[br] == 0:
            br += 1
        if br == n:
            # no rows with nonzero entries
                left; check for inconsistency
            for j in range(i, n):
                if b[j] & 1:
                    return -1, None
            break

        # pivot column: first set bit in row
            br at or after i
        bc = _first_set_bit_at_or_after(A[br],
            i, m)
        if bc == m:
            # row has no set bit, but row != 0
                should not happen here;
                continue
            i += 1
            continue

        A[i], A[br] = A[br], A[i]
        b[i], b[br] = b[br], b[i]
        col[i], col[bc] = col[bc], col[i]

        # swap bits i and bc in all rows
            (simulate column permutation)
        for j in range(n):
            bit_i = (A[j] >> i) & 1
            bit_bc = (A[j] >> bc) & 1
            if bit_i != bit_bc:
                A[j] ^= (1 << i)
                A[j] ^= (1 << bc)

        # eliminate below
        for j in range(i + 1, n):
            if ((A[j] >> i) & 1) == 1:
                b[j] ^= b[i]
                A[j] ^= A[i]

        rank += 1
        i += 1

    # back-substitution
    x_mask = 0
    for i in reversed(range(rank)):
        if not (b[i] & 1):
            continue
        pivot_col = col[i]
        x_mask |= (1 << pivot_col)
        # subtract this row from all above
            (since pivot is 1)
        for j in range(i):
            if ((A[j] >> i) & 1) == 1:
                b[j] ^= 1

    return rank, x_mask
```

6.4 Tridiagonal Solver

```
# =========================
#  Tridiagonal solver
# =========================

def tridiagonal(diag, super_diag, sub_diag,
    b):
    """
    Solve tridiagonal system with main
        diagonal 'diag',
    super-diagonal 'super_diag', sub-diagonal
        'sub_diag' and RHS b.
    All are lists of floats. Returns x
        (solution), leaves copies of inputs.
    This matches the KACTL algorithm,
        including the special stability trick.
    """
    n = len(b)
    diag = diag[:]        # copy, we will
        modify
    b = b[:]              # copy
    tr = [0] * n          # "swap-trick" flags

    # forward elimination
    i = 0
    while i < n - 1:
        if abs(diag[i]) < 1e-9 *
            abs(super_diag[i]): # diag[i] == 0
            (numerically)
```

```
            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]
            tr[i + 1] = 1
            i += 2 # note the ++i in C++ after
                setting tr[++i]
        else:
            diag[i + 1] -= super_diag[i] *
                sub_diag[i] / diag[i]
            b[i + 1] -= b[i] * sub_diag[i] /
                diag[i]
            i += 1

    # backward substitution
    for i in range(n - 1, -1, -1):
        if tr[i]:
            # swap b[i] and b[i-1]; diag[i-1] =
                diag[i]; divide by
                super_diag[i-1]
            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 > 0:
                b[i - 1] -= b[i] * super_diag[i
                    - 1]

    return b
```

7 Prefix Sums

7.1 2D Prefix Sums

```
"""2D prefix sums helper for fast submatrix
    sum queries.

Usage example:
    >>> mat = [[1,2],[3,4]]
    >>> sm = SubMatrix(mat)
    >>> sm.sum(0,0,2,2)
    10

Query uses half-open ranges: `sum(u, l, d,
    r)` returns sum over rows [u,d) and cols
    [l,r).
"""

class SubMatrix:
    # prefix sums on matrix, query sum over
        [u:d) x [l:r)
    def __init__(self, v):
        R, C = len(v), len(v[0])
        p = [[0]*(C+1) for _ in range(R+1)]
        for r in range(R):
            pr = p[r+1]
            for c in range(C):
                pr[c+1] = v[r][c] + p[r][c+1] +
                    p[r+1][c] - p[r][c]
        self.p = p

    def sum(self, u, l, d, r):
        p = self.p
        return p[d][r] - p[d][l] - p[u][r] +
            p[u][l]
```

8 Search

8.1 Binary Search

```
import bisect

"""Binary search utilities: find lower/upper
    bounds and boolean condition searches.

Usage examples:
    >>> a = [1,3,5,7]
    >>> lower_bound(a,5)
    2

`bin_search_low`/`bin_search_high` expect an
    `ok(x)` predicate and search an integer
    interval.
"""

def bin_search_low(lo, hi, ok):
    # find min x in [lo, hi] with ok(x) True
    while lo < hi:
        mid = (lo + hi) // 2
        if ok(mid): hi = mid
        else: lo = mid + 1
    return lo

def bin_search_high(lo, hi, ok):
    # find max x in [lo, hi] with ok(x) True
    while lo < hi:
        mid = (lo + hi + 1) // 2
        if ok(mid): lo = mid
        else: hi = mid - 1
    return lo


def lower_bound(a, x):
    return bisect.bisect_left(a, x)

def upper_bound(a, x):
    return bisect.bisect_right(a, x)
```

8.2 Ternary Search

```
# Discrete ternary search on [lo, hi] for a
    unimodal function f: int -> value
# Assumes f is FIRST strictly increasing,
    THEN strictly decreasing (single peak).
# Returns (argmax_x, f(argmax_x)).
def ternary_search_discrete(lo, hi, f):
    while hi - lo > 3:
        m1 = lo + (hi - lo) // 3
        m2 = hi - (hi - lo) // 3
        f1 = f(m1)
        f2 = f(m2)
        if f1 < f2:    # for maximum
            lo = m1 + 1
```

```
        else:
            hi = m2 - 1

    # brute-force the tiny remaining range
    best_x = lo
    best_val = f(lo)
    for x in range(lo + 1, hi + 1):
        val = f(x)
        if val > best_val:
            best_val = val
            best_x = x
    return best_x, best_val

# Example usage:
# a is unimodal array, we want index of
    maximum:
# idx, val = ternary_search_discrete(0,
    len(a) - 1, lambda i: a[i])


# Continuous ternary search on [lo, hi] for a
    unimodal f: float -> float.
# Returns (x_opt, f(x_opt)) approximately.
def ternary_search_continuous(lo, hi, f,
    iterations=80):
    for _ in range(iterations):
        m1 = (2 * lo + hi) / 3.0
        m2 = (lo + 2 * hi) / 3.0
        f1 = f(m1)
        f2 = f(m2)
        if f1 < f2:     # for maximum
            lo = m1
        else:
            hi = m2
    x_opt = (lo + hi) / 2.0
    return x_opt, f(x_opt)

# Example:
# def f(x): return - (x - 3) ** 2 + 5 #
    maximum at x = 3
# x_opt, y_opt =
    ternary_search_continuous(0.0, 10.0, f)
```

9 Strings

9.1 Aho Corasick

```
from collections import deque


""" AhoCorasick  automaton for multi-pattern
    string matching.

Usage example:
    >>> ac = AhoCorasick()
    >>> ac.add("he", 0)
    >>> ac.add("she", 1)
    >>> ac.build()
    >>> ac.search("she")
    [(1, 0), (2, 1)]

‘add(pattern, index)‘, ‘build()‘, then
    ‘search(text)‘ returning list of (pos,
    pat_idx).
"""

class AhoCorasick:
    def __init__(self):
        self.next = [{}]
        self.link = [0]
        self.out = [[]]

    def add(self, s, idx):
        v = 0
        for ch in s:
            if ch not in self.next[v]:
                self.next[v][ch] =
                    len(self.next)
                self.next.append({})
                self.link.append(0)
                self.out.append([])
            v = self.next[v][ch]
        self.out[v].append(idx)

    def build(self):
        q = deque()
        for ch, v in self.next[0].items():
            q.append(v)
            self.link[v] = 0
        while q:
            v = q.popleft()
            for ch, u in self.next[v].items():
                q.append(u)
                j = self.link[v]
                while j and ch not in
                    self.next[j]:
                    j = self.link[j]
                self.link[u] =
                    self.next[j].get(ch, 0)
                self.out[u] +=
                    self.out[self.link[u]]

    def search(self, text):
        v = 0
        res = [] # list of (pos, pattern_index)
        for i, ch in enumerate(text):
            while v and ch not in self.next[v]:
                v = self.link[v]
            v = self.next[v].get(ch, 0)
            for pat in self.out[v]:
                res.append((i, pat))
        return res
```

9.2 KMP

```
"""  KnuthMorrisPratt   (KMP) prefix-function
    and matcher.

Usage example:
    >>> kmp_match("abababa", "aba")
    [0,2,4]

‘prefix_function(s)‘ computes the pi array;
    ‘kmp_match(text, pat)‘ returns starting
    indices.
"""

def prefix_function(s):
    n = len(s)
    p = [0]*n
```

```
    for i in range(1, n):
        j = p[i - 1]
        while j and s[i] != s[j]:
            j = p[j - 1]
        if s[i] == s[j]:
            j += 1
        p[i] = j
    return p

def kmp_match(text, pat):
    if not pat:
        return list(range(len(text) + 1))
    s = pat + "#" + text
    p = prefix_function(s)
    res = []
    m = len(pat)
    for i in range(m + 1, len(s)):
        if p[i] == m:
            res.append(i - 2*m)
    return res # starting indices
```

9.3 Manarcher

```
"""Manacher's algorithm to compute
    palindromic radii (odd/even centers).

Usage example:
    >>> p0, p1 = manacher("abba")
    # p1 contains odd radii, p0 even radii

Returns `(p0, p1)` where `p0[i]` is
    even-radius at i and `p1[i]` odd-radius.
"""

def manacher(s):
    n = len(s)
    # p[0][i]: even, p[1][i]: odd radii
    p0 = [0]*n # even
    p1 = [0]*n # odd

    # odd
    l = r = 0
    for i in range(n):
        k = 1 if i > r else min(p1[l + r - i],
            r - i + 1)
        while i - k >= 0 and i + k < n and
            s[i-k] == s[i+k]:
            k += 1
        p1[i] = k
        if i + k - 1 > r:
            l, r = i - k + 1, i + k - 1

    # even
    l = r = 0
    for i in range(n):
        k = 0 if i > r else min(p0[l + r - i +
            1], r - i + 1)
        while i - k - 1 >= 0 and i + k < n and
            s[i-k-1] == s[i+k]:
            k += 1
        p0[i] = k
        if i + k - 1 > r:
            l, r = i - k, i + k - 1

    return p0, p1
```

9.4 Z Function

```
"""Z-function: for each position i, longest
    substring starting at i matching prefix.

Usage example:
    >>> z_function("abacaba")
    [0,0,1,0,3,0,1]

Returns list `z` of length n where z[i] is
    the match length at i.
"""

def z_function(s):
    n = len(s)
    z = [0]*n
    l = r = 0
    for i in range(1, n):
        if i < r:
            z[i] = min(r - i, z[i - l])
        while i + z[i] < n and s[z[i]] == s[i
            + z[i]]:
            z[i] += 1
        if i + z[i] > r:
            l, r = i, i + z[i]
    return z
```

10 Trees

10.1 LCA

```
"""Lowest Common Ancestor (LCA) using binary
    lifting.

Usage example:
    >>> adj = [[1,2],[0],[0]]
    >>> up, depth = build_lca(adj, 0)
    >>> lca(1,2,up,depth)
    0

Functions: `build_lca(adj, root)` returns
    `(up, depth)`, then use
    `lca(u,v,up,depth)`.
"""

def build_lca(adj, root=0):
    n = len(adj)
    LOG = max(1, (n).bit_length())
    up = [[root] * n for _ in range(LOG)]
    depth = [0] * n
    parent = [-1] * n
    parent[root] = root
    stack = [root]
    while stack:
        u = stack.pop()
        for v in adj[u]:
            if v == parent[u]:
                continue
            parent[v] = u
            depth[v] = depth[u] + 1
            up[0][v] = u
            stack.append(v)
```

```python
    up[0][root] = root
    for k in range(1, LOG):
        for v in range(n):
            up[k][v] = up[k - 1][up[k - 1][v]]
    return up, depth

def lca(u, v, up, depth):
    if depth[u] < depth[v]:
        u, v = v, u
    LOG = len(up)
    diff = depth[u] - depth[v]
    for k in range(LOG):
        if (diff >> k) & 1:
            u = up[k][u]
    if u == v:
        return u
    for k in range(LOG - 1, -1, -1):
        if up[k][u] != up[k][v]:
            u = up[k][u]
            v = up[k][v]
    return up[0][u]
```

11 Utils

11.1 Header and IO

```python
"""Common header and IO helpers used across
    cheatsheet examples.

Provides ‘INF‘, ‘MOD‘, and simple input
    helpers:
       - ‘ints()‘ -> list of ints from a line
       - ‘int1()‘ -> single int
       - ‘strs()‘ -> list of strings

Usage:
               >>> # import these helpers in
                   your scripts
               >>> # from Header and IO import
                   ints, INF
"""

import sys, math, random, bisect, heapq
from collections import deque, defaultdict,
    Counter
sys.setrecursionlimit(10**7)
input = sys.stdin.readline


INF = 10**18
MOD = 10**9 + 7 # or 998244353

# Read helpers
def ints(): return list(map(int,
    input().split()))
def int1(): return int(input())
def strs(): return input().strip().split()
```