

Chapter 1

Contest

Template.py

```
1 import sys
2 from collections import *
3 from itertools import permutations #No repeated elements
4 sys.setrecursionlimit(10**5)
5 itr = (line for line in sys.stdin.read().strip().split('\n'))
6 INP = lambda: next(itr)
7 def ni(): return int(INP())
8 def nl(): return [int(_) for _ in INP().split()]
9
10
11
12 def solve(n,a):
13     pass
14
15
16 t = ni()
17 for case in range(t):
18     n = ni()
19     a = nl()
20     solve(n,a)
```

Troubleshooting: Pre-submit: Write a few simple test cases if sample is not enough. Are time limits close? If so, generate max cases. Is the memory usage fine? Could anything overflow? Make sure to submit the right file.

Wrong answer: Print your solution! Print debug output, as well. Are you clearing all data structures between test cases? Can your algorithm handle the whole range of input? Read the full problem statement again. Do you handle all corner cases correctly? Have you understood the problem correctly? Any uninitialized variables? Any overflows? Confusing N and M, i and j, etc.? Are you sure your algorithm works? What special cases have you not thought of? Are you sure the STL functions you use work as you think? Add some assertions, maybe resubmit. Create some test-cases to run your algorithm on. Go through the algorithm for a simple case. Go through this list again. Explain your algorithm

to a teammate. Ask the teammate to look at your code. Go for a small walk, e.g. to the toilet. Is your output format correct? (including whitespace) Rewrite your solution from the start or let a teammate do it.

Runtime error: Have you tested all corner cases locally? Any uninitialized variables? Are you reading or writing outside the range of any vector? Any assertions that might fail? Any possible division by 0? (mod 0 for example) Any possible infinite recursion? Invalidated pointers or iterators? Are you using too much memory? Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded: Do you have any possible infinite loops? What is the complexity of your algorithm? Are you copying a lot of unnecessary data? (References) How big is the input and output? (consider buffering output) What do your teammates think about your algorithm?

Memory limit exceeded: What is the max amount of memory your algorithm should need? Are you clearing all data structures between test cases?

Chapter 2

Mathematics

2.1 Equations

$$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The extremum is given by $x = -b/2a$.

$$\begin{aligned} ax + by &= e & x &= \frac{ed - bf}{ad - bc} \\ cx + dy &= f & y &= \frac{af - ec}{ad - bc} \end{aligned}$$

In general, given an equation $Ax = b$, the solution to a variable x_i is given by

$$x_i = \frac{\det A'_i}{\det A}$$

where A'_i is A with the i 'th column replaced by b .

2.2 Recurrences

If $a_n = c_1 a_{n-1} + \dots + c_k a_{n-k}$, and r_1, \dots, r_k are distinct roots of $x^k + c_1 x^{k-1} + \dots + c_k$, there are d_1, \dots, d_k s.t.

$$a_n = d_1 r_1^n + \dots + d_k r_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

2.3 Trigonometry

$$\sin(v + w) = \sin v \cos w + \cos v \sin w$$

$$\cos(v + w) = \cos v \cos w - \sin v \sin w$$

$$\tan(v + w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$

$$\sin v + \sin w = 2 \sin \frac{v + w}{2} \cos \frac{v - w}{2}$$

$$\cos v + \cos w = 2 \cos \frac{v + w}{2} \cos \frac{v - w}{2}$$

$$(V + W) \tan(v - w)/2 = (V - W) \tan(v + w)/2$$

where V, W are lengths of sides opposite angles v, w .

$$a \cos x + b \sin x = r \cos(x - \phi)$$

$$a \sin x + b \cos x = r \sin(x + \phi)$$

where $r = \sqrt{a^2 + b^2}, \phi = \text{atan2}(b, a)$.

2.4 Geometry

2.4.1 Triangles

Side lengths: a, b, c

Semiperimeter: $p = \frac{a + b + c}{2}$

Area: $A = \sqrt{p(p - a)(p - b)(p - c)}$

Circumradius: $R = \frac{abc}{4A}$

Inradius: $r = \frac{A}{p}$

Length of median (divides triangle into two equal-area triangles):

$$m_a = \frac{1}{2} \sqrt{2b^2 + 2c^2 - a^2}$$

Length of bisector (divides angles in two): $s_a =$

$$\sqrt{bc \left[1 - \left(\frac{a}{b+c} \right)^2 \right]}$$

$$\text{Law of sines: } \frac{\sin \alpha}{a} = \frac{\sin \beta}{b} = \frac{\sin \gamma}{c} = \frac{1}{2R}$$

$$\text{Law of cosines: } a^2 = b^2 + c^2 - 2bc \cos \alpha$$

$$\text{Law of tangents: } \frac{a+b}{a-b} = \frac{\tan \frac{\alpha+\beta}{2}}{\tan \frac{\alpha-\beta}{2}}$$

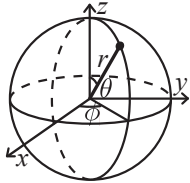
2.4.2 Quadrilaterals

With side lengths a, b, c, d , diagonals e, f , diagonals angle θ , area A and magic flux $F = b^2 + d^2 - a^2 - c^2$:

$$4A = 2ef \cdot \sin \theta = F \tan \theta = \sqrt{4e^2 f^2 - F^2}$$

For cyclic quadrilaterals the sum of opposite angles is 180° , $ef = ac + bd$, and $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

2.4.3 Spherical coordinates



$$\begin{aligned} x &= r \sin \theta \cos \phi & r &= \sqrt{x^2 + y^2 + z^2} \\ y &= r \sin \theta \sin \phi & \theta &= \arccos(z / \sqrt{x^2 + y^2 + z^2}) \\ z &= r \cos \theta & \phi &= \operatorname{atan2}(y, x) \end{aligned}$$

2.5 Derivatives/Integrals

$$\frac{d}{dx} \arcsin x = \frac{1}{\sqrt{1-x^2}} \quad \frac{d}{dx} \arccos x = -\frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx} \tan x = 1 + \tan^2 x \quad \frac{d}{dx} \arctan x = \frac{1}{1+x^2}$$

$$\int \tan ax = -\frac{\ln |\cos ax|}{a} \quad \int x \sin ax = \frac{\sin ax - ax \cos ax}{a^2}$$

$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2} \operatorname{erf}(x) \quad \int x e^{ax} dx = \frac{e^{ax}}{a^2} (ax - 1)$$

Integration by parts:

$$\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$$

2.6 Sums

$$c^a + c^{a+1} + \dots + c^b = \frac{c^{b+1} - c^a}{c - 1}, c \neq 1$$

$$1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$$

$$1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(2n+1)(n+1)}{6}$$

$$1^3 + 2^3 + 3^3 + \dots + n^3 = \frac{n^2(n+1)^2}{4}$$

$$1^4 + 2^4 + 3^4 + \dots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

2.7 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots, (-\infty < x < \infty)$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots, (-1 < x \leq 1)$$

$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \dots, (-1 \leq x \leq 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots, (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots, (-\infty < x < \infty)$$

2.8 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x . It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x x p_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent X and Y ,

$$V(aX + bY) = a^2 V(X) + b^2 V(Y).$$

2.8.1 Discrete distributions

Binomial distribution

The number of successes in n independent yes/no experiments, each which yields success with probability p is $\operatorname{Bin}(n, p)$, $n = 1, 2, \dots$, $0 \leq p \leq 1$.

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

$$\mu = np, \sigma^2 = np(1-p)$$

$\operatorname{Bin}(n, p)$ is approximately $\operatorname{Po}(np)$ for small p .

First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each which yields success with probability p is $\operatorname{Fs}(p)$, $0 \leq p \leq 1$.

$$p(k) = p(1-p)^{k-1}, k = 1, 2, \dots$$

$$\mu = \frac{1}{p}, \sigma^2 = \frac{1-p}{p^2}$$

Poisson distribution

The number of events occurring in a fixed period of time t if these events occur with a known average rate κ and independently of the time since the last event is $\operatorname{Po}(\lambda)$, $\lambda = t\kappa$.

$$p(k) = e^{-\lambda} \frac{\lambda^k}{k!}, k = 0, 1, 2, \dots$$

$$\mu = \lambda, \sigma^2 = \lambda$$

2.8.2 Continuous distributions

Uniform distribution

If the probability density function is constant between a and b and 0 elsewhere it is $\operatorname{U}(a, b)$, $a < b$.

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2}, \sigma^2 = \frac{(b-a)^2}{12}$$

Exponential distribution

The time between events in a Poisson process is $\operatorname{Exp}(\lambda)$, $\lambda > 0$.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$\mu = \frac{1}{\lambda}, \sigma^2 = \frac{1}{\lambda^2}$$

Normal distribution

Most real random values with mean μ and variance σ^2 are well described by $\mathcal{N}(\mu, \sigma^2)$, $\sigma > 0$.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

If $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$ then

$$aX_1 + bX_2 + c \sim \mathcal{N}(\mu_1 + \mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$

2.9 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let X_1, X_2, \dots be a sequence of random variables generated by the Markov process. Then there is a transition matrix $\mathbf{P} = (p_{ij})$, with $p_{ij} = \Pr(X_n = i | X_{n-1} = j)$, and $\mathbf{p}^{(n)} = \mathbf{P}^n \mathbf{p}^{(0)}$ is the probability distribution for X_n (i.e., $p_i^{(n)} = \Pr(X_n = i)$), where $\mathbf{p}^{(0)}$ is the initial distribution.

π is a stationary distribution if $\pi = \pi \mathbf{P}$. If the Markov chain is *irreducible* (it is possible to get to any state from any state), then $\pi_i = \frac{1}{\mathbb{E}(T_i)}$ where $\mathbb{E}(T_i)$ is the expected time between two visits in state i . π_j / π_i is the expected number of visits in state j between two visits in state i .

For a connected, undirected and non-bipartite graph, where the transition probability is uniform among all neighbors, π_i is proportional to node i 's degree.

A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1). $\lim_{k \rightarrow \infty} \mathbf{P}^k = \mathbf{1}\pi$.

A Markov chain is an A-chain if the states can be partitioned into two sets \mathbf{A} and \mathbf{G} , such that all states in \mathbf{A} are absorbing ($p_{ii} = 1$), and all states in \mathbf{G} leads to an absorbing state in \mathbf{A} . The probability for absorption in state $i \in \mathbf{A}$, when the initial state is $j \in \mathbf{G}$ is $a_{ij} = p_{ij} + \sum_{k \in \mathbf{G}} a_{ik} p_{kj}$. The expected time until absorption when the initial state is i , is $t_i = 1 + \sum_{k \in \mathbf{G}} p_{ki} t_k$.

The formula for elements in pascals triangel is

$$(n+1)C(r) = (n)C(r-1) + (n)C(r)$$

Chapter 3

Graph Algorithms

bfs:

```

1 # S is index, G is adjacency list
2 # finds distance from S to all verticies in G
3 def bfs(S, G):
4     q = [S]
5     INF = 10**18
6     dist = [INF]*len(G)
7     dist[S] = 0
8     while q:
9         q2 = []
10        for u in q:
11            for v in G[u]:
12                # early break here if only interested in length
13                if dist[u] + 1 < dist[v]:
14                    dist[v] = dist[u] + 1
15                    q2.append(v)
16        q = q2
17    return dist

```

dijkstra:

```

1 from heapq import heappop as pop, heappush as push
2 # adj: adj-list where edges are tuples (node_id, weight):
3 # (1) --2-- (0) --3-- (2) has the adj-list:
4 # adj = [(1, 2), (2, 3)], [(0, 2)], [0, 3]]
5 def dijk(adj, S, T):
6     N = len(adj)
7     INF = 10**18
8     dist = [INF]*N
9     pq = []
10    def add(i, dst):
11        if dst < dist[i]:
12            dist[i] = dst
13            push(pq, (dst, i))
14    add(S, 0)
15
16    while pq:
17        D, i = pop(pq)
18        if i == T: return D
19        if D != dist[i]: continue
20        for j, w in adj[i]:
21            add(j, D + w)
22
23    return dist[T]

```

twoSat:

```

1 # used in sevenkingdoms, illumination
2 import sys
3 sys.setrecursionlimit(10**5)
4 class Sat:
5     def __init__(self, no_vars):
6         self.size = no_vars*2
7         self.no_vars = no_vars
8         self.adj = [[] for _ in range(self.size)]
9         self.back = [[] for _ in range(self.size)]
10    def add_imply(self, i, j):
11        self.adj[i].append(j)
12        self.back[j].append(i)
13    def add_or(self, i, j):
14        self.add_imply(i^1, j)
15        self.add_imply(j^1, i)
16    def add_xor(self, i, j):
17        self.add_or(i, j)
18        self.add_or(i^1, j^1)
19    def add_eq(self, i, j):
20        self.add_xor(i, j^1)
21
22    def dfs1(self, i):
23        if i in self.marked: return
24        self.marked.add(i)
25        for j in self.adj[i]:
26            self.dfs1(j)
27        self.stack.append(i)
28
29    def dfs2(self, i):
30        if i in self.marked: return
31        self.marked.add(i)
32        for j in self.back[i]:
33            self.dfs2(j)
34        self.comp[i] = self.no_c
35
36    def is_sat(self):
37        self.marked = set()
38        self.stack = []
39        for i in range(self.size):
40            self.dfs1(i)
41        self.marked = set()
42        self.no_c = 0
43        self.comp = [0]*self.size
44        while self.stack:
45            i = self.stack.pop()
46            if i not in self.marked:
47                self.no_c += 1
48                self.dfs2(i)
49        for i in range(self.no_vars):
50            if self.comp[i*2] == self.comp[i*2+1]:
51                return False
52        return True
53
54    # assumes is_sat.
55    # If not xi is after xi in topological sort,
56    # xi should be FALSE. It should be TRUE otherwise.
57    # https://codeforces.com/blog/entry/16205
58    def solution(self):
59        V = []
60        for i in range(self.no_vars):
61            V.append(self.comp[i*2] > self.comp[i*2+1])
62        return V
63
64 if __name__ == '__main__':
65     S = Sat(1)

```

```

66 S.add_or(0, 0)
67 print(S.is_sat())
68 print(S.solution())

```

maxflow:

```

1 from collections import defaultdict
2 class Dinitz:
3     def __init__(self, sz, INF=10**10):
4         self.G = [defaultdict(int) for _ in range(sz)]
5         self.sz = sz
6         self.INF = INF
7
8     def add_edge(self, i, j, w):
9         self.G[i][j] += w
10
11     def bfs(self, s, t):
12         level = [0]*self.sz
13         q = [s]
14         level[s] = 1
15         while q:
16             q2 = []
17             for u in q:
18                 for v, w in self.G[u].items():
19                     if w and level[v] == 0:
20                         level[v] = level[u] + 1
21                         q2.append(v)
22             q = q2
23         self.level = level
24         return level[t] != 0
25
26     def dfs(self, s, t, FLOW):
27         if s in self.dead: return 0
28         if s == t: return FLOW
29
30         for idx in range(self.pos[s], len(self.adj[s])):
31             u = self.adj[s][idx]
32             w = self.G[s][u]
33             F = self.dfs(u, t, min(FLOW, w))
34             if F:
35                 self.G[s][u] -= F
36                 self.G[u][s] += F
37                 if self.G[s][u] == 0:
38                     self.pos[s] = idx+1
39                     if idx + 1 == len(self.adj[s]):
40                         self.dead.add(s)
41                 return F
42             self.pos[s] = idx+1
43             self.dead.add(s)
44         return 0
45
46     def setup_after_bfs(self):
47         self.adj = [[v for v, w in self.G[u].items() if w and
48 self.level[u] + 1 == self.level[v]] for u in range(self.sz)]
49         self.pos = [0]*self.sz
50         self.dead = set()
51     def max_flow(self, s, t):
52         flow = 0
53         while self.bfs(s, t):
54             self.setup_after_bfs()
55             while True:
56                 pushed = self.dfs(s, t, self.INF)
57                 if not pushed: break
58                 flow += pushed
59         return flow

```

hopcroftCarp:

```

1 # Hopcroft-Karp bipartite max-cardinality matching and max
   independent set
2 # David Eppstein, UC Irvine, 27 Apr 2002
3 # Used in https://open.kattis.com/problems/cuckoo
4 def bipartiteMatch(graph):
5     '''Find maximum cardinality matching of a bipartite graph (
6     U,V,E).
7     The input format is a dictionary mapping members of U to a
8     list
9     of their neighbors in V. The output is a triple (M,A,B)
10    where M is a
11    dictionary mapping members of V to their matches in U, A is
12    the part
13    of the maximum independent set in U, and B is the part of
14    the MIS in V.
15    The same object may occur in both U and V, and is treated
16    as two
17    distinct vertices if this happens.'''
18
19    # initialize greedy matching (redundant, but faster than
20    full search)
21    matching = {}
22    for u in graph:
23        for v in graph[u]:
24            if v not in matching:
25                matching[v] = u
26                break
27
28    while 1:
29        # structure residual graph into layers
30        # pred[u] gives the neighbor in the previous layer for
31        u in U
32        # preds[v] gives a list of neighbors in the previous
33        layer for v in V
34        # unmatched gives a list of unmatched vertices in final
35        layer of V,
36        # and is also used as a flag value for pred[u] when u
37        is in the first layer
38        preds = {}
39        unmatched = []
40        pred = dict([(u,unmatched) for u in graph])
41        for v in matching:
42            del pred[matching[v]]
43        layer = list(pred)
44
45        # repeatedly extend layering structure by another pair
46        of layers
47        while layer and not unmatched:
48            newLayer = {}
49            for u in layer:
50                for v in graph[u]:
51                    if v not in preds:
52                        newLayer.setdefault(v, []).append(u)
53            layer = []
54            for v in newLayer:
55                preds[v] = newLayer[v]
56            if v in matching:
57                layer.append(matching[v])
58            pred[matching[v]] = v
59        else:
60            unmatched.append(v)
61
62        # did we finish layering without finding any
63        alternating paths?
64        if not unmatched:

```

```

52 unlayered = {}
53 for u in graph:
54     for v in graph[u]:
55         if v not in preds:
56             unlayered[v] = None
57     return (matching, list(pred), list(unlayered))
58
59     # recursively search backward through layers to find
60     alternating paths
61     # recursion returns true if found path, false otherwise
62     def recurse(v):
63         if v in preds:
64             L = preds[v]
65             del preds[v]
66             for u in L:
67                 if u in pred:
68                     pu = pred[u]
69                     del pred[u]
70                     if pu is unmatched or recurse(pu):
71                         matching[v] = u
72                         return 1
73             return 0
74
75     for v in unmatched: recurse(v)

```

HungarianAlgorithm

```

1 # used on https://open.kattis.com/problems/arboriculture
2 # G is Bipartite graph N x M (N <= M) where [i][j] is cost to
3   match L[i] and R[j]
4 # Ported from: https://raw.githubusercontent.com/kth-
5   competitive-programming/kactl/main/content/graph/
6   WeightedMatching.h
7 # Description: Given a weighted bipartite graph, matches every
8   node on
9 # the left with a node on the right such that no
10 # nodes are in two matchings and the sum of the edge weights is
11   minimal. Takes
12 # cost[N][M], where cost[i][j] = cost for L[i] to be matched
13   with R[j] and
14 # Returns: (min cost, match), where L[i] is matched with R[
15   match[i]].
16 # Negate costs for max cost.
17 # Time: O(N^2M)
18
19 def hungarian(G):
20     INF = 10**18
21     if len(G) == 0:
22         return 0, []
23
24     n, m = len(G) + 1, len(G[0]) + 1
25     u, v, p = [0]*n, [0]*m, [0]*m
26     ans = [0]*(n-1)
27     for i in range(1, n):
28         p[0], j0 = i, 0
29         dist, pre = [INF]*m, [-1]*m
30         done = [False]*(m+1)
31         while True:
32             done[j0] = True
33             i0, j1, delta = p[j0], 0, INF
34             for j in range(1, m):
35                 if done[j]: continue
36                 cur = G[i0 - 1][j-1] - u[i0] - v[j]
37                 if cur < dist[j]:
38                     dist[j], pre[j] = cur, j0
39             if dist[j] < delta:
40                 delta, j1 = dist[j], j

```

```

34     for j in range(0, m):
35         if done[j]:
36             u[p[j]] += delta
37             v[j] -= delta
38         else:
39             dist[j] -= delta
40     j0 = j1
41     if p[j0] == 0: break
42     while j0:
43         j1 = pre[j0]
44         p[j0] = p[j1]
45         j0 = j1
46     return -v[0], ans

```

ShortestCycle

```

1 from collections import *
2 def shortest_cycle(G):
3     ''' Returns the length of shortest cycle even in an
4     undirected graph.
5     Floyd Warshall only handles directed graphs,
6     but considers an undirected edge to be a cycle of length 2.
7     G is adjacency list. '''
8     n = len(G)
9     ans = 10**18
10    INF = 10**9
11    for i in range(n):
12        dist = [INF] * n
13        par = [-1] * n
14        dist[i] = 0
15        q = deque()
16        q.append(i)
17        while q:
18            x = q[0]
19            q.popleft()
20
21            for child in G[x]:
22                if dist[child] == INF:
23                    dist[child] = 1 + dist[x]
24
25                par[child] = x
26                q.append(child)
27
28            elif par[x] != child and par[child] != x:
29                ans = min(ans, dist[x] +
30                        dist[child] + 1)
31
32    return ans

```

ConvexHull:

```

1 def convex_hull(pts):
2     pts = sorted(set(pts))
3
4     if len(pts) <= 2:
5         return pts
6
7     def cross(o, a, b):
8         return (a[0] - o[0]) * (b[1] - o[1]) - (a[1] - o[1]) *
9             (b[0] - o[0])
10
11    lo = []
12    for p in pts:
13        while len(lo) >= 2 and cross(lo[-2], lo[-1], p) <= 0:
14            lo.pop()
15        lo.append(p)

```

```

16    hi = []
17    for p in reversed(pts):
18        while len(hi) >= 2 and cross(hi[-2], hi[-1], p) <= 0:
19            hi.pop()
20        hi.append(p)
21
22    return lo[:-1] + hi[:-1]

```

Chapter 4

Data Structures

Segment tree:

```

1 # Tested on: https://open.kattis.com/problems/supercomputer
2 class SegmentTree:
3     def __init__(self, arr, func=min):
4         self.sz = len(arr)
5         assert self.sz > 0
6         self.func = func
7         sz4 = self.sz*4
8         self.L, self.R = [None]*sz4, [None]*sz4
9         self.value = [None]*sz4
10        def setup(i, lo, hi):
11            self.L[i], self.R[i] = lo, hi
12            if lo == hi:
13                self.value[i] = arr[lo]
14            return
15        mid = (lo + hi)//2
16        setup(2*i, lo, mid)
17        setup(2*i + 1, mid+1, hi)
18        self._fix(i)
19        setup(1, 0, self.sz-1)
20    def _fix(self, i):
21        self.value[i] = self.func(self.value[2*i], self.value
22            [2*i+1])
23
24    def _combine(self, a, b):
25        if a is None: return b
26        if b is None: return a
27        return self.func(a, b)
28
29    def query(self, lo, hi):
30        assert 0 <= lo <= hi < self.sz
31        return self.__query(1, lo, hi)
32
33    def __query(self, i, lo, hi):
34        l, r = self.L[i], self.R[i]
35        if r < lo or hi < l:
36            return None
37        if lo <= l <= r <= hi:

```

```

37        return self.value[i]
38    return self._combine(
39        self.__query(i*2, lo, hi),
40        self.__query(i*2 + 1, lo, hi)
41    )
42
43    def assign(self, pos, value):
44        assert 0 <= pos < self.sz
45        return self.__assign(1, pos, value)
46
47    def __assign(self, i, pos, value):
48        l, r = self.L[i], self.R[i]
49        if pos < l or r < pos: return
50        if pos == l == r:
51            self.value[i] = value
52            return
53        self.__assign(i*2, pos, value)
54        self.__assign(i*2 + 1, pos, value)
55        self._fix(i)
56
57    def inc(self, pos, delta):
58        assert 0 <= pos < self.sz
59        self.__inc(1, pos, delta)
60
61    def __inc(self, i, pos, delta):
62        l, r = self.L[i], self.R[i]
63        if pos < l or r < pos: return
64        if pos == l == r:
65            self.value[i] += delta
66            return
67        self.__inc(i*2, pos, delta)
68        self.__inc(i*2 + 1, pos, delta)
69        self._fix(i)
70
71    # for indexing - nice to have but not required
72    def __setitem__(self, i, v):
73        self.assign(i, v)
74    def __fixslice__(self, k):
75        return slice(k.start or 0, self.sz if k.stop == None
76            else k.stop)
77    def __getitem__(self, k):
78        if type(k) == slice:
79            k = self.__fixslice__(k)
80            return self.query(k.start, k.stop - 1)
81        elif type(k) == int:
82            return self.query(k, k)

```

Fenwick Tree:

```

1 # Tested on: https://open.kattis.com/problems/froshweek
2 class FenwickTree: # zero indexed calls!
3     # Give array or size!
4     def __init__(self, blob):
5         if type(blob) == int:
6             self.sz = blob
7             self.data = [0]*(blob+1)
8         elif type(blob) == list:
9             A = blob
10            self.sz = len(A)
11            self.data = [0]*(self.sz + 1)
12            for i, a in enumerate(A):
13                self.inc(i, a)
14
15    # A[i] = v
16    def assign(self, i, v):
17        currV = self.query(i, i)
18        self.inc(i, v - currV)
19
20    # A[i] += delta

```

```

19 # this method is ~3x faster than doing A[i] += delta
20 def inc(self, i, delta):
21     i += 1 # (to 1 indexing)
22     while i <= self.sz:
23         self.data[i] += delta
24         i += i&-i # lowest oneBit
25 # sum(A[:i+1])
26 def sum(self, i):
27     i += 1 # (to 1 indexing)
28     S = 0
29     while i > 0:
30         S += self.data[i]
31         i -= i&-i
32     return S
33 # return sum(A[lo:hi+1])
34 def query(self, lo, hi):
35     return self.sum(hi) - self.sum(lo-1)
36
37 # for indexing - nice to have but not required
38 def __fixslice__(self, k):
39     return slice(k.start or 0, self.sz if k.stop == None
40                 else k.stop)
41 def __setitem__(self, i, v):
42     self.assign(i, v)
43 def __getitem__(self, k):
44     if type(k) == slice:
45         k = self.__fixslice__(k)
46         return self.query(k.start, k.stop - 1)
47     elif type(k) == int:
48         return self.query(k, k)

```

RMQ:

```

1 import math
2 class RMQ:
3     def __init__(self, arr, func=min):
4         self.sz = len(arr)
5         self.func = func
6         MAXN = self.sz
7         LOGMAXN = int(math.ceil(math.log(MAXN + 1, 2)))
8         self.data = [[0]*LOGMAXN for _ in range(MAXN)]
9         for i in range(MAXN):
10             self.data[i][0] = arr[i]
11         for j in range(1, LOGMAXN):
12             for i in range(MAXN - (1<<j)+1):
13                 self.data[i][j] = func(self.data[i][j-1],
14                                         self.data[i + (1<<(j-1))][j-1])
15
16     def query(self, a, b):
17         if a > b:
18             # some default value when query is empty
19             return 1
20         d = b - a + 1
21         k = int(math.log(d, 2))
22         return self.func(self.data[a][k], self.data[b-(1<<k)
23                         +1][k])

```

Union Find:

```

1 class UnionFind:
2     def __init__(self, N):
3         self.parent = [i for i in range(N)]
4         self.sz = [1]*N
5     def find(self, i):
6         path = []
7         while i != self.parent[i]:

```

```

8         path.append(i)
9         i = self.parent[i]
10        for u in path: self.parent[u] = i
11        return i
12    def union(self, u, v):
13        uR, vR = map(self.find, (u, v))
14        if uR == vR: return False
15        if self.sz[uR] < self.sz[vR]:
16            self.parent[uR] = vR
17            self.sz[vR] += self.sz[uR]
18        else:
19            self.parent[vR] = uR
20            self.sz[uR] += self.sz[vR]
21        return True

```

Chapter 5

Div

Hungarian algorithm:

```

1 # G is Bipartite graph N x M (N <= M) where [i][j] is cost to
2   match L[i] and R[j]
3 # Description: Given a weighted bipartite graph, matches every
4   node on
5   the left with a node on the right such that no
6   nodes are in two matchings and the sum of the edge weights is
7   minimal. Takes
8   cost[N][M], where cost[i][j] = cost for L[i] to be matched
9   with R[j] and
10  Returns: (min cost, match), where L[i] is matched with R[
11            match[i]].
12  # Negate costs for max cost.
13  # Time: O(N^2M)
14
15 def hungarian(G):
16     INF = 10**18
17     if len(G) == 0:
18         return 0, []
19
20     n, m = len(G) + 1, len(G[0]) + 1
21     u, v, p = [0]*n, [0]*m, [0]*m
22     ans = [0]*(n-1)
23     for i in range(1, n):
24         p[0], j0 = i, 0
25         dist, pre = [INF]*m, [-1]*m
26         done = [False]*(m+1)
27         while True:
28             done[j0] = True
29             i0, j1, delta = p[j0], 0, INF
30             for j in range(1, m):

```

```

26         if done[j]: continue
27         cur = G[i0 - 1][j-1] - u[i0] - v[j]
28         if cur < dist[j]:
29             dist[j], pre[j] = cur, j0
30         if dist[j] < delta:
31             delta, j1 = dist[j], j
32         for j in range(0, m):
33             if done[j]:
34                 u[p[j]] += delta
35                 v[j] -= delta
36             else:
37                 dist[j] -= delta
38             j0 = j1
39             if p[j0] == 0: break
40         while j0:
41             j1 = pre[j0]
42             p[j0] = p[j1]
43             j0 = j1
44         return -v[0], ans

```

Gauss:

```

1 # monoid needs to implement
2 # __add__, __mul__, __sub__, __div__ and isZ
3 def gauss(A, b, monoid=None):
4     def Z(v): return abs(v) < 1e-6 if not monoid else v.isZ()
5
6     N = len(A[0])
7     for i in range(N):
8         try:
9             m = next(j for j in range(i, N) if Z(A[j][i]) ==
10                False)
11         except:
12             return None #A is not independent!
13         if i != m:
14             A[i], A[m] = A[m], A[i]
15             b[i], b[m] = b[m], b[i]
16         for j in range(i+1, N):
17             sub = A[j][i]/A[i][i]
18             b[j] -= sub*b[i]
19             for k in range(N):
20                 A[j][k] -= sub*A[i][k]
21
22     for i in range(N-1, -1, -1):
23         for j in range(N-1, i, -1):
24             sub = A[i][j]/A[j][j]
25             b[i] -= sub*b[j]
26             b[i], A[i][i] = b[i]/A[i][i], A[i][i]/A[i][i]
27     return b

```

FFT:

```

1 import cmath
2 # A has to be of length a power of 2.
3
4 def FFT(A, inverse=False):
5     N = len(A)
6     if N <= 1:
7         return A
8     if inverse:
9         D = FFT(A) # d_0/N, d_{N-1}/N, d_{N-2}/N, ...
10        return map(lambda x: x/N, [D[0]] + D[:0:-1])
11    evn = FFT(A[0::2])
12    odd = FFT(A[1::2])
13    Nh = N//2
14    return [evn[k%Nh]+cmath.exp(2j*cmath.pi*k/N)*odd[k%Nh]

```



```

15         for k in range(N):
16
17 # A has to be of length a power of 2.
18 def FFT2(a, inverse=False):
19     N = len(a)
20     j = 0
21     for i in range(1, N):
22         bit = N>>1
23         while j&bit:
24             j ^= bit
25             bit >>= 1
26         j ^= bit
27         if i < j:
28             a[i], a[j] = a[j], a[i]
29
30     L = 2
31     MUL = -1 if inverse else 1
32     while L <= N:
33         ang = 2j*cmath.pi/L * MUL
34         wlen = cmath.exp(ang)
35         for i in range(0, N, L):
36             w = 1
37             for j in range(L//2):
38                 u = a[i+j]
39                 v = a[i+j+L//2] * w
40                 a[i+j] = u + v
41                 a[i+j+L//2] = u - v
42                 w *= wlen
43             L *= 2
44     if inverse:
45         for i in range(N):
46             a[i] /= N
47     return a
48
49 def uP(n):
50     while n != (n&n):
51         n += n&n
52     return n
53
54 # C[x] = sum_{i=0..N}{A[x-i]*B[i]}
55 def polymul(A, B):
56     sz = 2*max(uP(len(A)), uP(len(B)))
57     A = A + [0]*(sz - len(A))
58     B = B + [0]*(sz - len(B))
59     fA = FFT(A)
60     fB = FFT(B)
61     fAB = [a*b for a, b in zip(fA, fB)]
62     C = [x.real for x in FFT(fAB, True)]
63     return C

```

Convex Hull:

```

1 def convex_hull(pts):
2     pts = sorted(set(pts))
3
4     if len(pts) <= 2:
5         return pts
6
7     def cross(o, a, b):
8         return (a[0] - o[0]) * (b[1] - o[1]) - (a[1] - o[1]) *
          (b[0] - o[0])
9
10    lo = []
11    for p in pts:
12        while len(lo) >= 2 and cross(lo[-2], lo[-1], p) <= 0:
13            lo.pop()
14        lo.append(p)

```

```

15    hi = []
16    for p in reversed(pts):
17        while len(hi) >= 2 and cross(hi[-2], hi[-1], p) <= 0:
18            hi.pop()
19            hi.append(p)
20
21    return lo[:-1] + hi[:-1]
22

```

Chapter 6

Geometry

Diverse:

```

1 import math
2
3 # Distance between two points
4 def dist(p, q):
5     return math.hypot(p[0]-q[0], p[1] - q[1])
6
7 # Square distance between two points
8 def d2(p, q):
9     return (p[0] - q[0])**2 + (p[1] - q[1])**2
10
11 # Converts two points to a line (a, b, c),
12 # ax + by + c = 0
13 # if p == q, a = b = c = 0
14 def pts2line(p, q):
15     return (-q[1] + p[1],
16            q[0] - p[0],
17            p[0]*q[1] - p[1]*q[0])
18
19 # Distance from a point to a line,
20 # given that a != 0 or b != 0
21 def dist1(l, p):
22     return (abs(l[0]*p[0] + l[1]*p[1] + l[2])
23            /math.hypot(l[0], l[1]))
24
25 # intersects two lines.
26 # if parallell, returnes False.
27 # lines on format (a, b, c) where ax + by + c == 0
28 def line_intersection(l1, l2):
29     a1,b1,c1 = l1
30     a2,b2,c2 = l2
31     cp = a1*b2 - a2*b1
32     if cp != 0:
33         return float(b1*c2 - b2*c1)/cp, float(a2*c1 - a1*c2)/cp
34     else:
35         return False
36

```

```

37 # projects a point on a line
38 def project(l, p):
39     a, b, c = l
40     return ((b*(b*p[0] - a*p[1]) - a*c)/(a*a + b*b),
41            (a*(a*p[1] - b*p[0]) - b*c)/(a*a + b*b))
42
43 # Intersections between circles
44 def circle_intersection(c1, c2):
45     if c1[2] > c2[2]:
46         c1, c2 = c2, c1
47     x1, y1, r1 = c1
48     x2, y2, r2 = c2
49     if x1 == x2 and y1 == y2 and r1 == r2:
50         return False
51
52     dist2 = (x1 - x2)*(x1-x2) + (y1 - y2)*(y1 - y2)
53     rsq = (r1 + r2)*(r1 + r2)
54     if dist2 > rsq or dist2 < (r1-r2)*(r1-r2):
55         return []
56     elif dist2 == rsq:
57         cx = x1 + (x2-x1)*r1/(r1+r2)
58         cy = y1 + (y2-y1)*r1/(r1+r2)
59         return [(cx, cy)]
60     elif dist2 == (r1-r2)*(r1-r2):
61         cx = x1 - (x2-x1)*r1/(r2-r1)
62         cy = y1 - (y2-y1)*r1/(r2-r1)
63         return [(cx, cy)]
64
65     d = math.sqrt(dist2)
66     f = (r1*r1 - r2*r2 + dist2)/(2*dist2)
67     xf = x1 + f*(x2-x1)
68     yf = y1 + f*(y2-y1)
69     dx = xf-x1
70     dy = yf-y1
71     h = math.sqrt(r1*r1 - dx*dx - dy*dy)
72     norm = abs(math.hypot(dx, dy))
73     p1 = (xf + h*(-dy)/norm, yf + h*(dx)/norm)
74     p2 = (xf + h*(dy)/norm, yf + h*(-dx)/norm)
75     return sorted([p1, p2])
76
77 # Finds the bisector through origo
78 # between two points by normalizing.
79 def bisector(p1, p2):
80     d1 = math.hypot(p1[0], p2[1])
81     d2 = math.hypot(p2[0], p2[1])
82     return ((p1[0]/d1 + p2[0]/d2),
83            (p1[1]/d1 + p2[1]/d2))
84
85 # Distance from P to origo
86 def norm(P):
87     return (P[0]**2 + P[1]**2 + P[2]**2)**(0.5)
88
89 # Finds distance between point p
90 # and line A + t*u in 3D
91 def dist3D(A, u, p):
92     AP = tuple(A[i] - p[i] for i in range(3))
93     cross = tuple(AP[i]*u[(i+1)%3] - AP[(i+1)%3]*u[i]
94                for i in range(3))
95     return norm(cross)/norm(u)
96
97 def vec(p1, p2):
98     return p2[0]-p1[0], p2[1] - p1[1]
99
100
101 def sign(x):
102     if x < 0: return -1
103     return 1 if x > 0 else 0

```

```

1004
1005 def cross(u, v):
1006     return u[0] * v[1] - u[1] * v[0]
1007
1008 def on_segment(p, q, r):
1009     """Check if point q lies on line segment 'pr'"""
1010     if (q[0] <= max(p[0], r[0]) and q[0] >= min(p[0], r[0]) and
1011         q[1] <= max(p[1], r[1]) and q[1] >= min(p[1], r[1])):
1012         return True
1013     return False
1014
1015 def is_segment_intersection(s1, s2):
1016     u = vec(*s1)
1017     v = vec(*s2)
1018     p1, p2 = s1
1019     q1, q2 = s2
1020
1021     # Calculate cross products
1022     d1 = cross(u, vec(p1, q1))
1023     d2 = cross(u, vec(p1, q2))
1024     d3 = cross(v, vec(q1, p1))
1025     d4 = cross(v, vec(q1, p2))
1026
1027     # Check general case
1028     if d1 != 0 or d2 != 0 or d3 != 0 or d4 != 0:
1029         return sign(d1) != sign(d2) and sign(d3) != sign(d4)
1030
1031     # Check collinear case
1032     return (on_segment(p1, q1, p2) or on_segment(p1, q2, p2) or
1033             on_segment(q1, p1, q2) or on_segment(q1, p2, q2))

```

```

15 ]
16
17 def getPrimesBelow(N):
18     primes = []
19     soll = [1]*N
20     for p in range(2, N):
21         if soll[p]:
22             primes.append(p)
23             for k in range(p*p, N, p):
24                 soll[k] = 0
25     return primes
26
27 def SieveOfEratosthenes(num):
28     prime = [True for i in range(num+1)]
29     # boolean array
30     p = 2
31     out = []
32     while (p * p <= num):
33
34         if (prime[p] == True):
35
36             # Updating all multiples of p
37             for i in range(p * p, num+1, p):
38                 prime[i] = False
39
40             p += 1
41
42     for p in range(2, num+1):
43         if prime[p]:
44             out.append(p)
45     return out
46
47
48 def isPrime(N):
49     if N < 2: return False
50     if N%2 == 0: return N == 2
51     mx = min(int(N**.5) + 2, N)
52     for i in range(3, mx, 2):
53         if N % i == 0: return False
54     return True
55
56 def genPrimesFrom(N):
57     while True:
58         if isPrime(N):
59             yield N
60         N += 1
61
62 def getPrimesFrom(N, cnt):
63     itr = genPrimesFrom(N)
64     return [next(itr) for _ in range(cnt)]
65
66 def is_power_of_two(num):
67     return num > 0 and (num & (num - 1)) == 0

```

Some useful functions:

```

1 import math
2
3 # Evaluates to n! / (k! * (n - k)!) when k <= n and evaluates
4 # to zero when k > n.
5 # math.comb(n, k) #introduced in python3.8
6
7 # math.gcd(a, b)
8 def gcd(a, b):
9     return b if a%b == 0 else gcd(b, a%b)
10
11 def lcm ( a, b) :

```

```

11     return a / math.gcd(a, b) * b
12
13
14 # returns b where (a*b)%MOD == 1
15 def inv(a, MOD):
16     return pow(a, -1, MOD)
17
18 # returns g = gcd(a, b), x0, y0,
19 # where g = x0*a + y0*b
20 def xgcd(a, b):
21     x0, x1, y0, y1 = 1, 0, 0, 1
22     while b != 0:
23         q, a, b = (a // b, b, a % b)
24         x0, x1 = (x1, x0 - q * x1)
25         y0, y1 = (y1, y0 - q * y1)
26     return (a, x0, y0)
27
28 def crt(la, ln):
29     assert len(la) == len(ln)
30     for i in range(len(la)):
31         assert 0 <= la[i] < ln[i]
32     prod = 1
33     for n in ln:
34         assert gcd(prod, n) == 1
35         prod *= n
36     lN = []
37     for n in ln:
38         lN.append(prod//n)
39     x = 0
40     for i, a in enumerate(la):
41         print(lN[i], ln[i])
42         _, Mi, mi = xgcd(lN[i], ln[i])
43         x += a*Mi*ln[i]
44     return x % prod
45
46 # finds x^e mod m
47 # Or just pow(x, e, m)
48 def modpow(x, m, e):
49     res = 1
50     while e:
51         if e%2 == 1:
52             res = (res*x) % m
53         x = (x*x) % m
54         e = e//2
55     return res
56
57 # Divides a list of digits with an int.
58 # A lot faster than using bigint-division.
59 def div(L, d):
60     r = [0]*(len(L) + 1)
61     q = [0]*len(L)
62     for i in range(len(L)):
63         x = int(L[i]) + r[i]*10
64         q[i] = x//d
65         r[i+1] = x-q[i]*d
66     s = []
67     for i in range(len(L) - 1, 0, -1):
68         s.append(q[i]%10)
69         q[i-1] += q[i]//10
70
71     while q[0]:
72         s.append(q[0]%10)
73         q[0] = q[0]//10
74     s = s[::-1]
75     i = 0
76     while s[i] == 0:
77         i += 1

```

Chapter 7

Number theory

Primes:

```

1 large_primes = [
2 5915587277,
3 1500450271,
4 3267000013,
5 5754853343,
6 4093082899,
7 9576890767,
8 3628273133,
9 2860486313,
10 5463458053,
11 3367900313,
12 100000000000000061,
13 10**16 + 61,
14 10**17 + 3

```



```
78     return s[i:]
79
80 # Multiplies a list of digits with an int.
81 # A lot faster than using bigint-multiplication.
82 def mul(L, d):
83     r = [d*x for x in L]
84     s = []
85     for i in range(len(r) - 1, 0, -1):
86         s.append(r[i]%10)
87         r[i-1] += r[i]//10
88     while r[0]:
89         s.append(r[0]%10)
90         r[0] = r[0]//10
91     return s[::-1]
```