Chapter 1

Contest

Template.py

```python
import sys
from collections import *
from itertools import import permutations #No repeated elements
sys.setrecursionlimit(10**5)
itr = (line for line in sys.stdin.read().strip().split('\n'))
INP = lambda: next(itr)
def ni(): return int(INP())
def nl(): return [int(_) for _ in INP().split()]


def solve(n,a):
    pass



t = ni()
for case in range(t):
    n = ni()
    a = nl()
    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 $x = -b/2a$.

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

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

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

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): $s_a =$

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

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

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

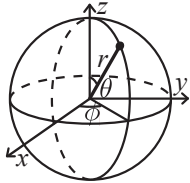Law of tangents: $\dfrac{a+b}{a-b}=\dfrac{\tan\dfrac{\alpha+\beta}{2}}{\tan\dfrac{\alpha-\beta}{2}}$

2.4.2 Quadrilaterals

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

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

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

$$x=r\sin\theta\cos\phi \qquad r=\sqrt{x^2+y^2+z^2}$$
$$y=r\sin\theta\sin\phi \qquad \theta=\mathrm{acos}(z/\sqrt{x^2+y^2+z^2})$$
$$z=r\cos\theta \qquad \phi=\mathrm{atan2}(y,x)$$

2.5 Derivatives/Integrals

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

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

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

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

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

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

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

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

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

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

2.7 Series

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

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

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

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

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

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) $\mu=\mathbb{E}(X)=\sum_x xp_X(x)$ and variance $\sigma^2=V(X)=\mathbb{E}(X^2)-(\mathbb{E}(X))^2=\sum_x(x-\mathbb{E}(X))^2p_X(x)$ 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,

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

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 $\mathrm{Bin}(n,p)$, $n=1,2,\ldots, 0\leq p\leq 1$.

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

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

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

First success distribution

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

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

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

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 $\mathrm{Po}(\lambda)$, $\lambda=t\kappa$.

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

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

2.8.2 Continuous distributions

Uniform distribution

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

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

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

Exponential distribution

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

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

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

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 $X_1, X_2, \ldots$ 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). $\lim_{k \to \infty} \mathbf{P}^k = \mathbf{1}\pi$.

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, 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 $t_i = 1 + \sum_{k \in \mathbf{G}} p_{ki}t_k$.

Chapter 3

Graph Algorithms

bfs:

```
# S is index, G is adjacancy list
# finds distance from S to all verticies in G
def bfs(S, G):
    q = [S]
    INF = 10**18
    dist = [INF]*len(G)
    dist[S] = 0
    while q:
        q2 = []
        for u in q:
            for v in G[u]:
                # early break here if only interesed in length
 of S -> T path.
                if dist[u] + 1 < dist[v]:
                    dist[v] = dist[u] + 1
                    q2.append(v)
        q = q2
    return dist
```

djikstra:

```
from heapq import heappop as pop, heappush as push
# adj: adj-list where edges are tuples (node_id, weight):
# (1) --2-- (0) --3-- (2) has the adj-list:
# adj = [[(1, 2), (2, 3)], [(0, 2)], [0, 3]]
def dijk(adj, S, T):
    N = len(adj)
    INF = 10**18
    dist = [INF]*N
    pq = []
    def add(i, dst):
        if dst < dist[i]:
            dist[i] = dst
            push(pq, (dst, i))
    add(S, 0)

    while pq:
        D, i = pop(pq)
        if i == T: return D
        if D != dist[i]: continue
        for j, w in adj[i]:
            add(j, D + w)

    return dist[T]
```

twoSat:

```
# used in sevenkingdoms, illumination
import sys
sys.setrecursionlimit(10**5)
class Sat:
    def __init__(self, no_vars):
        self.size = no_vars*2
        self.no_vars = no_vars
        self.adj = [[] for _ in range(self.size)]
        self.back = [[] for _ in range(self.size)]
    def add_imply(self, i, j):
        self.adj[i].append(j)
        self.back[j].append(i)
    def add_or(self, i, j):
        self.add_imply(i^1, j)
        self.add_imply(j^1, i)
    def add_xor(self, i, j):
        self.add_or(i, j)
        self.add_or(i^1, j^1)
    def add_eq(self, i, j):
        self.add_xor(i, j^1)

    def dfs1(self, i):
        if i in self.marked: return
        self.marked.add(i)
        for j in self.adj[i]:
            self.dfs1(j)
        self.stack.append(i)

    def dfs2(self, i):
        if i in self.marked: return
        self.marked.add(i)
        for j in self.back[i]:
            self.dfs2(j)
        self.comp[i] = self.no_c

    def is_sat(self):
        self.marked = set()
        self.stack = []
        for i in range(self.size):
            self.dfs1(i)
        self.marked = set()
        self.no_c = 0
        self.comp = [0]*self.size
        while self.stack:
            i = self.stack.pop()
            if i not in self.marked:
                self.no_c += 1
                self.dfs2(i)
        for i in range(self.no_vars):
            if self.comp[i*2] == self.comp[i*2+1]:
                return False
        return True

    # assumes is_sat.
    # If not xi is after xi in topological sort,
    # xi should be FALSE. It should be TRUE otherwise.
    # https://codeforces.com/blog/entry/16205
    def solution(self):
        V = []
        for i in range(self.no_vars):
            V.append(self.comp[i*2] > self.comp[i*2^1])
        return V

if __name__ == '__main__':
    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
   self.level[u] + 1 == self.level[v]] for u in range(self.sz
   )]
48         self.pos = [0]*self.sz
49         self.dead = set()
50     def max_flow(self, s, t):
51         flow = 0
52         while self.bfs(s, t):
53             self.setup_after_bfs()
54             while True:
55                 pushed = self.dfs(s, t, self.INF)
56                 if not pushed: break
57                 flow += pushed
58         return flow
```

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
   [2*i+1])
22
23     def _combine(self, a, b):
24         if a is None: return b
25         if b is None: return a
26         return self.func(a, b)
27
28     def query(self, lo, hi):
29         assert 0 <= lo <= hi < self.sz
30         return self.__query(1, lo, hi)
31
32     def __query(self, i, lo, hi):
33         l, r = self.L[i], self.R[i]
34         if r < lo or hi < l:
35             return None
36         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
   else k.stop)
76     def __getitem__(self, k):
77         if type(k) == slice:
78             k = self.__fixslice__(k)
79             return self.query(k.start, k.stop - 1)
80         elif type(k) == int:
81             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     # A[i] = v
15     def assign(self, i, v):
16         currV = self.query(i, i)
17         self.inc(i, v - currV)
18     # 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
    else k.stop)
40    def __setitem__(self, i, v):
41        self.assign(i, v)
42    def __getitem__(self, k):
43        if type(k) == slice:
44            k = self.__fixslice__(k)
45            return self.query(k.start, k.stop - 1)
46        elif type(k) == int:
47            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(
14                     self.data[i][j-1],
15                     self.data[i + (1<<(j-1))][j-1])
16
17     def query(self, a, b):
18         if a > b:
19             # some default value when query is empty
20             return 1
21         d = b - a + 1
22         k = int(math.log(d, 2))
23         return self.func(self.data[a][k], self.data[b-(1<<k)
    +1][k])
```

Uniion 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
    match L[i] and R[j]
2  # Description: Given a weighted bipartite graph, matches every
    node on
3  # the left with a node on the right such that no
4  # nodes are in two matchings and the sum of the edge weights is
    minimal. Takes
5  # cost[N][M], where cost[i][j] = cost for L[i] to be matched
    with R[j] and
6  # Returns: (min cost, match), where L[i] is matched with R[
    match[i]].
7  # Negate costs for max cost.
8  # Time: O(N^2M)
9  #
10 def hungarian(G):
11     INF = 10**18
12     if len(G) == 0:
13         return 0, []
14
15     n, m = len(G) + 1, len(G[0]) + 1
16     u, v, p = [0]*n, [0]*m, [0]*m
17     ans = [0]*(n-1)
18     for i in range(1, n):
19         p[0], j0 = i, 0
20         dist, pre = [INF]*m, [-1]*m
21         done = [False]*(m+1)
22         while True:
23             done[j0] = True
24             i0, j1, delta = p[j0], 0, INF
25             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]) ==
    False)
10         except:
11             return None #A is not independent!
12         if i != m:
13             A[i], A[m] = A[m], A[i]
14             b[i], b[m] = b[m], b[i]
15         for j in range(i+1, N):
16             sub = A[j][i]/A[i][i]
17             b[j] -= sub*b[i]
18             for k in range(N):
19                 A[j][k] -= sub*A[i][k]
20
21     for i in range(N-1, -1, -1):
22         for j in range(N-1, i, -1):
23             sub = A[i][j]/A[j][j]
24             b[i] -= sub*b[j]
25         b[i], A[i][i] = b[i]/A[i][i], A[i][i]/A[i][i]
26     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
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 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 distl(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 ditance 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 def sign(x):
101    if x < 0: return -1
102    return 1 if x > 0 else 0
103
104 def cross(u, v):
105    return u[0] * v[1] - u[1] * v[0]
106
107 # s1: (Point, Point)
108 # s2: (Point, Point)
109 # Point : (x, y)
```

```python
110  # returns true if intersecting s1 & s2 shares at least 1 point.
111  def is_segment_intersection(s1, s2):
112      u = vec(*s1)
113      v = vec(*s2)
114      p1, p2 = s1
115      q1, q2 = s2
116      d1 = cross(u, vec(p1, q1))
117      d2 = cross(u, vec(p1, q2))
118      d3 = cross(v, vec(q1, p1))
119      d4 = cross(v, vec(q1, p2))
120      if d1 * d2 * d3 * d4 == 0:
121          return True
122      return sign(d1) != sign(d2) and sign(d3) != sign(d4)
```

Chapter 7

Number theory

Primes:

```python
1   large_primes = [
2   5915587277,
3   1500450271,
4   3267000013,
5   5754853343,
6   4093082899,
7   9576890767,
8   3628273133,
9   2860486313,
10  5463458053,
11  3367900313,
12  1000000000000000061,
13  10**16 + 61,
14  10**17 + 3
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 isPrime(N):
28      if N < 2: return False
29      if N%2 == 0: return N == 2
30      mx = min(int(N**.5) + 2, N)
31      for i in range(3, mx, 2):
```

```python
32          if N % i == 0: return False
33      return True
34
35  def genPrimesFrom(N):
36      while True:
37          if isPrime(N):
38              yield N
39          N += 1
40
41  def getPrimesFrom(N, cnt):
42      itr = genPrimesFrom(N)
43      return [next(itr) for _ in range(cnt)]
```

Some useful functions:

```python
1   import math
2
3   # Evaluates to n! / (k! * (n - k)!) when k <= n and evaluates
        to zero when k > n.
4   # math.comb(n, k) #introduced in python3.8
5
6   # math.gcd(a, b)
7   def gcd(a, b):
8       return b if a%b == 0 else gcd(b, a%b)
9
10  # returns b where (a*b)%MOD == 1
11  def inv(a, MOD):
12      return pow(a, -1, MOD)
13
14  # returns g = gcd(a, b), x0, y0,
15  # where g = x0*a + y0*b
16  def xgcd(a, b):
17      x0, x1, y0, y1 = 1, 0, 0, 1
18      while b != 0:
19          q, a, b = (a // b, b, a % b)
20          x0, x1 = (x1, x0 - q * x1)
21          y0, y1 = (y1, y0 - q * y1)
22      return (a, x0, y0)
23
24  def crt(la, ln):
25      assert len(la) == len(ln)
26      for i in range(len(la)):
27          assert 0 <= la[i] < ln[i]
28      prod = 1
29      for n in ln:
30          assert gcd(prod, n) == 1
31          prod *= n
32      lN = []
33      for n in ln:
34          lN.append(prod//n)
35      x = 0
36      for i, a in enumerate(la):
37          print(lN[i], ln[i])
38          _, Mi, mi = xgcd(lN[i], ln[i])
39          x += a*Mi*lN[i]
40      return x % prod
41
42  # finds x^e mod m
43  # Or just pow(x, e, m)
44  def modpow(x, m, e):
45      res = 1
46      while e:
47          if e%2 == 1:
48              res = (res*x) % m
49          x = (x*x) % m
50          e = e//2
51      return res
```

```python
52
53  # Divides a list of digits with an int.
54  # A lot faster than using bigint-division.
55  def div(L, d):
56      r = [0]*(len(L) + 1)
57      q = [0]*len(L)
58      for i in range(len(L)):
59          x = int(L[i]) + r[i]*10
60          q[i] = x//d
61          r[i+1] = x-q[i]*d
62      s = []
63      for i in range(len(L) - 1, 0, -1):
64          s.append(q[i]%10)
65          q[i-1] += q[i]//10
66
67      while q[0]:
68          s.append(q[0]%10)
69          q[0] = q[0]//10
70      s = s[::-1]
71      i = 0
72      while s[i] == 0:
73          i += 1
74      return s[i:]
75
76  # Multiplies a list of digits with an int.
77  # A lot faster than using bigint-multiplication.
78  def mul(L, d):
79      r = [d*x for x in L]
80      s = []
81      for i in range(len(r) - 1, 0, -1):
82          s.append(r[i]%10)
83          r[i-1] += r[i]//10
84      while r[0]:
85          s.append(r[0]%10)
86          r[0] = r[0]//10
87      return s[::-1]
```