Contents

1 Math

1.1 Cramer's Rule

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

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.

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

1.3 Triangles

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

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

Inradius: $r=\dfrac{A}{s}$

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

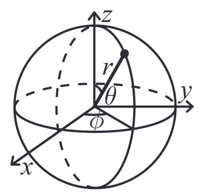
1.4 Quadrilaterals

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 $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

1.5 Spherical coordinates



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

1.6 Sums & Combinatorics

$\sum_{k=0}^{n} k = n(n+1)/2$
$\sum_{k=0}^{n} k^2 = n(n+1)(2n+1)/6$
$\sum_{k=0}^{n} k^3 = n^2(n+1)^2/4$
$\sum_{k=0}^{n} k^4 = (6n^5+15n^4+10n^3-n)/30$
$\sum_{k=0}^{n} k^5 = (2n^6+6n^5+5n^4-n^2)/12$
$\sum_{k=0}^{n} x^k = (x^{n+1}-1)/(x-1)$
$\sum_{k=0}^{n} kx^k = (x-(n+1)x^{n+1}+nx^{n+2})/(x-1)^2$
$\binom{n}{k} = \frac{n!}{(n-k)!k!}$
$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$
$\binom{n}{k} = \frac{n}{n-k}\binom{n-1}{k}$
$\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}$
$\binom{n+1}{k} = \frac{n+1}{n-k+1}\binom{n}{k}$
$\binom{n}{k+1} = \frac{n-k}{k+1}\binom{n}{k}$
$\sum_{k=1}^{n} k\binom{n}{k} = n2^{n-1}$
$\sum_{k=1}^{n} k^2\binom{n}{k} = (n+n^2)2^{n-2}$
$\binom{m+n}{r} = \sum_{k=0}^{r}\binom{m}{k}\binom{n}{r-k}$
$\binom{n}{k} = \prod_{i=1}^{k}\frac{n-k+i}{i}$

Hockey stick Formulas:

$\sum_{i=k}^{n}\binom{i}{k} = \binom{n+1}{k+1}$
$\binom{n+1}{n-k} = \sum_{j=0}^{n-k}\binom{j+k}{k} = \sum_{j=0}^{n-k}\binom{j+k}{j}$

Taylor Series:

$f(x) = \sum_{n=0}^{\infty}\frac{f^{(n)}(a)}{n!}(x-a)^n$

1.7 Burnside's Lemma

Given a group G of symmetries and a set X, the number of elements of X *up to symmetry* equals

$$\frac{1}{|G|}\sum_{g\in G}|X^g|,$$

where X^g are the elements fixed by g ($g.x=x$).
If $f(n)$ counts "configurations" (of some sort) of length n, we can ignore rotational symmetry using $G=\mathbb{Z}_n$ to get

$$g(n) = \frac{1}{n}\sum_{k=0}^{n-1} f(\gcd(n,k)) = \frac{1}{n}\sum_{k|n} f(k)\phi(n/k).$$

1.8 Distributions

1.8.1 Binomial distribution

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

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

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

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

1.8.2 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 $\mathrm{Fs}(p)$, $0\le p\le 1$.

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

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

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

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

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

1.8.4 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

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

1.9 Primes

Lucas' Theorem: For non-negative integers m and n and a prime p,

$$\binom{m}{n} \equiv \prod_{i=0}^{k}\binom{m_i}{n_i} \pmod p,$$

where

$$m = m_k p^k + m_{k-1}p^{k-1} + \cdots + m_1 p + m_0$$

is the base p representation of m, and similarly for n.
Prime Number Theorem: $\pi(x) \approx \frac{x}{\ln(x)}$ is the # of primes $\leq x$.

n	$\pi(10^n)$		
1	4	13	346,065,536,839
2	25	14	3,204,941,750,802
3	168	15	29,844,570,422,669
4	1,229	16	279,238,341,033,925
5	9,592	17	2,623,557,157,654,233
6	78,498	18	24,739,954,287,740,860
7	664,579	19	234,057,667,276,344,607
8	5,761,455	20	2,220,819,602,560,918,840
9	50,847,534	21	21,127,269,486,018,731,928
10	455,052,511	22	201,467,286,689,315,906,290
11	4,118,054,813	23	1,925,320,391,606,803,968,923
12	37,607,912,018	24	18,435,599,767,349,200,867,866
		25	176,846,309,399,143,769,411,680

1.10 Stirling Numbers of the second kind

Number of ways to partition a set of n numbers into k non-empty subsets.

$$\left\{{n \atop k}\right\} = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{(k-j)}\binom{k}{j}j^n$$

$$\left\{{0 \atop 0}\right\} = 1, \qquad \left\{{n \atop 0}\right\} = \left\{{0 \atop n}\right\} = 1$$

$$\left\{{n+1 \atop k}\right\} = k\left\{{n \atop k}\right\} + \left\{{n \atop k-1}\right\}$$

1.11 Derangements

Permutations of a set such that none of the elements appear in their original position.
$D(n) = (n-1)(D(n-1) + D(n-2))$
$= nD(n-1) + (-1)^n = \left\lfloor \frac{n!}{e} \right\rceil$

1.12 Partition function

Number of ways of writing n as a sum of positive integers, disregarding the order of the summands.

$$p(0) = 1,\ p(n) = \sum_{k \in \mathbb{Z}\setminus\{0\}}(-1)^{k+1}p(n - k(3k-1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	~2e5	~2e8

1.13 Bell numbers

Total number of partitions of n distinct elements. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \dots$. For p prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod p$$

1.14 Catalan numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1,\ C_{n+1} = \frac{2(2n+1)}{n+2}C_n,\ C_{n+1} = \sum C_i C_{n-i}$$

$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \dots$

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

1.15 Erdős–Gallai theorem

A simple graph with node degrees $d_1 \geq \cdots \geq d_n$ exists iff $d_1 + \cdots + d_n$ is even, and for every $k = 1 \dots n$:

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n}\min(d_i, k)$$

1.16 Misc

Bayes' Theorem: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$
Planar Graph Formula: $v - e + f = 2$

2 CPP Header, Compilation

```cpp
#include <bits/stdc++.h>
using namespace std;

// KACTL Directives (For Geo)
#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

int main() {
        cin.tie(0)->sync_with_stdio(0);
        cin.exceptions(cin.failbit);
}

// Compile:
// g++ -Wall -Wconversion -Wfatal-errors -g
↪  -std=c++17
```

3 String Algorithms

3.1 Aho Corasick

```java
public class AhoCorasick {
    final int ALPHABET_SIZE = 26;
    final int MAX_STATES = 200_000;

    int[][] transitions = new int[MAX_STATES][ALPHABET_SIZE];
    int[] sufflink = new int[MAX_STATES];
    int[] escape = new int[MAX_STATES];
    int states = 1;

    public int addString(String s) {
        int v = 0;
        for (char c : s.toCharArray()) {
            c -= 'a';
            if (transitions[v][c] == 0) {
                transitions[v][c] = states++;
            }
            v = transitions[v][c];
        }
        escape[v] = v;
        return v;
    }

    public void buildLinks() {
        int[] q = new int[MAX_STATES];
        for (int s = 0, t = 1; s < t; ) {
            int v = q[s++];
            int u = sufflink[v];
            if (escape[v] == 0) {
                escape[v] = escape[u];
            }
            for (int c = 0; c < ALPHABET_SIZE; c++) {
                if (transitions[v][c] != 0) {
                    q[t++] = transitions[v][c];
                    sufflink[transitions[v][c]] = v != 0 ? transitions[u][c] : 0;
                } else {
                    transitions[v][c] = transitions[u][c];
                }
            }
        }
    }

    // Usage example
    public static void main(String[] args) {
        AhoCorasick ahoCorasick = new AhoCorasick();
        ahoCorasick.addString("a");
        ahoCorasick.addString("aa");
        ahoCorasick.addString("abaaa");
        ahoCorasick.buildLinks();

        int[][] t = ahoCorasick.transitions;
        int[] e = ahoCorasick.escape;

        String s = "abaa";

        int state = 0;
        for (int i = 0; i < s.length(); i++) {
            state = t[state][s.charAt(i) - 'a'];
            if (e[state] != 0)
                System.out.println(i);
        }
    }
}
```

3.2 Prefix, Z, and Manacher Functions

```java
public class Strings {
    private static void prefixFunc(String s, int[] pi) {
        int n = s.length();
        for (int i = 1; i < n; i++) {
            int j = pi[i - 1];
            while (j > 0 && s.charAt(i) != s.charAt(j))
                j = pi[j - 1];
            if (s.charAt(i) == s.charAt(j))
                j++;
            pi[i] = j;
        }
    }

    private static void zFunc(String s, int[] z) {
        int n = s.length();
        for (int i = 1, l = 0, r = 0; i < n; ++i) {
            if (i <= r)
                z[i] = Math.min(r - i + 1, z[i - l]);
            while (i + z[i] < n && s.charAt(z[i]) == s.charAt(i + z[i]))
                ++z[i];
            if (i + z[i] - 1 > r) {
                l = i;
                r = i + z[i] - 1;
            }
        }
    }

    // p[0][i] = half length of longest even palindrome around pos i, p[1][i] =
    ↪ longest odd (half rounded down).
    private static int[][] manacher(String s) {
        int n = s.length();
        int[][] p = new int[2][n + 1];
        for (int z = 0; z < 2; z++) {
            for (int i = 0, l = 0, r = 0; i < n; i++) {
                int t = r - i + 1 - z;
                if (i < r) {
                    p[z][i] = Math.min(t, p[z][l + t]);
                }
                int L = i - p[z][i], R = i + p[z][i] + 1 - z;
                while (L >= 1 && R + 1 < n && s.charAt(L - 1) == s.charAt(R + 1)) {
                    p[z][i]++;
                    L--;
                    R++;
                }
                if (R > r) {
                    l = L;
                    r = R;
                }
```

```
            }
        }
        return p;
    }

    // Booth's
    private static String leastRotation(String s) {
        s += s;
        int[] f = new int[s.length()];
        int k = 0;
        for (int j = 1; j < s.length(); j++) {
            int i = f[j - k - 1];
            while (i != -1 && s.charAt(j) != s.charAt(k + i + 1)) {
                if (s.charAt(j) < s.charAt(k + i + 1)) k = j - i - 1;
                i = f[i];
            }

            if (s.charAt(j) != s.charAt(k + i + 1)) {
                if (s.charAt(j) < s.charAt(k)) k = j;
                f[j - k] = -1;
            } else f[j - k] = i + 1;
        }
        return s.substring(k, k + s.length() / 2);
    }
}
```

3.3 Suffix Array

```
import java.util.stream.IntStream;

public class SuffixArray {
    static boolean leq(int a1, int a2, int b1, int b2) {
        return a1 < b1 || a1 == b1 && a2 <= b2;
    }

    static boolean leq(int a1, int a2, int a3, int b1, int b2, int b3) {
        return a1 < b1 || a1 == b1 && leq(a2, a3, b2, b3);
    }

    // stably sort a[0..n-1] to b[0..n-1] with keys in 0..K from r
    static void radixPass(int[] a, int[] b, int[] r, int offset, int n, int K) {
        int[] cnt = new int[K + 1];
        for (int i = 0; i < n; i++) ++cnt[r[a[i] + offset]];
        for (int i = 1; i < cnt.length; i++) cnt[i] += cnt[i - 1];
        for (int i = n - 1; i >= 0; i--) b[--cnt[r[a[i] + offset]]] = a[i];
    }

    // find the suffix array SA of T[0..n-1] in {1..K}^n
    // require T[n]=T[n+1]=T[n+2]=0, n>=2
    private static void suffixArray(int[] T, int[] SA, int n, int K) {
        int n0 = (n + 2) / 3;
        int n1 = (n + 1) / 3;
        int n2 = n / 3;
        int n02 = n0 + n2;

        //******* Step 0: Construct sample ********
        // generate positions of mod 1 and mod 2 suffixes
        // the "+(n0-n1)" adds a dummy mod 1 suffix if n%3 == 1
        int[] R = new int[n02 + 3];
        for (int i = 0, j = 0; i < n + (n0 - n1); i++)
            if (i % 3 != 0)
                R[j++] = i;

        //******* Step 1: Sort sample suffixes ********
        // lsb radix sort the mod 1 and mod 2 triples
        int[] SA12 = new int[n02 + 3];
        radixPass(R, SA12, T, 2, n02, K);
        radixPass(SA12, R, T, 1, n02, K);
        radixPass(R, SA12, T, 0, n02, K);

        // find lexicographic names of triples and
        // write them to correct places in R
        int name = 0;
        for (int i = 0; i < n02; i++) {
            if (i == 0 || T[SA12[i]] != T[SA12[i - 1]] || T[SA12[i] + 1] != T[SA12[i - 1] + 1]
                    || T[SA12[i] + 2] != T[SA12[i - 1] + 2]) {
                ++name;
            }
            R[SA12[i] / 3 + (SA12[i] % 3 == 1 ? 0 : n0)] = name;
        }

        if (name < n02) {
            // recurse if names are not yet unique
            suffixArray(R, SA12, n02, name);
            // store unique names in R using the suffix array
            for (int i = 0; i < n02; i++) R[SA12[i]] = i + 1;
        } else {
            // generate the suffix array of R directly
            for (int i = 0; i < n02; i++) SA12[R[i] - 1] = i;
        }

        //******* Step 2: Sort nonsample suffixes ********
        // stably sort the mod 0 suffixes from SA12 by their first character
        int[] R0 = new int[n0];
        for (int i = 0, j = 0; i < n02; i++)
            if (SA12[i] < n0)
                R0[j++] = 3 * SA12[i];
        int[] SA0 = new int[n0];
        radixPass(R0, SA0, T, 0, n0, K);

        //******* Step 3: Merge ********
        // merge sorted SA0 suffixes and sorted SA12 suffixes
        for (int p = 0, t = n0 - n1, k = 0; k < n; k++) {
            int i = SA12[t] < n0 ? SA12[t] * 3 + 1 : (SA12[t] - n0) * 3 + 2; // pos of current offset 12 suffix
            int j = SA0[p]; // pos of current offset 0 suffix
            if (SA12[t] < n0 ? // different compares for mod 1 and mod 2 suffixes
                    leq(T[i], R[SA12[t] + n0], T[j], R[j / 3])
                    : leq(T[i], T[i + 1], R[SA12[t] - n0 + 1], T[j], T[j + 1],
                    R[j / 3 + n0])) { // suffix from SA12 is smaller
                SA[k] = i;
                if (++t == n02) // done --- only SA0 suffixes left
                    for (k++; p < n0; p++, k++) SA[k] = SA0[p];
            } else { // suffix from SA0 is smaller
                SA[k] = j;
                if (++p == n0) // done --- only SA12 suffixes left
```

```java
                for (k++; t < n02; t++, k++) SA[k] = SA12[t] < n0 ? SA12[t] * 3 + 1 : (SA12[t] - n0) * 3 + 2;
            }
        }
    }

    public static int[] suffixArray(CharSequence s) {
        int n = s.length();
        if (n <= 1)
            return new int[n];
        int[] S = IntStream.range(0, n + 3).map(i -> i < n ? s.charAt(i) : 0).toArray();
        int[] sa = new int[n];
        suffixArray(S, sa, n, 255);
        return sa;
    }

    // longest common prefixes array in O(n)
    public static int[] lcp(int[] sa, CharSequence s) {
        int n = sa.length;
        int[] rank = new int[n];
        for (int i = 0; i < n; i++) rank[sa[i]] = i;
        int[] lcp = new int[n - 1];
        for (int i = 0, h = 0; i < n; i++) {
            if (rank[i] < n - 1) {
                for (int j = sa[rank[i] + 1]; Math.max(i, j) + h < s.length() && s.charAt(i + h) == s.charAt(j + h);
                     ++h)
                    ;
                lcp[rank[i]] = h;
                if (h > 0)
                    --h;
            }
        }
        return lcp;
    }
}
```

4 Data structures

4.1 Disjoint Set

```java
public class DisjointSet {
    int[] parent;
    int[] rank;
    int components;

    public DisjointSet(int size) {
        parent = new int[size];
        rank = new int[size];
        components = size;
        for (int i = 0; i < size; i++) {
            parent[i] = i;
        }
    }

    public int find(int i) {
        return parent[i] == i ? i : (parent[i] = find(parent[i]));
    }

    public void union(int x, int y) {
        int px = find(x);
        int py = find(y);
        if (px == py) {
            return;
        }
        components--;
        if (rank[px] < rank[py]) {
            parent[px] = py;
        } else if (rank[px] > rank[py]) {
            parent[py] = px;
        } else {
            parent[px] = py;
            rank[px]++;
        }
    }
}
```

4.2 Segment Tree

```java
public class SegmentTree {
    public static int get(int[] t, int i) {
        return t[i + t.length / 2];
    }

    public static void add(int[] t, int i, int value) {
        i += t.length / 2;
        t[i] += value;
        for (; i > 1; i >>= 1) t[i >> 1] = Math.max(t[i], t[i ^ 1]);
    }

    public static int max(int[] t, int a, int b) {
        int res = Integer.MIN_VALUE;
        for (a += t.length / 2, b += t.length / 2; a <= b; a = (a + 1) >> 1, b = (b - 1) >> 1) {
            if ((a & 1) != 0)
                res = Math.max(res, t[a]);
            if ((b & 1) == 0)
                res = Math.max(res, t[b]);
        }
        return res;
    }

    // Usage example
    public static void main(String[] args) {
        int n = 10;
        int[] t = new int[n + n];
        add(t, 0, 1);
        add(t, 9, 2);
        System.out.println(max(t, 0, 9));
    }
}
```

```
typedef struct MyDS {
    MyDS() {}
    MyDS operator+(const MyDS& right) {}
} SegTreeType;

struct SegmentTree {
    vector<SegTreeType> tree;
    int arrSize;
    SegTreeType ST_ZERO = SegTreeType();
    SegmentTree(vector<SegTreeType>& arr, int size):
        tree(size*2, ST_ZERO), arrSize(size) {
        // Builds segment tree
        for (int i = 0; i < arrSize; i++)
            tree[arrSize + i] = arr[i];
        for (int i = arrSize - 1; i > 0; i--)
            tree[i] = tree[i << 1] + tree[i << 1 | 1];
    }
    // Update array position idx with val.
    void update(int idx, SegTreeType val) {
        for (tree[idx += arrSize] = val; idx /= 2;)
            tree[idx] = tree[idx << 1] + tree[idx << 1 | 1];
    }
    // Query the tree to find the answer for a range of [l, r] (inclusive)
    SegTreeType query(int l, int r) {
        SegTreeType lAns = ST_ZERO, rAns = ST_ZERO;
        for (l += arrSize, r += arrSize + 1; l < r; l/=2, r/=2) {
            if (l % 2) lAns = lAns + tree[l++];
            if (r % 2) rAns = tree[--r] + rAns;
        }
        return lAns + rAns;
    }
};
```

4.3 Lazy Segment Tree

```
/*
 * Description: Segment tree with ability to add or set values of large intervals,
↪  and compute max of intervals.
 * Time: O(\log N).
 * Usage: Node* tr = new Node(v, 0, sz(v));
 */
const int inf = 1e9;
struct Node {
        Node *l = 0, *r = 0;
        int lo, hi, mset = inf, madd = 0, val = -inf;
        Node(int lo,int hi):lo(lo),hi(hi){} // Large interval of -inf
        Node(vector<int>& v, int lo, int hi) : lo(lo), hi(hi) {
                if (lo + 1 < hi) {
                        int mid = lo + (hi - lo)/2;
                        l = new Node(v, lo, mid); r = new Node(v, mid, hi);
                        val = max(l->val, r->val);
                }
                else val = v[lo];
        }
        int query(int L, int R) {
                if (R <= lo || hi <= L) return -inf;
                if (L <= lo && hi <= R) return val;
                push();
                return max(l->query(L, R), r->query(L, R));
        }
        void set(int L, int R, int x) {
                if (R <= lo || hi <= L) return;
                if (L <= lo && hi <= R) mset = val = x, madd = 0;
                else {
                        push(), l->set(L, R, x), r->set(L, R, x);
                        val = max(l->val, r->val);
                }
        }
        void add(int L, int R, int x) {
                if (R <= lo || hi <= L) return;
                if (L <= lo && hi <= R) {
                        if (mset != inf) mset += x;
                        else madd += x;
                        val += x;
                }
                else {
                        push(), l->add(L, R, x), r->add(L, R, x);
                        val = max(l->val, r->val);
                }
        }
        void push() {
                if (!l) {
                        int mid = lo + (hi - lo)/2;
                        l = new Node(lo, mid); r = new Node(mid, hi);
                }
                if (mset != inf)
                        l->set(lo,hi,mset), r->set(lo,hi,mset), mset = inf;
                else if (madd)
                        l->add(lo,hi,madd), r->add(lo,hi,madd), madd = 0;
        }
};
```

4.4 Fenwick Tree

```
/* FenwickTree(int numElems): Fenwick/Binary Indexed Tree. 0-indexed inputs to
↪  functions. */
struct FenwickTree {
    vector<ll> data;
    int size;
    FenwickTree(int n) : data(n+1, 0), size(n+1) {}
    ll getSum(int currIdx) const {
        ll sum = 0; ++currIdx;
        while (currIdx > 0) {
            sum += data[currIdx];
            currIdx -= currIdx & (-currIdx);
        }
        return sum;
    }
    void update(int currIdx, ll delta) {
        ++currIdx;
        while (currIdx < size) {
            data[currIdx] += delta;
            currIdx += currIdx & (-currIdx);
```

```
        }
    }
};
```

4.5 Order Statistic Tree

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

// Use pair<int,int> if dup elems can exist -- first val, second index.
typedef tree<int, null_type, less<int>, rb_tree_tag,
                         tree_order_statistics_node_update>
        ordered_set;
ordered_set X;
X.insert(1);
X.insert(2);
X.insert(4);

cout<<*X.find_by_order(1)<<endl; // 2
cout<<*X.find_by_order(2)<<endl; // 4
cout<<(end(X)==X.find_by_order(4))<<endl; // true

cout<<X.order_of_key(-5)<<endl;  // 0
cout<<X.order_of_key(1)<<endl;   // 0
cout<<X.order_of_key(3)<<endl;   // 2
cout<<X.order_of_key(4)<<endl;   // 2
cout<<X.order_of_key(400)<<endl; // 2
```

5 Graph Algorithms

5.1 Dinics Max Flow

```
public class DinicsMaxFlow {
    private static class Node {
        int level;
        List<Edge> edges;
        int id;

        public Node(int i) {
            id = i;
            level = -1;
            edges = new ArrayList<>();
        }
    }

    private static class Edge {
        int capacity;
        int used;
        // Flows from a to b, if directed
        Node a;
        Node b;

        public Edge(int c, Node a, Node b) {
            capacity = c;
            used = 0;
            this.a = a;
            this.b = b;
        }

        public Node getOther(Node t) {
            return t == a ? b : a;
        }

        public boolean isNodeSource(Node t) {
            return t == a;
        }
    }

    private static int Dinics(Node source, Node sink) {
        int total = 0;
        while (createLevels(source, sink)) {
            int flow = getFlow(Integer.MAX_VALUE, source, sink);
            while (flow != 0) {
                total += flow;
                flow = getFlow(Integer.MAX_VALUE, source, sink);
            }
        }
        return total;
    }

    private static int getFlow(int max, Node source, Node sink) {
        if (source == sink) {
            return max;
        } else {
            for (Edge e : source.edges) {
                Node other = e.getOther(source);
                int flow;
                if (other.level - source.level == 1) {
                    if (canFlow(source, e)) {
                        flow = getFlow(Math.min(max, e.capacity - e.used), other,
                           sink);
                        if (flow != 0) {
                            e.used += flow;
                            return flow;
                        }
                    } else if (canUnFlow(source, e)) {
                        flow = getFlow(Math.min(max, e.used), other, sink);
                        if (flow != 0) {
                            e.used -= flow;
                            return flow;
                        }
                    }
                }
            }
            return 0;
        }
    }

    private static boolean createLevels(Node source, Node sink) {
        HashSet<Node> visited = new HashSet<>();
        visited.add(source);
        source.level = 0;
        Queue<Node> queue = new ArrayDeque<>();
```

```java
        queue.add(source);
        while (!queue.isEmpty()) {
            Node s = queue.remove();
            for (Edge e : s.edges) {
                Node o = e.getOther(s);
                if (!visited.contains(o) && (canFlow(s, e) || canUnFlow(s, e))) {
                    o.level = s.level + 1;
                    visited.add(o);
                    queue.add(o);
                }
            }
        }
        return visited.contains(sink);
    }

    private static boolean canFlow(Node s, Edge e) {
        return e.isNodeSource(s) && e.capacity > e.used;
    }

    private static boolean canUnFlow(Node s, Edge e) {
        return !e.isNodeSource(s) && e.used > 0;
    }
}
```

5.2 Min Cost Flow

```java
import java.util.*;
import java.util.stream.Stream;

// https://cp-algorithms.com/graph/min_cost_flow.html in O(E * V + min(E * logV *
↪  FLOW, V^2 * FLOW))
// negative-cost edges are allowed
// negative-cost cycles are not allowed
public class MinCostFlow {
    List<Edge>[] graph;

    public MinCostFlow(int nodes) {
        graph = Stream.generate(ArrayList::new).limit(nodes).toArray(List[]::new);
    }

    class Edge {
        int to, rev, cap, f, cost;

        Edge(int to, int rev, int cap, int cost) {
            this.to = to;
            this.rev = rev;
            this.cap = cap;
            this.cost = cost;
        }
    }

    public void addEdge(int s, int t, int cap, int cost) {
        graph[s].add(new Edge(t, graph[t].size(), cap, cost));
        graph[t].add(new Edge(s, graph[s].size() - 1, 0, -cost));
    }

    void bellmanFord(int s, int[] dist) {
        int n = graph.length;
        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[s] = 0;
        boolean[] inqueue = new boolean[n];
        int[] q = new int[n];
        int qt = 0;
        q[qt++] = s;
        for (int qh = 0; qh != qt; qh++) {
            int u = q[qh % n];
            inqueue[u] = false;
            for (int i = 0; i < graph[u].size(); i++) {
                Edge e = graph[u].get(i);
                if (e.cap <= e.f)
                    continue;
                int v = e.to;
                int ndist = dist[u] + e.cost;
                if (dist[v] > ndist) {
                    dist[v] = ndist;
                    if (!inqueue[v]) {
                        inqueue[v] = true;
                        q[qt++ % n] = v;
                    }
                }
            }
        }
    }

    void dijkstra(
            int s, int t, int[] pot, int[] dist, boolean[] finished, int[] curflow,
            ↪  int[] prevnode, int[] prevedge) {
        PriorityQueue<Long> q = new PriorityQueue<>();
        q.add((long) s);
        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[s] = 0;
        Arrays.fill(finished, false);
        curflow[s] = Integer.MAX_VALUE;
        while (!finished[t] && !q.isEmpty()) {
            long cur = q.remove();
            int u = (int) (cur & 0xFFFF_FFFFL);
            int priou = (int) (cur >>> 32);
            if (priou != dist[u])
                continue;
            finished[u] = true;
            for (int i = 0; i < graph[u].size(); i++) {
                Edge e = graph[u].get(i);
                if (e.f >= e.cap)
                    continue;
                int v = e.to;
                int nprio = dist[u] + e.cost + pot[u] - pot[v];
                if (dist[v] > nprio) {
                    dist[v] = nprio;
                    q.add(((long) nprio << 32) + v);
                    prevnode[v] = u;
                    prevedge[v] = i;
                    curflow[v] = Math.min(curflow[u], e.cap - e.f);
                }
            }
        }
    }

```

```java
    void dijkstra2(
            int s, int t, int[] pot, int[] dist, boolean[] finished, int[] curflow, int[] prevnode, int[] prevedge) {
        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[s] = 0;
        int n = graph.length;
        Arrays.fill(finished, false);
        curflow[s] = Integer.MAX_VALUE;
        for (int i = 0; i < n && !finished[t]; i++) {
            int u = -1;
            for (int j = 0; j < n; j++)
                if (!finished[j] && (u == -1 || dist[u] > dist[j]))
                    u = j;
            if (dist[u] == Integer.MAX_VALUE)
                break;
            finished[u] = true;
            for (int k = 0; k < graph[u].size(); k++) {
                Edge e = graph[u].get(k);
                if (e.f >= e.cap)
                    continue;
                int v = e.to;
                int nprio = dist[u] + e.cost + pot[u] - pot[v];
                if (dist[v] > nprio) {
                    dist[v] = nprio;
                    prevnode[v] = u;
                    prevedge[v] = k;
                    curflow[v] = Math.min(curflow[u], e.cap - e.f);
                }
            }
        }
    }

    public int[] minCostFlow(int s, int t, int maxf) {
        int n = graph.length;
        int[] pot = new int[n];
        int[] dist = new int[n];
        boolean[] finished = new boolean[n];
        int[] curflow = new int[n];
        int[] prevedge = new int[n];
        int[] prevnode = new int[n];

        bellmanFord(s, pot); // this can be commented out if edges costs are non-negative
        int flow = 0;
        int flowCost = 0;
        while (flow < maxf) {
            dijkstra(s, t, pot, dist, finished, curflow, prevnode, prevedge); // E*logV
            //          dijkstra2(s, t, pot, dist, finished, curflow, prevnode, prevedge); // V^2
            if (dist[t] == Integer.MAX_VALUE)
                break;
            for (int i = 0; i < n; i++)
                if (finished[i])
                    pot[i] += dist[i] - dist[t];
            int df = Math.min(curflow[t], maxf - flow);
            flow += df;
            for (int v = t; v != s; v = prevnode[v]) {
                Edge e = graph[prevnode[v]].get(prevedge[v]);
                e.f += df;
                graph[v].get(e.rev).f -= df;
                flowCost += df * e.cost;
            }
        }
        return new int[]{flow, flowCost};
    }

    // Usage example
    public static void main(String[] args) {
        MinCostFlow mcf = new MinCostFlow(3);
        mcf.addEdge(0, 1, 3, 1);
        mcf.addEdge(0, 2, 2, 1);
        mcf.addEdge(1, 2, 2, 1);
        int[] res = mcf.minCostFlow(0, 2, Integer.MAX_VALUE);
        int flow = res[0];
        int flowCost = res[1];
        System.out.println(4 == flow);
        System.out.println(6 == flowCost);
    }
}
```

5.3 Euler Walk

```java
// gr is adjacency List, with pair (destination, global edge index), output is reverse path
private static List<Integer> eulerWalk(List<List<int[]>> gr, int nEdges, int src) {
    int n = gr.size();
    int[] D = new int[n];
    int[] its = new int[n];
    int[] eu = new int[n];
    List<Integer> ret = new ArrayList<>();
    Stack<Integer> s = new Stack<>();
    s.push(src);
    D[src]++; // to allow Euler paths, not just cycles
    while (!s.empty()) {
        int x = s.peek();
        int end = gr.get(x).size();
        if (its[x] == end) {
            ret.add(x);
            s.pop();
            continue;
        }
        int y = gr.get(x).get(its[x])[0];
        int e = gr.get(x).get(its[x])[1];
        its[x]++;
        if (eu[e] == 0) {
            D[x]--;
            D[y]++;
            eu[e] = 1;
            s.push(y);
        }
    }
    for (int x : D) {
        if (x < 0 || ret.size() != nEdges + 1) return null;
    }
    return ret;
```

```
}
```

5.4 Bellman Ford

```
public class BellmanFord {
    static final int INF = Integer.MAX_VALUE / 2;

    public static class Edge {
        final int v, cost;

        public Edge(int v, int cost) {
            this.v = v;
            this.cost = cost;
        }
    }

    public static boolean bellmanFord(List<Edge>[] graph, int s, int[] dist, int[]
    ↪  pred) {
        Arrays.fill(pred, -1);
        Arrays.fill(dist, INF);
        dist[s] = 0;
        int n = graph.length;
        for (int step = 0; step < n; step++) {
            boolean updated = false;
            for (int u = 0; u < n; u++) {
                if (dist[u] == INF)
                    continue;
                for (Edge e : graph[u]) {
                    if (dist[e.v] > dist[u] + e.cost) {
                        dist[e.v] = dist[u] + e.cost;
                        dist[e.v] = Math.max(dist[e.v], -INF);
                        pred[e.v] = u;
                        updated = true;
                    }
                }
            }
            if (!updated)
                return true;
        }
        // a negative cycle exists
        return false;
    }

    public static int[] findNegativeCycle(List<Edge>[] graph) {
        int n = graph.length;
        int[] pred = new int[n];
        Arrays.fill(pred, -1);
        int[] dist = new int[n];
        int last = -1;
        for (int step = 0; step < n; step++) {
            last = -1;
            for (int u = 0; u < n; u++) {
                if (dist[u] == INF)
                    continue;
                for (Edge e : graph[u]) {
                    if (dist[e.v] > dist[u] + e.cost) {
                        dist[e.v] = dist[u] + e.cost;
                        dist[e.v] = Math.max(dist[e.v], -INF);
                        pred[e.v] = u;
                        last = e.v;
                    }
                }
            }
            if (last == -1)
                return null;
        }
        for (int i = 0; i < n; i++) {
            last = pred[last];
        }
        int[] p = new int[n];
        int cnt = 0;
        for (int u = last; u != last || cnt == 0; u = pred[u]) {
            p[cnt++] = u;
        }
        int[] cycle = new int[cnt];
        for (int i = 0; i < cycle.length; i++) {
            cycle[i] = p[--cnt];
        }
        return cycle;
    }

    // Usage example
    public static void main(String[] args) {
        List<Edge>[] graph =
        ↪  Stream.generate(ArrayList::new).limit(4).toArray(List[]::new);
        graph[0].add(new Edge(1, 1));
        graph[1].add(new Edge(0, 1));
        graph[1].add(new Edge(2, 1));
        graph[2].add(new Edge(3, -10));
        graph[3].add(new Edge(1, 1));
        int[] cycle = findNegativeCycle(graph);
        System.out.println(Arrays.toString(cycle));
    }
}
```

5.5 Floyd Warshall

```
import java.util.Arrays;

public class FloydWarshall {
    static final int INF = Integer.MAX_VALUE / 2;

    // precondition: d[i][i] == 0
    public static int[][] floydWarshall(int[][] d) {
        int n = d.length;
        int[][] pred = new int[n][n];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) pred[i][j] = (i == j || d[i][j] == INF) ?
            ↪  -1 : i;
        for (int k = 0; k < n; k++) {
            for (int i = 0; i < n; i++) {
                // if (d[i][k] == INF) continue;
                for (int j = 0; j < n; j++) {
                    // if (d[k][j] == INF) continue;
                    if (d[i][j] > d[i][k] + d[k][j]) {
```

```java
                        d[i][j] = d[i][k] + d[k][j];
                        // d[i][j] = Math.max(d[i][j], -INF);
                        pred[i][j] = pred[k][j];
                    }
                }
            }
        }
        for (int i = 0; i < n; i++)
            if (d[i][i] < 0)
                return null;
        return pred;
    }

    public static int[] restorePath(int[][] pred, int i, int j) {
        int n = pred.length;
        int[] path = new int[n];
        int pos = n;
        while (true) {
            path[--pos] = j;
            if (i == j)
                break;
            j = pred[i][j];
        }
        return Arrays.copyOfRange(path, pos, n);
    }

    // Usage example
    public static void main(String[] args) {
        int[][] dist = {{0, 3, 2}, {0, 0, 1}, {INF, 0, 0}};
        int[][] pred = floydWarshall(dist);
        int[] path = restorePath(pred, 0, 1);

        System.out.println(0 == dist[0][0]);
        System.out.println(2 == dist[0][1]);
        System.out.println(2 == dist[0][2]);
        System.out.println(-1 == pred[0][0]);
        System.out.println(2 == pred[0][1]);
        System.out.println(0 == pred[0][2]);
        System.out.println(Arrays.equals(new int[]{0, 2, 1}, path));
    }
}
```

5.6 Strongly Connected Components

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Stack;

public class SCComponents {
    int[] indexes;
    int[] lowlink;
    boolean[] onStack;
    int index;
    Stack<Node> s;
    List<List<Node>> SCCs; // List of Strongly Connected Components

    // Tarjan's Algorithm, which has the additional benefit of reverse topological
    ↪ sorting the SCCs
    public List<List<Node>> findComponents(Node[] nodes) {
        indexes = new int[nodes.length];
        lowlink = new int[nodes.length];
        onStack = new boolean[nodes.length];
        SCCs = new ArrayList<>();

        Arrays.fill(indexes, -1);
        Arrays.fill(lowlink, -1);
        Arrays.fill(onStack, false);

        index = 0;
        s = new Stack<>();
        for (Node n : nodes) {
            if (indexes[n.id] == -1) {
                strongConnect(n);
            }
        }
        return SCCs;
    }

    private void strongConnect(Node v) {
        indexes[v.id] = index;
        lowlink[v.id] = index++;
        onStack[v.id] = true;
        s.push(v);
        for (Edge e : v.adj) {
            Node w = e.destination();
            if (indexes[w.id] == -1) {
                strongConnect(w);
                lowlink[v.id] = Math.min(lowlink[v.id], lowlink[w.id]);
            } else if (onStack[w.id]) {
                lowlink[v.id] = Math.min(lowlink[v.id], indexes[w.id]);
            }
        }
        if (lowlink[v.id] == indexes[v.id]) {
            List<Node> SCC = new ArrayList<>();
            Node w;
            do {
                w = s.pop();
                onStack[w.id] = false;
                SCC.add(w);
            } while (w != v);
            SCCs.add(SCC);
        }
    }
}
```

5.7 Segment Tree LCA

```java
public class LCA {
    // LCA
    int[] height;
    List<Integer> euler;
    int[] first;
    int[] segTree;
    boolean[] visited;
```

```
    int n;

    LCA(List<Integer>[] adj, int root) {
        n = adj.length;
        height = new int[n];
        first = new int[n];
        euler = new ArrayList<>(2 * n);
        visited = new boolean[n];
        dfs(adj, root, 0);
        int m = euler.size();
        segTree = new int[m * 4];
        build(1, 0, m - 1);
    }

    void dfs(List<Integer>[] adj, int node, int h) {
        visited[node] = true;
        height[node] = h;
        first[node] = euler.size();
        euler.add(node);
        for (int to : adj[node]) {
            if (!visited[to]) {
                dfs(adj, to, h + 1);
                euler.add(node);
            }
        }
    }

    void build(int node, int b, int e) {
        if (b == e) {
            segTree[node] = euler.get(b);
        } else {
            int mid = (b + e) / 2;
            build(node << 1, b, mid);
            build(node << 1 | 1, mid + 1, e);
            int l = segTree[node << 1], r = segTree[node << 1 | 1];
            segTree[node] = (height[l] < height[r]) ? l : r;
        }
    }

    int query(int node, int b, int e, int L, int R) {
        if (b > R || e < L)
            return -1;
        if (b >= L && e <= R)
            return segTree[node];
        int mid = (b + e) >> 1;

        int left = query(node << 1, b, mid, L, R);
        int right = query(node << 1 | 1, mid + 1, e, L, R);
        if (left == -1) return right;
        if (right == -1) return left;
        return height[left] < height[right] ? left : right;
    }

    int lca(int u, int v) {
        int left = first[u], right = first[v];
        if (left > right) {
            int t = left;
            left = right;
            right = t;
        }
        return query(1, 0, euler.size() - 1, left, right);
    }
}
```

5.8 Binary Lifting LCA

```
// Needs testing. Adapted from cp-algorithms.

int numVertices = 0, logDepth = 0, timer = 0;
vector<vector<int>> adjList, up;
vector<int> timeIn, timeOut;

void dfs(int v, int p) {
    timeIn[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= logDepth; ++i)
        up[v][i] = up[up[v][i-1]][i-1];

    for (int u : adjList[v])
        if (u != p)
            dfs(u, v);

    timeOut[v] = ++timer;
}

// Answers if vertex u is an ancestor of vertex v.
bool is_ancestor(int u, int v) {
    return timeIn[u] <= timeIn[v] && timeOut[u] >= timeOut[v];
}

void preprocess(int root) {
    timeIn.resize(numVertices), timeOut.resize(numVertices);
    logDepth = ceil(log2(numVertices)); // <cmath> header
    up.resize(numVertices, vector<int>(logDepth + 1));

    dfs(root, root);
}

ll lca(int u, int v) {
    if (is_ancestor(u,v)) return u;
    if (is_ancestor(v,u)) return v;
    for (int i = logDepth; i >= 0; --i)
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    return up[u][0];
}
```

5.9 Bipartite Maximum Matching

```
// Bipartite Maximum Matching
// adjList has left elems only, which are adj to right elems.
// leftMatch[i] is matching right element, rightMatch[j] is matching left element.
// Assumes left elems are 0, ... , lSz-1 and right elems are 0, ... , rSz-1.

vector<vector<int>> adjList;
```

```
int leftSize, rightSize;
vector<bool> visited;
vector<int> leftMatch, rightMatch;

int dfs(int currLElem) {
    if (currLElem < 0) return 1;
    if (visited[currLElem]) return 0;
    visited[currLElem] = true;

    for (auto currRElem : adjList[currLElem]) {
        if (dfs(rightMatch[currRElem])) {
            leftMatch[currLElem] = currRElem;
            rightMatch[currRElem] = currLElem;
            return 1;
        }
    }
    return 0; //No matches found
}

int maxBipartiteMatching() {
    int max = 0;
    fill(leftMatch.begin(), leftMatch.end(), -1);
    fill(rightMatch.begin(), rightMatch.end(), -1);
    for (int currLElem = 0; currLElem < leftSize; currLElem++) {
        if (leftMatch[currLElem] < 0) {
            fill(visited.begin(), visited.end(), false);
            max += dfs(currLElem);
        }
    }
    return max;
}
```

5.10 Erdos-Gallai

```
// Erdos-Gallai - O(nlogn)
// check if it's possible to create a simple graph (undirected edges) from
// a sequence of vertex degrees
bool gallai(vector<int> v) {
  vector<ll> sum;
  sum.resize(v.size());

  sort(v.begin(), v.end(), greater<int>());
  sum[0] = v[0];
  for (int i = 1; i < v.size(); i++) sum[i] = sum[i-1] + v[i];
  if (sum.back() % 2) return 0;

  for (int k = 1; k < v.size(); k++) {
    int p = lower_bound(v.begin(), v.end(), k, greater<int>()) - v.begin();
    if (p < k) p = k;
    if (sum[k-1] > 1ll*k*(p-1) + sum.back() - sum[p-1]) return 0;
  }
  return 1;
}
```

6 Number Theory

6.1 CRT, GCD, LCM, Inverse, Pow, SQRT

```
public class NumberTheory {
    // calculates x S.T. x = a (mod m) and x = b (mod n)
    static long crt(long a, long m, long b, long n) {
        if (n > m) return crt(b, n, a, m);
        long[] out = eEuclid(m, n);
        long g = out[0];
        long x = out[1];
        long y = out[2];
        if ((a - b) % g != 0) {
            throw new IllegalArgumentException("No Solution");
        }
        assert ((a - b) % g == 0); // else no solution
        x = (b - a) % n * x % n / g * m + a;
        return x < 0 ? x + m * n / g : x;
    }

    static long lcm(long a, long b) {
        return a * b / gcd(a, b);
    }

    static long gcd(long a, long b) {
        while (a != 0) {
            long temp = a;
            a = b % a;
            b = temp;
        }
        return b;
    }

    // Bonus Extended Euclidean algorithm, returns an array for [g, x, y]
    // where g is the gcd, and x, y are the coefficients such that ax + by = g.
    // ax = 1  mod b
    // by = 1  mod a
    static long[] eEuclid(long a, long b) {
        long x = 1, y = 0, x1 = 0, y1 = 1, a1 = a, b1 = b;
        long t;
        while (b1 != 0) {
            long q = a1 / b1;   t = x1;
            x1 = x - q * x1;    x = t;
            t = y1;             y1 = y - q * y1;
            y = t;              t = b1;
            b1 = a1 - q * b1;   a1 = t;
        }
        return new long[]{a1, x, y};
    }

    // Smallest x S.T a^x = b (mod m)
    static long modLog(long a, long b, long m) {
        long n = (long) Math.sqrt(m) + 1;
        long e = 1, f = 1, j = 1;
        HashMap<Long, Long> A = new HashMap<>();
        while (j <= n && (e = f = e * a % m) != b % m)
            A.put(e * b % m, j++);
        if (e == b % m)
            return j;
```

```java
        if (gcd(m, e) == gcd(m, b)) {
            for (int i = 2; i < n + 2; i++) {
                if (A.containsKey(e = e * f % m)) {
                    return n * i - A.get(e);
                }
            }
        }
        return -1;
    }

    // smallest x S.T x^2 = a (mod p), p must be prime
    static long sqrt(long a, long p) {
        a %= p;
        if (a < 0)
            a += p;
        if (a == 0)
            return 0;
        if (modPow(a, (p - 1) / 2, p) != 1)
            throw new IllegalArgumentException("no solution");
        assert (modPow(a, (p - 1) / 2, p) == 1); // else no solution
        if (p % 4 == 3)
            return modPow(a, (p + 1) / 4, p);
        long s = p - 1, n = 2;
        int r = 0, m;
        while (s % 2 == 0) {
            ++r;
            s /= 2;
        }
        while (modPow(n, (p - 1) / 2, p) != p - 1)
            ++n;
        long x = modPow(a, (s + 1) / 2, p);
        long b = modPow(a, s, p), g = modPow(n, s, p);
        for (; ; r = m) {
            long t = b;
            for (m = 0; m < r && t != 1; ++m)
                t = t * t % p;
            if (m == 0)
                return x;
            long gs = modPow(g, 1L << (r - m - 1), p);
            g = gs * gs % p;
            x = x * gs % p;
            b = b * g % p;
        }
    }


    // evaluates a^b (mod p)
    static long modPow(long a, long b, long p) {
        long res = 1;
        while (b != 0) {
            if ((b & 1) == 1) {
                res *= a;
                res %= p;
            }
            a *= a;
            a %= p;
            b >>= 1;
        }
        return res;
    }

    // evaluates a^b (mod p)
    static long modRecPow(long a, long b, long p) {
        if (b == 0) {
            return 1;
        }
        long res;
        if ((b & 1) == 1) {
            res = a;
        } else {
            res = 1;
        }
        // (a * a)^(b/2) == a^b/2 * a ^ b/2
        return (res * modRecPow((a * a) % p, b >> 1, p)) % p;
    }

    // n choose k (mod m)
    static long choose(long n, long k, long m) {
        if (n - k < k) {
            return choose(n, n - k, m);
        }
        long ans = 1;
        for (int i = 1; i <= k; i++) {
            ans *= n - (k - i);
            ans %= m;
            ans *= eEuclid(i, m)[1];
            ans %= m;
        }
        return ans;
    }

    // find (x, y) such that a*x + b*y = c or return false if it's not possible
    // [x + k*b/gcd(a, b), y - k*a/gcd(a, b)] are also solutions
    long[] diof(long a, long b, long c) {
        long[] euclid = eEuclid(Math.abs(a), Math.abs(b));
        long g = Math.abs(euclid[0]);
        if (c % g != 0) return null;
        long x = euclid[1];
        long y = euclid[2];
        x *= c / g;
        y *= c / g;
        if (a < 0) x = -x;
        if (b < 0) y = -y;
        return new long[]{x, y};
    }

}
```

6.2 Primes

```java
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.IntStream;

public class Prime {
```

```java
    /**
     * Fast way to get all the primes up to some limit
     * @param n prime numbers to generate up to
     * @return A list of prime numbers
     */
    public static List<Integer> getPrimes(int n) {
        List<Integer> primes = new ArrayList<>();
        boolean[] isPrime = new boolean[n];
        Arrays.fill(isPrime, true);
        for (int i = 2; i < n; i++) {
            if (isPrime[i]) {
                primes.add(i);
                for (long j = (long) i * i; j < n; j += i) {
                    isPrime[(int) j] = false;
                }
            }
        }
        return primes;
    }

    // Euler's totient function
    public static int phi(int n) {
        int res = n;
        for (int i = 2; i * i <= n; i++)
            if (n % i == 0) {
                while (n % i == 0) n /= i;
                res -= res / i;
            }
        if (n > 1)
            res -= res / n;
        return res;
    }

    // Euler's totient function
    public static int[] generatePhi(int n) {
        int[] res = IntStream.range(0, n + 1).toArray();
        for (int i = 1; i <= n; i++)
            for (int j = i + i; j <= n; j += i) res[j] -= res[i];
        return res;
    }

    boolean isPrime(long n) {
        if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
        long[] A = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
        long s = Long.numberOfTrailingZeros(n-1);
        long d = n >> s;
        for (long a : A) {
            long p = NumberTheory.modPow(a % n, d, n);
            long i = s;
            while (p != 1 && p != n-1 && a % n != 0 && i-- != 0)
                p = p * p % n;
            if (p != n-1 && i != s)
                return false;
        }
        return true;
    }

}
```

7 Linear Algebra

7.1 Matrix Multipliaction

```cpp
typedef double T;

vector<vector<T>> MatMultiply(vector<vector<T>>& a, vector<vector<T>>& b) {
  const int n = a.size();
  const int m = b[0].size();
  const int z = a[0].size();
  vector<vector<T>> c(n, vector<T>(m, 0));
  if(b.size() != z){ cout << "matrices not the right size" << endl; exit(0);}
  for(int i = 0; i < n; i++)
    for(int j = 0; j < m; j++)
      for(int k = 0; k < z; k++)
        c[i][j] += a[i][k]*b[k][j];
  return c;
}
```

7.2 Inverse and Determinant

```cpp
typedef double T;
const double EPS = 1e-10;

// computes inverse and returns determinant of a, inverse will be stored in a
// solves Ax = B, solution will be stored in b, if b is not needed, just pass in identity
T GaussJordan(vector<vector<T>> &a, vector<vector<T>> &b) {
  const int n = a.size();
  const int m = b[0].size();
  vector<int> irow(n), icol(n), ipiv(n);
  T det = 1;

  for (int i = 0; i < n; i++) {
    int pj = -1, pk = -1;
    for (int j = 0; j < n; j++) if (!ipiv[j])
      for (int k = 0; k < n; k++) if (!ipiv[k])
        if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk = k; }
    if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular." << endl; exit(0); }
    ipiv[pk]++;
    swap(a[pj], a[pk]);
    swap(b[pj], b[pk]);
    if (pj != pk) det *= -1;
    irow[i] = pj;
    icol[i] = pk;

    T c = 1.0 / a[pk][pk];
    det *= a[pk][pk];
    a[pk][pk] = 1.0;
    for (int p = 0; p < n; p++) a[pk][p] *= c;
    for (int p = 0; p < m; p++) b[pk][p] *= c;
    for (int p = 0; p < n; p++) if (p != pk) {
      c = a[p][pk];
      a[p][pk] = 0;
      for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
```

```
      for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
    }
  }

  for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {
    for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
  }

  return det;
}

// computes nxn identity matrix
vector<vector<T>> Identity(int n){
    vector<vector<T>> a(n, vector<T>(n, 0));
    for(int i = 0; i < n; i++) a[i][i] = 1;
    return a;
}
```

7.3 Modular RREF

```
// Mod arithmetic for small modulos only. Resolves negative issue of % (e.g. -1 % 2 = -1 instead of 1).
vector<int> mod_inv;
int mod(int a, int m) {
    if (a < -m || a >= m) a = a % m;
    if (a < 0) a += m;
    return a;
}
void initModInv(int m) {
    mod_inv.resize(m, 0); mod_inv[1] = 1;
    for (int i = 2; i < m; ++i)
        mod_inv[i] = m - (m/i) * mod_inv[m%i] % m;
}
int modMult(int a, int b, int m) { return mod(a*b, m); }
int modAdd(int a, int b, int m) { return mod(a+b,m); }
int modSub(int a, int b, int m) { return mod(a-b,m); }
int modDiv(int a, int b, int m) { assert(b!=0); return mod(a*mod_inv[b], m); }

// Modular rref. Returns rank of matrix.
int rrefMod(vector<vector<int>> &a, int mod) {
    int n = a.size();
    int m = a[0].size();
    int r = 0;
    for (int c = 0; c < m && r < n; c++) {
        int j = r;
        for (int i = r + 1; i < n; i++)
            if (abs(a[i][c]) > abs(a[j][c]))
                j = i;
        if (a[j][c] == 0)
            continue;
        swap(a[j], a[r]);

        int s = mod_inv[a[r][c]];
        for (int j = 0; j < m; j++)
            a[r][j] = modMult(a[r][j], s, mod);
        for (int i = 0; i < n; i++)
            if (i != r) {
                int t = a[i][c];
                for (int j = 0; j < m; j++)
                    a[i][j] = modSub(a[i][j], modMult(t, a[r][j], mod), mod);
            }
        r++;
    }
    return r;
}

void testSystem(vector<vector<int>> linEqs) {
    vector<vector<int>> modEqs(100, vector<int>(100, 0));
    for (int i = 0; i < 100; i++)
        for (int j = 0; j < 100; j++)
            modEqs[i][j] = mod(linEqs[i][j], p);
    initModInv(p);
    rrefMod(modEqs, p);
}
```

7.4 RREF

```
// (Stanford) Reduced row echelon form via Gauss-Jordan elimination
// with partial pivoting.  This can be used for computing
// the rank of a matrix. O(n^3) runtime.
//
// OUTPUT:   rref[][] = an nxm matrix (stored in a[][])
//           returns rank of a[][]

const double eps = 1e-10;

typedef double T;

int rref(vector<vector<T>> &a) {
  int n = a.size();
  int m = a[0].size();
  int r = 0;
  for (int c = 0; c < m && r < n; c++) {
    int j = r;
    for (int i = r + 1; i < n; i++)
      if (fabs(a[i][c]) > fabs(a[j][c])) j = i;
    if (fabs(a[j][c]) < eps) continue;
    swap(a[j], a[r]);

    T s = 1.0 / a[r][c];
    for (int j = 0; j < m; j++) a[r][j] *= s;
    for (int i = 0; i < n; i++) if (i != r) {
      T t = a[i][c];
      for (int j = 0; j < m; j++) a[i][j] -= t * a[r][j];
    }
    r++;
  }
  return r;
}

int main() {
  const int n = 5, m = 4;
  double A[n][m] = {
    {16,  2,  3, 13},
    { 5, 11, 10,  8},
```

```
    { 9,  7,  6, 12},
    { 4, 14, 15,  1},
    {13, 21, 21, 13}};
  vector<vector<T>> a(n);
  for (int i = 0; i < n; i++)
    a[i] = vector<T>(A[i], A[i] + m);

  int rank = rref(a);

  // expected: 3
  cout << "Rank: " << rank << endl;

  // expected: 1 0 0 1
  //           0 1 0 3
  //           0 0 1 -3
  //           0 0 0 3.10862e-15
  //           0 0 0 2.22045e-15
  cout << "rref: " << endl;
  for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 4; j++)
      cout << a[i][j] << ' ';
    cout << endl;
  }
}
```

8 Misc

8.1 Bit Tricks

```
// Gosper's Hack: Iterates all size k subsets (belonging to set of n elems).
int mask = (1 << k) - 1, x, y;
while (mask <= (1 << n) - (1 << (n-k))) {
    // Code here
    x = mask&-mask, y = mask+x, mask = y | (((y^mask)>>2)/x);
}

// Get LSB of X
x & -x

// Submask Emumeration: Loops over all subset masks of m (except m itself).
for (int x = m; x; ) { --x &= m; ... }

// Uppercase to lowercase: ch |= ' ';
// Lowercase to uppercase: ch &= '_';

// Int bitset operations. Alternative is bitset (#include <bitset>) which does
// not have subtraction (thus has an uglier implementation for submask iteration)
#define iBitset unsigned __int128
void add(iBitset& set, int elem) { set |= (iBitset)1<<elem; }
void remove(iBitset& set, int elem) { set &= ~((iBitset)1<<elem); }
bool isIn(iBitset set, int elem) { return (((iBitset)1<<elem) & set); }
iBitset bSetUnion(iBitset s1, iBitset s2) { return s1 | s2; }
iBitset bSetIntersect(iBitset s1, iBitset s2) { return s1 & s2; }
iBitset bSetComplement(iBitset set)
    { return ~set & (((iBitset)1<<128) - 1); } // Throws -Wall warning
void printBits(iBitset bSet) {
    for (int i = 127; i >= 0; i--)
        cout << ((bSet & ((iBitset)1<<i)) ? '1' : '0');
    cout << endl;
}
vector<int> getElemsInBitset(iBitset bSet) {
    vector<int> elems;
    for (int i = 0; i < 128; i++)
        if (isIn(bSet, i)) elems.push_back(i);
    return elems;
}
```

8.2 CPP Builtins

```
__builtin_popcount(x): Counts number of set bits in an integer.
__builtin_parity(x): Parity of number. true(1) for odd parity, false(0) for even.
__builtin_clz(x): Counts the leading zeros of the integer.
__builtin_ctz(x): Counts the trailing zeros of the integer.
```

8.3 CPP Syntax Examples

```
// (Stanford) Example for using stringstreams and next_permutation
int main(void){
    vector<int> v = {1,2,3,4};

    // Expected output: 1 2 3 4
    //                  1 2 4 3
    //                  ...
    //                  4 3 2 1
    do {
        ostringstream oss;
        oss << v[0] << " " << v[1] << " " << v[2] << " " << v[3];

        // for input from a string s,
        //   istringstream iss(s);
        //   iss >> variable;

        cout << oss.str() << endl;
    } while (next_permutation (v.begin(), v.end()));

    v.clear();

    v.push_back(1); v.push_back(2); v.push_back(1); v.push_back(3);

    // To use unique, first sort numbers.  Then call
    // unique to place all the unique elements at the beginning
    // of the vector, and then use erase to remove the duplicate
    // elements.

    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());

    // Expected output: 1 2 3
    for (size_t i = 0; i < v.size(); i++)
    cout << v[i] << " ";
    cout << endl;
}
```

```

// Use of mt19937 for generating random numbers in range (0, MAX):
random_device device;
mt19937 generator(device());
uniform_int_distribution<int> unifDist(0,MAX);
int randNum = unifDist(generator)

// Use of chrono for getting program runtime:
#include <chrono>
auto start = chrono::high_resolution_clock::now();
auto duration = chrono::duration_cast<chrono::milliseconds>
                    (chrono::high_resolution_clock::now() - start);
if (duration.count() > 1000) { cout << "fail\n"; return 0; } // 1 second

// Bitsets:
#include <bitset>
_Find_first(i);
_Find_next(i);
// This code prints all set bits of BS:
for (int i = BS._Find_first(); i < BS.size(); i = BS._Find_next(i))
    cout << i << endl;

// Custom Sort Function:
struct less_than_key {
    inline bool operator() (const MyDS& ds1, const MyDS& struct2) {
        return (struct1.key < struct2.key);
    }
};
vector <MyDS> vec;
sort(vec.begin(), vec.end(), less_than_key());

// Lambdas:
sort(x, x + n,
        [](float a, float b) {return (std::abs(a) < std::abs(b));}
    );
// Capture clause: [x] captures x by value, [&x] captures by reference
```

8.4 Bisect & Ternary Search

```
private static int bisectRight(int[] A, int k, int s) {
    int lo = 0;
    int hi = s;
    while (lo < hi) {
        int mid = (lo + hi) / 2;
        if (A[mid] <= k) {
            lo = mid + 1;
        } else {
            hi = mid;
        }
    }
    return lo;
}

public static int bisectLeft(int[] A, int k, int s) {
    int lo = 0;
    int hi = s;
    while (lo < hi) {
        int mid = (lo + hi) / 2;
        if (A[mid] < k) {
            lo = mid + 1;
        } else {
            hi = mid;
        }
    }
    return lo;
}

private static int ternarySearch(int[] A) {
    int lo = 0;
    int hi = A.length;
    while (lo < hi) {
        int lm = (hi - lo) / 3;
        int hm = lo + 2 * lm;
        lm += lo;
        int lv = A[lm];
        int hv = A[hm];
        if (lv < hv) {
            hi = hm;
        } else {
            lo = lm + 1;
        }
    }
    return lo;
}

```

8.5 Knapsack

```
// c - capacity of knapsack; n - number of items
private static List<Integer> knapSack(int[] values, int[] weights, int c, int n) {
    int[][] dp = new int[n][c + 1];
    boolean[][] used = new boolean[n][c + 1];
    for (int i = 0; i < weights[n - 1] && i <= c; i++) {
        dp[n - 1][i] = 0;
    }
    for (int i = weights[n - 1]; i <= c; i++) {
        used[n - 1][i] = true;
        dp[n - 1][i] = values[n - 1];
    }
    for (int i = n - 2; i >= 0; i--) {
        for (int j = 0; j < weights[i] && j <= c; j++) {
            dp[i][j] = dp[i + 1][j];
        }
        for (int j = weights[i]; j <= c; j++) {
            int a = values[i] + dp[i + 1][j - weights[i]];
            int b = dp[i + 1][j];
            dp[i][j] = Math.max(a, b);
            used[i][j] = a > b;
        }
    }
    List<Integer> items = new ArrayList<>();
    for (int i = 0; i < n && c > 0; i++) {
        if (used[i][c]) {
            items.add(i);
            c -= weights[i];
```

```
        }
    }
    return items;
}
```

8.6 Longest Increasing Subsequence

```
// Needs bisect right
private static int[] LIS(int[] A) {
    int[] partialSequence = new int[A.length]; // An optimal partial sequence, may
    ↪  not exist in A
    int[] indexes = new int[A.length]; // Index mapping of the partialSequence
    int[] parent = new int[A.length]; // parent pointer used to reconstruct
    ↪  sequence at the end
    int length = 0;
    for (int n = 0; n < A.length; n++) {
        int i = bisectRight(partialSequence, A[n], length); // Find optimal
        ↪  placement
        partialSequence[i] = A[n]; // place
        indexes[i] = n;
        if (i != 0) {
            parent[n] = indexes[i - 1]; // set parent to be previous in sequence
        }
        if (i == length) {
            length++; // increment length if increased
        }
    }
    int[] sequence = new int[length]; // recover sequence
    int cur = indexes[length - 1];
    for (int i = length - 1; i >= 0; i--) {
        sequence[i] = A[cur];
        cur = parent[cur];
    }
    return sequence;
}
```

8.7 Fast Fourier Transform

```
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> & a, bool invert) {
    int n = a.size();
    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;
        if (i < j)
            swap(a[i], a[j]);
    }
    for (int len = 2; len <= n; len <<= 1) {
        double ang = 2 * PI / len * (invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int j = 0; j < len / 2; j++) {
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }
    if (invert)
        for (cd & x : a)
            x /= n;
}

vector<int> multiply(vector<int> const& a, vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
```

8.8 Polynomial Interpolation

Description: Given n points (x[i], y[i]), computes an n-1-degree polynomial p that passes through them: $p(x) = a[0] * x^0 + ... + a[n-1] * x^{n-1}$. For numerical precision, pick $x[k] = c * \cos(k/(n-1) * \pi), k = 0 \dots n-1$.

```
vector<double> interpolate(vector<double> x, vector<double> y, int n) {
        vector<double> res(n), temp(n);
        rep(k,0,n-1) rep(i,k+1,n)
                y[i] = (y[i] - y[k]) / (x[i] - x[k]);
        double last = 0; temp[0] = 1;
        rep(k,0,n) rep(i,0,n) {
                res[i] += y[k] * temp[i];
                swap(last, temp[i]);
                temp[i] -= last * x[k];
        }
        return res;
}
```

8.9 Dates

```
vector<int> monthDays = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
string dayOfWeek[] = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};

// converts Gregorian date to integer (Julian day number)
// Months from 1-12, days from 1-31
int dateToInt (int m, int d, int y){
  return
    1461 * (y + 4800 + (m - 14) / 12) / 4 +
    367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
    3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
    d - 32075;
}

// converts integer (Julian day number) to Gregorian date: month/day/year
void intToDate (int jd, int &m, int &d, int &y){
  int x, n, i, j;
  x = jd + 68569;
  n = 4 * x / 146097;
  x -= (146097 * n + 3) / 4;
  i = (4000 * (x + 1)) / 1461001;
  x -= 1461 * i / 4 - 31;
  j = 80 * x / 2447;
  d = x - 2447 * j / 80;
  x = j / 11;
  m = j + 2 - 12 * x;
  y = 100 * (n - 49) + i + x;
}

// converts integer (Julian day number) to day of week
string intToDay (int jd){
  return dayOfWeek[jd % 7];
}
```

8.10 Java Templates

```
public static void main(String[] args) {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));
    //sorts
    int[][] a = {{2, 1},{1, 3},{3, 2}};
    // sorts a by first element
    Arrays.sort(a, new Comparator<int[]>() {
            @Override
            public int compare(int[] o1, int[] o2) {
                return Integer.compare(o1[0], o2[0]);
            }
        });
    Arrays.sort(a, Comparator.comparingInt(o -> o[0]));
    PriorityQueue<int[]> q = new PriorityQueue<>(Comparator.comparingInt(o->o[0]));
    //dates
    LocalDate date = LocalDate.of(1901, 1, 1); // January 1st
    date = date.plusMonths(1);
    date.getDayOfWeek() == DayOfWeek.SUNDAY;

    br.close();
    bw.close();
}
```

Geometry (9)

9.1 Geometric primitives

Point.h

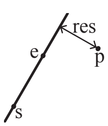
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

47ec0a, 28 lines

```
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T x=0, T y=0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
  bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
  P operator+(P p) const { return P(x+p.x, y+p.y); }
  P operator-(P p) const { return P(x-p.x, y-p.y); }
  P operator*(T d) const { return P(x*d, y*d); }
  P operator/(T d) const { return P(x/d, y/d); }
  T dot(P p) const { return x*p.x + y*p.y; }
  T cross(P p) const { return x*p.y - y*p.x; }
  T cross(P a, P b) const { return (a-*this).cross(b-*this); }
  T dist2() const { return x*x + y*y; }
  double dist() const { return sqrt((double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this/dist(); } // makes dist()=1
  P perp() const { return P(-y, x); } // rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the origin
  P rotate(double a) const {
    return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
  friend ostream& operator<<(ostream& os, P p) {
    return os << "(" << p.x << "," << p.y << ")"; }
};
```

lineDistance.h

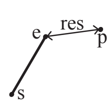
Description:
Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance. For Point3D, call .dist on the result of the cross product.



"Point.h" f6bf6b, 4 lines

```
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
  return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

SegmentDistance.h



Description:
Returns the shortest distance between point p and the line segment from point s to e.

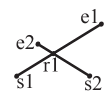
Usage:
```
Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;
```

"Point.h" 5c88f4, 6 lines

```
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
  if (s==e) return (p-s).dist();
  auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
  return ((p-s)*d-(e-s)*t).dist()/d;
}
```

SegmentIntersection.h

Description:
If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.



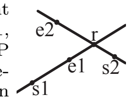
Usage:
```
vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
cout << "segments intersect at " << inter[0] << endl;
```

"Point.h", "OnSegment.h" 9d57f2, 13 lines

```
template<class P> vector<P> segInter(P a, P b, P c, P d) {
  auto oa = c.cross(d, a), ob = c.cross(d, b),
       oc = a.cross(b, c), od = a.cross(b, d);
  // Checks if intersection is single non-endpoint point.
  if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
    return {(a * ob - b * oa) / (ob - oa)};
  set<P> s;
  if (onSegment(c, d, a)) s.insert(a);
  if (onSegment(c, d, b)) s.insert(b);
  if (onSegment(a, b, c)) s.insert(c);
  if (onSegment(a, b, d)) s.insert(d);
  return {all(s)};
}
```

lineIntersection.h

Description:
If a unique intersection point of the lines going through s1,e1 and s2,e2 exists {1, point} is returned. If no intersection point exists {0, (0,0)} is returned and if infinitely many exists {-1, (0,0)} is returned. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or ll.



Usage:
```
auto res = lineInter(s1,e1,s2,e2);
if (res.first == 1)
cout << "intersection point at " << res.second << endl;
```

"Point.h" a01f81, 8 lines

```
template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
  auto d = (e1 - s1).cross(e2 - s2);
  if (d == 0) // if parallel
    return {-(s1.cross(e1, s2) == 0), P(0, 0)};
  auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
  return {1, (s1 * p + e1 * q) / d};
}
```

sideOf.h

Description: Returns where p is as seen from s towards e. 1/0/-1 $\Leftrightarrow$ left/on line/right. If the optional argument *eps* is given 0 is returned if p is within distance *eps* from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.

Usage:
```
bool left = sideOf(p1,p2,q)==1;
```

"Point.h" 3af81c, 9 lines

```
template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }

template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
  auto a = (e-s).cross(p-s);
  double l = (e-s).dist()*eps;
  return (a > l) - (a < -l);
}
```

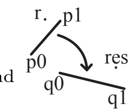
OnSegment.h

Description: Returns true iff p lies on the line segment from s to e. Use `(segDist(s,e,p)<=epsilon)` instead when using Point<double>.

"Point.h" c597e8, 3 lines

```
template<class P> bool onSegment(P s, P e, P p) {
  return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

linearTransformation.h



Description:
Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.

"Point.h" 03a306, 6 lines

```
typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
    const P& q0, const P& q1, const P& r) {
  P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
  return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
}
```

Angle.h

Description: A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.

Usage:
```
vector<Angle> v = {w[0], w[0].t360() ...}; // sorted
int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }
// sweeps j such that (j-i) represents the number of positively
oriented triangles with vertices at 0 and i
```

0f0602, 35 lines

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
  int half() const {
    assert(x || y);
    return y < 0 || (y == 0 && x < 0);
  }
  Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
  Angle t180() const { return {-x, -y, t + half()}; }
  Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also compare distances
  return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
         make_tuple(b.t, b.half(), a.x * (ll)b.y);
}

// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
  if (b < a) swap(a, b);
  return (b < a.t180() ?
          make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
  Angle r(a.x + b.x, a.y + b.y, a.t);
  if (a.t180() < r) r.t--;
  return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
  int tu = b.t - a.t; a.t = b.t;
  return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

9.2 Circles

CircleIntersection.h

Description: Computes the pair of points at which two circles intersect. Returns false in case of no intersection.

"Point.h" 84d6d3, 11 lines

```
typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out) {
  if (a == b) { assert(r1 != r2); return false; }
  P vec = b - a;
  double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
         p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
  if (sum*sum < d2 || dif*dif > d2) return false;
  P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
  *out = {mid + per, mid - per};
  return true;
}
```

CircleTangents.h

Description: Finds the external tangents of two circles, or internal if r2 is negated. Can return 0, 1, or 2 tangents – 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers). .first and .second give the tangency points at circle 1 and 2 respectively. To find the tangents of a circle with a point set r2 to 0.

"Point.h" b0153d, 13 lines

```
template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2) {
  P d = c2 - c1;
  double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
  if (d2 == 0 || h2 < 0)  return {};
  vector<pair<P, P>> out;
  for (double sign : {-1, 1}) {
    P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
    out.push_back({c1 + v * r1, c2 + v * r2});
  }
  if (h2 == 0) out.pop_back();
  return out;
}
```

CirclePolygonIntersection.h

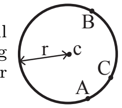
Description: Returns the area of the intersection of a circle with a ccw polygon.
Time: $\mathcal{O}(n)$

"../../content/geometry/Point.h" a1ee63, 19 lines

```
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
  auto tri = [&](P p, P q) {
    auto r2 = r * r / 2;
    P d = q - p;
    auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
    if (t < 0 || 1 <= s) return arg(p, q) * r2;
    P u = p + d * s, v = p + d * t;
    return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
  };
  auto sum = 0.0;
  rep(i,0,sz(ps))
    sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
  return sum;
}
```

circumcircle.h

Description:



The circumcirle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.

"Point.h" 1caa3a, 9 lines

```
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
  return (B-A).dist()*(C-B).dist()*(A-C).dist()/
      abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
  P b = C-A, c = B-A;
  return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

MinimumEnclosingCircle.h

Description: Computes the minimum circle that encloses a set of points.
Time: expected $\mathcal{O}(n)$

"circumcircle.h" 09dd0a, 17 lines

```
pair<P, double> mec(vector<P> ps) {
  shuffle(all(ps), mt19937(time(0)));
  P o = ps[0];
  double r = 0, EPS = 1 + 1e-8;
  rep(i,0,sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
    o = ps[i], r = 0;
    rep(j,0,i) if ((o - ps[j]).dist() > r * EPS) {
      o = (ps[i] + ps[j]) / 2;
      r = (o - ps[i]).dist();
      rep(k,0,j) if ((o - ps[k]).dist() > r * EPS) {
        o = ccCenter(ps[i], ps[j], ps[k]);
        r = (o - ps[i]).dist();
      }
    }
  }
  return {o, r};
}
```

9.3 Polygons

InsidePolygon.h

Description: Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.
Usage: `vector<P> v = {P{4,4}, P{1,2}, P{2,1}};`
`bool in = inPolygon(v, P{3, 3}, false);`
Time: $\mathcal{O}(n)$

"Point.h", "OnSegment.h", "SegmentDistance.h" 2bf504, 11 lines

```
template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
  int cnt = 0, n = sz(p);
  rep(i,0,n) {
    P q = p[(i + 1) % n];
    if (onSegment(p[i], q, a)) return !strict;
    //or: if (segDist(p[i], q, a) <= eps) return !strict;
    cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
  }
  return cnt;
}
```

PolygonArea.h

Description: Returns twice the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!

"Point.h" f12300, 6 lines

```
template<class T>
T polygonArea2(vector<Point<T>>& v) {
  T a = v.back().cross(v[0]);
  rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
  return a;
}
```

PolygonCenter.h

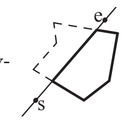
Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$

"Point.h" 9706dc, 9 lines

```
typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
  P res(0, 0); double A = 0;
  for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
    res = res + (v[i] + v[j]) * v[j].cross(v[i]);
    A += v[j].cross(v[i]);
  }
  return res / A / 3;
}
```

PolygonCut.h

Description:



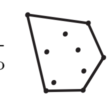
Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.
Usage: `vector<P> p = ...;`
`p = polygonCut(p, P(0,0), P(1,0));`

"Point.h", "lineIntersection.h" f2b7d4, 13 lines

```
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
  vector<P> res;
  rep(i,0,sz(poly)) {
    P cur = poly[i], prev = i ? poly[i-1] : poly.back();
    bool side = s.cross(e, cur) < 0;
    if (side != (s.cross(e, prev) < 0))
      res.push_back(lineInter(s, e, cur, prev).second);
    if (side)
      res.push_back(cur);
  }
  return res;
}
```

ConvexHull.h

Description:



Returns a vector of the points of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.
Time: $\mathcal{O}(n \log n)$

"Point.h" 310954, 13 lines

```
typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
  if (sz(pts) <= 1) return pts;
  sort(all(pts));
  vector<P> h(sz(pts)+1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(all(pts)))
    for (P p : pts) {
      while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;
      h[t++] = p;
    }
  return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
```

HullDiameter.h

Description: Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).

"Point.h" c571b8, 12 lines

```
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
```

```
  int n = sz(S), j = n < 2 ? 0 : 1;
  pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
  rep(i,0,j)
    for (;; j = (j + 1) % n) {
      res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
      if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
        break;
    }
  return res.second;
}
```

PointInsideHull.h

Description: Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.
Time: $\mathcal{O}(\log N)$

"Point.h", "sideOf.h", "OnSegment.h" 71446b, 14 lines

```
typedef Point<ll> P;

bool inHull(const vector<P>& l, P p, bool strict = true) {
  int a = 1, b = sz(l) - 1, r = !strict;
  if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
  if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
  if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p)<= -r)
    return false;
  while (abs(a - b) > 1) {
    int c = (a + b) / 2;
    (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
  }
  return sgn(l[a].cross(l[b], p)) < r;
}
```

LineHullIntersection.h

Description: Line-convex polygon intersection. The polygon must be ccw and have no collinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon: $\bullet$ $(-1,-1)$ if no collision, $\bullet$ $(i,-1)$ if touching the corner i, $\bullet$ (i,i) if along side $(i,i+1)$, $\bullet$ (i,j) if crossing sides $(i,i+1)$ and $(j,j+1)$. In the last case, if a corner i is crossed, this is treated as happening on side $(i,i+1)$. The points are returned in the same order as the line hits the polygon. `extrVertex` returns the point of a hull with the max projection onto a line.
Time: $\mathcal{O}(\log n)$

"Point.h" 7cf45b, 39 lines

```
#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
  int n = sz(poly), lo = 0, hi = n;
  if (extr(0)) return 0;
  while (lo + 1 < hi) {
    int m = (lo + hi) / 2;
    if (extr(m)) return m;
    int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
    (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
  }
  return lo;
}

#define cmpL(i) sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
  int endA = extrVertex(poly, (a - b).perp());
  int endB = extrVertex(poly, (b - a).perp());
  if (cmpL(endA) < 0 || cmpL(endB) > 0)
    return {-1, -1};
  array<int, 2> res;
  rep(i,0,2) {
    int lo = endB, hi = endA, n = sz(poly);
    while ((lo + 1) % n != hi) {
      int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
      (cmpL(m) == cmpL(endB) ? lo : hi) = m;
    }
    res[i] = (lo + !cmpL(hi)) % n;
    swap(endA, endB);
  }
  if (res[0] == res[1]) return {res[0], -1};
  if (!cmpL(res[0]) && !cmpL(res[1]))
    switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
      case 0: return {res[0], res[0]};
      case 2: return {res[1], res[1]};
    }
  return res;
}
```

9.4 Misc. Point Set Problems

ClosestPair.h

Description: Finds the closest pair of points.
Time: $\mathcal{O}(n \log n)$

"Point.h" ac41a6, 17 lines

```
typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
  assert(sz(v) > 1);
  set<P> S;
  sort(all(v), [](P a, P b) { return a.y < b.y; });
  pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
  int j = 0;
  for (P p : v) {
    P d{1 + (ll)sqrt(ret.first), 0};
    while (v[j].y <= p.y - d.x) S.erase(v[j++]);
    auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
    for (; lo != hi; ++lo)
      ret = min(ret, {(*lo - p).dist2(), {*lo, p}});
    S.insert(p);
  }
  return ret.second;
}
```

kdTree.h

Description: KD-tree (2d, can be extended to 3d)

"Point.h" bac5b0, 63 lines

```
typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();

bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }

struct Node {
  P pt; // if this is a leaf, the single point in it
  T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
  Node *first = 0, *second = 0;

  T distance(const P& p) { // min squared distance to a point
    T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
    T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
    return (P(x,y) - p).dist2();
  }

  Node(vector<P>&& vp) : pt(vp[0]) {
    for (P p : vp) {
      x0 = min(x0, p.x); x1 = max(x1, p.x);
      y0 = min(y0, p.y); y1 = max(y1, p.y);
    }
    if (vp.size() > 1) {
      // split on x if width >= height (not ideal...)
      sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
      // divide by taking half the array for each child (not
      // best performance with many duplicates in the middle)
      int half = sz(vp)/2;
      first = new Node({vp.begin(), vp.begin() + half});
      second = new Node({vp.begin() + half, vp.end()});
    }
  }
};

struct KDTree {
  Node* root;
  KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}

  pair<T, P> search(Node *node, const P& p) {
    if (!node->first) {
      // uncomment if we should not find the point itself:
      // if (p == node->pt) return {INF, P()};
      return make_pair((p - node->pt).dist2(), node->pt);
    }

    Node *f = node->first, *s = node->second;
    T bfirst = f->distance(p), bsec = s->distance(p);
    if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

    // search closest side first, other side if needed
    auto best = search(f, p);
    if (bsec < best.first)
      best = min(best, search(s, p));
    return best;
  }

  // find nearest point to a point, and its squared distance
  // (requires an arbitrary operator< for Point)
  pair<T, P> nearest(const P& p) {
    return search(root, p);
  }
};
```

FastDelaunay.h

Description: Fast Delaunay triangulation. Each circumcircle contains none of the input points. There must be no duplicate points. If all points are on a line, no triangles will be returned. Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in order {t[0][0], t[0][1], t[0][2], t[1][0], ...}, all counter-clockwise.
Time: $\mathcal{O}(n \log n)$

"Point.h" eefdf5, 88 lines

```
typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point

struct Quad {
  Q rot, o; P p = arb; bool mark;
  P& F() { return r()->p; }
  Q& r() { return rot->rot; }
  Q prev() { return rot->o->rot; }
  Q next() { return r()->prev(); }
} *H;

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
  lll p2 = p.dist2(), A = a.dist2()-p2,
      B = b.dist2()-p2, C = c.dist2()-p2;
  return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}
Q makeEdge(P orig, P dest) {
  Q r = H ? H : new Quad{new Quad{new Quad{new Quad{0}}}};
  H = r->o; r->r()->r() = r;
  rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r();
  r->p = orig; r->F() = dest;
  return r;
```

```
}
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
}

pair<Q,Q> rec(const vector<P>& s) {
  if (sz(s) <= 3) {
    Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
    if (sz(s) == 2) return { a, a->r() };
    splice(a->r(), b);
    auto side = s[0].cross(s[1], s[2]);
    Q c = side ? connect(b, a) : 0;
    return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
  }

#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
  Q A, B, ra, rb;
  int half = sz(s) / 2;
  tie(ra, A) = rec({all(s) - half});
  tie(B, rb) = rec({sz(s) - half + all(s)});
  while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
         (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
  Q base = connect(B->r(), A);
  if (A->p == ra->p) ra = base->r();
  if (B->p == rb->p) rb = base;

#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
    while (circ(e->dir->F(), H(base), e->F())) { \
      Q t = e->dir; \
      splice(e, e->prev()); \
      splice(e->r(), e->r()->prev()); \
      e->o = H; H = e; e = t; \
    }
  for (;;) {
    DEL(LC, base->r(), o);  DEL(RC, base, prev());
    if (!valid(LC) && !valid(RC)) break;
    if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
      base = connect(RC, base->r());
    else
      base = connect(base->r(), LC->r());
  }
  return { ra, rb };
}

vector<P> triangulate(vector<P> pts) {
  sort(all(pts));  assert(unique(all(pts)) == pts.end());
  if (sz(pts) < 2) return {};
  Q e = rec(pts).first;
  vector<Q> q = {e};
  int qi = 0;
  while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
  q.push_back(c->r()); c = c->next(); } while (c != e); }
  ADD; pts.clear();
  while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
  return pts;
}
```

9.5 3D

PolyhedronVolume.h

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

3058c3, 6 lines

```
template<class V, class L>
double signedPolyVolume(const V& p, const L& trilist) {
  double v = 0;
  for (auto i : trilist) v += p[i.a].cross(p[i.b]).dot(p[i.c]);
  return v / 6;
}
```

Point3D.h

Description: Class to handle points in 3D space. T can be e.g. double or long long.

8058ae, 32 lines

```
template<class T> struct Point3D {
  typedef Point3D P;
  typedef const P& R;
  T x, y, z;
  explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
  bool operator<(R p) const {
    return tie(x, y, z) < tie(p.x, p.y, p.z); }
  bool operator==(R p) const {
    return tie(x, y, z) == tie(p.x, p.y, p.z); }
  P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
  P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
  P operator*(T d) const { return P(x*d, y*d, z*d); }
  P operator/(T d) const { return P(x/d, y/d, z/d); }
  T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
  P cross(R p) const {
    return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
  }
  T dist2() const { return x*x + y*y + z*z; }
  double dist() const { return sqrt((double)dist2()); }
  //Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
  double phi() const { return atan2(y, x); }
  //Zenith angle (latitude) to the z-axis in interval [0, pi]
  double theta() const { return atan2(sqrt(x*x+y*y),z); }
  P unit() const { return *this/(T)dist(); } //makes dist()=1
  //returns unit vector normal to *this and p
  P normal(P p) const { return cross(p).unit(); }
  //returns point rotated 'angle' radians ccw around axis
  P rotate(double angle, P axis) const {
    double s = sin(angle), c = cos(angle); P u = axis.unit();
    return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
  }
};
```

3dHull.h

Description: Computes all faces of the 3-dimension hull of a point set. *No four points must be coplanar*, or else random results will be returned. All faces will point outwards.

Time: $\mathcal{O}(n^2)$

"Point3D.h" 5b45fc, 49 lines

```
typedef Point3D<double> P3;

struct PR {
  void ins(int x) { (a == -1 ? a : b) = x; }
  void rem(int x) { (a == x ? a : b) = -1; }
  int cnt() { return (a != -1) + (b != -1); }
  int a, b;
};

struct F { P3 q; int a, b, c; };

vector<F> hull3d(const vector<P3>& A) {
  assert(sz(A) >= 4);
  vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
  vector<F> FS;
  auto mf = [&](int i, int j, int k, int l) {
    P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
    if (q.dot(A[l]) > q.dot(A[i]))
      q = q * -1;
    F f{q, i, j, k};
    E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
    FS.push_back(f);
  };
  rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
    mf(i, j, k, 6 - i - j - k);

  rep(i,4,sz(A)) {
    rep(j,0,sz(FS)) {
      F f = FS[j];
      if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
        E(a,b).rem(f.c);
        E(a,c).rem(f.b);
        E(b,c).rem(f.a);
        swap(FS[j--], FS.back());
        FS.pop_back();
      }
    }
    int nw = sz(FS);
    rep(j,0,nw) {
      F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);
      C(a, b, c); C(a, c, b); C(b, c, a);
    }
  }
  for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
    A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
  return FS;
};
```

sphericalDistance.h

Description: Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 (ϕ_1) and f2 (ϕ_2) from x axis and zenith angles (latitude) t1 (θ_1) and t2 (θ_2) from z axis (0 = north pole). All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. dx*radius is then the difference between the two points in the x direction and d*radius is the total distance between the points.

611f07, 8 lines

```
double sphericalDistance(double f1, double t1,
    double f2, double t2, double radius) {
  double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
  double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
  double dz = cos(t2) - cos(t1);
  double d = sqrt(dx*dx + dy*dy + dz*dz);
  return radius*2*asin(d/2);
}
```