# Contents

# Listings

# 1 Number Theory

## 1.1 GCD and LCM

A common divisor $c$ of two integers $a$ and $b$ is an integer such that $a|c$ and $b|c$. The greatest common divisor $d$ is the largest such $c$, and has the property that for all common divisors $c$, $c|d$.
Two integers are said to be relatively prime if their greatest common divisor is 1.
We can find the greatest common divisor of two numbers using Eulclid's algorithm:

Listing 1: Finding the greatest common divisor

```
public static int gcd(int a, int b)
{
    if (b > a)
        return gcd(b, a);
    int rem = a % b;
    while (rem > 0)
    {
        a = b;
        b = rem;
        rem = a % b;
    }
    return b;
}

public static int gcd(int... nums)
{
    if (nums.length == 0)
        return 0;
    else if (nums.length == 1)
        return nums[0];
    int cur = gcd(nums[0], nums[1]);
    for (int i = 2; i < nums.length; i++)
        cur = gcd(cur, nums[i]);
    return cur;
}
```

The least common multiple can be obtained by dividing the product by the greatest common divisor:

Listing 2: Finding the least common multiple

```
public static int lcm(int a, int b)
{
    return a * (b / gcd(a, b));
}

public static int lcm(int... nums)
{
    int cur = 1;
    for (int i = 0; i < nums.length; i++)
        cur = lcm(cur, nums[i]);
    return cur;
}
```

## 1.2 Diophantine Equations

Diophantine equations are equations in which variables only take on integer values. The typical form of a linear diophantine equation is $ax + by = c$, and there are infinitely solutions if $d = \gcd(a, b) \mid c$ and no solutions otherwise. We can obtain solutions by first obtaining an initial solution with the Bezout coefficients[1] of $a$ and $b$:

[1] Bezout's identity states that the greatest common divisor of two integers can always be obtained as a linear combination the two integers.

Listing 3: Obtaining Bezout coefficients

```java
/* returns an array bezout such that
 * bezout[0] = gcd(a, b) and
 * bezout[1] * a + bezout[2] * b = bezout[0] */
public static int[] bezout(int a, int b)
{
    if (b < 0)
    {
        int[] flip = bezout(a, -b);
        return new int[] {flip[0], flip[1], -flip[2]};
    }
    if (b > a)
    {
        int[] flip = bezout(b, a);
        return new int[] {flip[0], flip[2], flip[1]};
    }
    int rem = a % b;
    int quot = a / b;
    int s3, s2, s1, t3, t2, t1;
    s2 = 1;
    s1 = 0;
    t2 = 0;
    t1 = 1;
    while (rem > 0)
    {
        t3 = t2;
        t2 = t1;
        t1 = t3 - t2 * quot;
        s3 = s2;
        s2 = s1;
        s1 = s3 - s2 * quot;
        a = b;
        b = rem;
        rem = a % b;
        quot = a / b;
    }
    return new int[] {b, s1, t1};
}
```

With $d$ as a linear combination of $a$ and $b$, $d = am + bn$, we can obtain an initial solution to our diophantine equation by multiplying both sides by $\frac{c}{d}$, yielding initial solutions $x_0 = \frac{mc}{d}$ and $y_0 = \frac{nc}{d}$. Then we can enuterate all solutions to the diophantine equation with

$$c = a\left(x_0 + t \cdot \frac{b}{d}\right) + b\left(y_0 - t \cdot \frac{a}{d}\right) \qquad \forall\, t \in \mathbb{Z}$$

Listing 4: Obtaining solutions to a linear diophantine equation

```java
/* Returns an array enumerating the solutions to the dipohantine
 * equation ax + by = c, or null if there are no solutions.
 * for all integers t,
 * a * (dioph[0] + t * dioph[1]) + b * (dioph[2] - t * dioph[3]) = c */
public static int[] dioph(int a, int b, int c)
{
    int[] bez = bezout(a, b);
    int d = bez[0];
    if (c % d != 0)
        return null;
    int x0, y0;
    int[] dioph = new int[4];
    dioph[0] = bez[1] * (c / d); // x0
    dioph[1] = b / d;
    dioph[2] = bez[2] * (c / d); // y0
    dioph[3] = a / d;
    return dioph;
}
```

## 1.3 Congruencies

Given the equation $ax \equiv b \mod n$, we obtain the equivalent diophantine equation $ax - ny = b$, which can be solved as above. Note that this has a solution only when $\gcd(a, n) \mid b$.
The inverse of an integer $a$ mod $n$ (with $\gcd(a, n) = 1$) is a solution to the conguency $ax \equiv 1 \mod n$. This can be found by solving the corresponding diophantine equation:

Listing 5: Obtaining the inverse of $a$ mod $n$

```
/* Returns the inverse of a mod n, or 0 if no such inverse exists */
public static int inverse(int a, int n)
{
    int[] dioph = dioph(a, n, 1);
    if (dioph == null)
        return 0;
    int inv = dioph[0] % dioph[1];
    if (inv < 0)
        inv += dioph[1];
    return inv;
}
```

- Wilson's Thereom: For all positive integers $n > 1$, $(n-1)! \equiv -1 \mod n$ if and only if $n$ is prime.
- Euler's Theorem: For all positive integers $a$ and $n$, $a^{\varphi(n)} \equiv 1 \mod n$ if and only if $\gcd(a, n) = 1$, where $\varphi(n)$ is the number of positive intgers less than $n$ that are relatively prime to $n$

## 1.4 Chinese Remainder Theorem

The Chinese Remainder Theorem states that for pairwise relatively prime $n_i$s, the following system always has a solution, and that it is unique mod $N = n_1 n_2 \cdots n_k$:

$$\begin{aligned} x &\equiv a_1 \mod n_1 \\ x &\equiv a_2 \mod n_2 \\ &\vdots \\ x &\equiv a_k \mod n_k \end{aligned}$$

and that the solution is equal to $\sum_{i=1}^{k} a_i y_i N_i$, where $N_i = \frac{N}{n_i}$ and $y_i$ is the inverse of $N_i$ mod $n_i$.

Listing 6: Obtaining a solution to a Chinese Remainder Theorem Problem

```
/* Returns the smallest nonnegative solution to the Chinese Remainder Theorem
 * system of equations, or -1 if no solution exists */
public static int crt(int[] a, int[] n)
{
    int len = a.length;
    if (n.length != len)
        return -1;
    int N = 1;
    for (int i = 0; i < len; i++)
        N *= n[i];
    int sol = 0;
    for (int i = 0; i < len; i++)
    {
        int Nk = N / n[i];
        int yk = inverse(Nk, n[i]);
        if (yk == 0)
            return -1;
        sol += a[i] * yk * Nk;
        sol %= N;
    }
    return sol;
}
```

## 1.5 Prime Numbers

We can generate a list of primes by using the Sieve of Eratosthenes[2]:

Listing 7: Obtaining a list of prime numbers

```
// returns a list of all primes less than max
public static int[] primelist(int max)
{
    LinkedList<Integer> potential = new LinkedList<Integer>();
    LinkedList<Integer> primes = new LinkedList<Integer>();
    primes.add(2);
    for (int i = 1; i < max / 2; i++)
        potential.add(2 * i + 1);
    int cur;
    ListIterator<Integer> li;
    while (!potential.isEmpty())
    {
        cur = potential.poll();
        primes.add(cur);
        li = potential.listIterator(0);
        while (li.hasNext())
        {
            if (li.next() % cur == 0)
                li.remove();
        }
    }
    int[] list = new int[primes.size()];
    li = primes.listIterator(0);
    int k = 0;
    while (li.hasNext())
        list[k++] = li.next();
    return list;
}
```

- Euclid's Lemma: If $p$ is a prime and $p|ab$, then $p|a$ or $p|b$.
- Dirichlet's Theorem on Arithmetic Progressions: For any two relatively prime integers $a$ and $d$, there are infinitely many primes of the form $a + nd$.

The power of a prime $p$ in the factorization of $n!$ is $\sum_{k=1}^{\infty} \left\lfloor \frac{n}{p^k} \right\rfloor$. Note that this is not actually an infinite sum since all remaining terms are zero once $p^k$ exceeds $n$.

Listing 8: Obtaining the power of a prime in $n!$

```
/* returns the power of p in the prime factorization of n!
 * note that this is only valid if p is prime */
public static int powerofp(int p, int n)
{
    int count = 0;
    while (n > 0)
    {
        n /= p;
        count += n;
    }
    return count;
}
```

To find the number of zeroes at the end of $n!$, we need only consider the power of 5 since $10 = 2 \cdot 5$ and the power of 2 always exceeds that of 5.

[2]Begin by creating a list of numbers from 2 up to a desired maximum. At each stage, the smallest number is guaranteed to be prime, and remove all multiples of that number from the list. Once the square root of the maximum is reached, all remaining numbers on the list are also guaranteed to be prime.

# 2 Combinatorics

## 2.1 Binomial Coefficients

Listing 9: Obtaining binomial coefficients using Pascal's Triangle

```
public static int[][] binom(int max)
{
    int[][] coef = new int[max][];
    for (int n = 0; n < max; n++)
    {
        coef[n] = new int[n + 1];
        coef[n][0] = 1;
        coef[n][n] = 1;
    }
    for (int n = 2; n < max; n++)
        for (int k = 1; k < n; k++)
            coef[n][k] = coef[n - 1][k - 1] + coef[n - 1][k];
    return coef;
}
```

The Binomial Theorem states that

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

## 2.2 Generating Functions

$$\frac{1}{1-x} = \sum_{k=0}^{\infty} x^k \qquad \frac{x}{(1-x)^2} = \sum_{k=0}^{\infty} kx^k$$

$$\frac{1}{(1-x)^s} = \sum_{k=0}^{\infty} \binom{k+s-1}{s-1} x^k \qquad e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

## 2.3 Catalan Numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \sum_{i=0}^{n-1} C_i C_{n-i-1} = \{1, 1, 2, 5, 14, 42, 132, \dots\}$$

- number of expressions with $n$ pairs of correctly matched parentheses
- number of ways of associating $n$ applications of a binary operator
- number of lattice paths on an $n \times n$ grid that do not cross the diagonal
- number of ways a convex polygon with $n + 2$ sides can be cut into triangles
- number of ways to connect pairs of $2n$ points on a circle with no chords intersecting
- number of noncrossing partitions of an $n$-element set

## 2.4 Colorings

For a cube:

- cycle index of vertices: $\frac{1}{24}\left(x_1^8 + 6x_4^2 + 9x_2^4 + 8x_1^2x_3^2\right)$
- cycle index of edges: $\frac{1}{24}\left(x_1^{12} + 6x_4^3 + 3x_2^6 + 8x_3^4 + 6x_1^2x_2^5\right)$
- cycle index of faces: $\frac{1}{24}\left(x_1^6 + 6x_2^3 + 8x_3^2 + 3x_1^2x_2^2 + 6x_1^2x_4\right)$

# 3 Geometry

Listing 10: Basic Vector and Line classes

```
public static class Vector
{
    int x, y;
    public double angle()
    {
        return Math.atan2(y, x);
    }

    public double angleTo(Vector v) // returns positive clockwise angle
    {
        double ans = angle() - v.angle();
        if (ans < 0)
            ans += 2 * Math.PI;
        return ans;
    }

    public int mag2()
    {
        return y * y + x * x;
    }
}

public static class Line
{
    double x1, y1, x2, y2;
    public Point intersection(Line ol)
    {
        double x3 = ol.x1, y3 = ol.y1;
        double x4 = ol.x2, y4 = ol.y2;

        double d = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
        if (d == 0)
            return null; // lines are parallel

        double Px = ((x1 * y2 - y1 * x2) * (x3 - x4) - (x1 - x2) * (x3 * y4 - y3 * x4)) / d;
        double Py = ((x1 * y2 - y1 * x2) * (y3 - y4) - (y1 - xy) * (x3 * y4 - y3 * x4)) / d;
        return new Point(Px, Py);
    }
}
```

Listing 11: Finding the area of a polygon with coordinates in two dimensions

```
public static double polygonArea(double[] X, double[] Y)
{
    double area = 0;
    int j = X.length - 1;

    for (int i = 0; i < X.length; i++)
    {
        area += (X[j] + X[i]) * (Y[j] - Y[i]);
        j = i;
    }

    return Math.abs(area / 2);
}
```

Listing 12: Finding the convex hull of a set of points

```java
public static LinkedList<Point> giftWrapping(Point[] points)
{
    int n = points.length;
    LinkedList<Point> hull = new LinkedList<Point>();
    int min, miny;
    min = 0;
    miny = points[0].y
    for (int i = 1; i < n; i++)
    {
        if (points[i].y < miny < 0)
        {
            min = i;
            miny = points[i].y;
        }
    }
    Point first = points[min];
    Point cur = first;
    Vector curvec = new Vector(1, 0);
    while (!cur.equals(first) || hull.isEmpty())
    {
        hull.add(cur);
        double minang = 3 * Math.PI;
        int minmag = 0;
        for (int i = 0; i < n; i++)
        {
            Point p = points[i];
            if (p.equals(cur))
                continue;
            double angle = curvec.angleTo(cur.to(p));
            if ((angle > 0 && angle < minang) ||
                (angle == minang && cur.to(p).mag2() > minmag))
            {
                min = i;
                minang = angle;
                minmag = cur.to(p).mag2();
            }
        }
        curvec = points[min].to(cur);
        cur = points[min];
    }
    return hull;
}
```

Area of a triangle:

$$A = \sqrt{s(s-a)(s-b)(s-c)}, \text{ where } s = \frac{a+b+c}{2} \qquad\qquad A = \frac{1}{2}ab\sin C$$

Area of a regular polygon:

$$A = \frac{nS^2}{2} \cdot \sin\left(\frac{2\pi}{n}\right), \text{ where } S \text{ is the length from center to corner}$$

Triangle identities:

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} \qquad\qquad c^2 = a^2 + b^2 - 2ab\cos C$$

$$\frac{a-b}{a+b} = \frac{\tan\left(\frac{A-B}{2}\right)}{\tan\left(\frac{A+B}{2}\right)}$$

# 4 General Algorithms

Listing 13: Performing a floodfill

```
public static void floodfill(int[][] array, Point node, int target, int replacement)
{
    if (array[node.y][node.x] != target)
        return;
    Queue<Point> Q = new LinkedList<Point>();
    Q.add(node);
    while (!Q.isEmpty())
    {
        Point n = Q.poll();
        if (array[n.y][n.x] == target)
        {
            int w, e;
            w = n.x;
            e = n.x;
            while (w > 0 && array[n.y][w - 1] == target)
                w -= 1;
            while (e < array[n.y].length - 1 && array[n.y][e + 1] == target)
                e += 1;
            for (int i = w; i <= e; i++)
            {
                array[n.y][i] = replacement;
                if (n.y > 0 && array[n.y - 1][i] == target)
                    Q.add(new Point(i, n.y - 1));
                if (n.y < array.length - 1 && array[n.y + 1][i] == target)
                    Q.add(new Point(i, n.y + 1));
            }
        }
    }
}
```

Listing 14: Obtaining the next permutation of an array

```
public static void swap(int[] array, int i, int j)
{
    int v = array[i];
    array[i] = array[j];
    array[j] = v;
}

public static void reverse(int[] array, int start)
{
    int len = array.length - start;
    for (int i = 0; i < len / 2; i++)
    {
        swap(array, start + i, array.length - i - 1);
    }
}

public static boolean nextPermutation(int[] array)
{
    int k = array.length - 2;
    while (k >= 0 && array[k] >= array[k + 1])
        k--;
    if (k < 0)
        return false;
    int l = array.length - 1;
    while (array[k] >= array[l])
        l--;
    swap(array, k, l);
    reverse(array, k + 1);
    return true;
}
```

Listing 15: Finding the longest increasing subsequence of a sequence

```
    public static int longestIncreasingSubsequence(int[] values)
    {
        int n = values.length;
        int[] best = new int[n];
//      int[] prev = new int[n];
        best[0] = 1;
//      prev[0] = -1;
        for (int i = 1; i < n; i++)
        {
            int cur = values[i];
            best[i] = 0;
            for (int j = 0; j < i; j++)
            {
                if (cur > values[j])
                {
                    if (best[j] > best[i])
                    {
                        best[i] = best[j];
//                      prev[i] = j;
                    }
                }
            }
            best[i]++;
        }
        int maxind = 0;
        for (int j = 0; j < n; j++)
            if (best[j] > best[maxind])
                maxind = j;
        return best[maxind];
//      int[] path = new int[best[maxind]];
//      path[path.length - 1] = maxind;
//      for (int i = path.length - 2; i >= 0; i--)
//          path[i] = prev[i + 1];
//      return path;
    }
```

Listing 16: Finding the longest common subsequence

```
import static java.lang.Math.max;

    public static int longestCommonSubsequence(char[] s1, char[] s2)
    {
        int m = s1.length, n = s2.length;
        int maxlen[][] = new int[m + 1][n + 1];
        for (int j = 0; j <= m; j++)
            maxlen[j][0] = 0;
        for (int i = 0; i <= n; i++)
            maxlen[0][i] = 0;
        for (int j = 0; j < m; j++)
        {
            for (int i = 0; i < n; i++)
            {
                if (s1[j] == s2[i])
                    maxlen[j + 1][i + 1] = maxlen[j][i] + 1;
                else
                    maxlen[j + 1][i + 1] = max(maxlen[j + 1][i], maxlen[j][i + 1]);
            }
        }
        return maxlen[m][n];
    }
```

Listing 17: Finding the longest common substring

```java
import static java.lang.Math.max;

    public static int longestCommonSubstring(char[] s1, char[] s2)
    {
        int m = s1.length, n = s2.length;
        int maxlen[][] = new int[m + 1][n + 1];
        int best = 0, cur;
        for (int j = 0; j <= m; j++)
            maxlen[j][0] = 0;
        for (int i = 0; i <= n; i++)
            maxlen[0][i] = 0;
        for (int j = 0; j < m; j++)
        {
            for (int i = 0; i < n; i++)
            {
                if (s1[j] == s2[i])
                    cur = maxlen[j][i] + 1;
                else
                    cur = 0;
                maxlen[j + 1][i + 1] = cur;
                best = max(best, cur);
            }
        }
        return best;
    }
```

Listing 18: Finding the maximum contiguous subarray

```java
import static java.lang.Math.max;

    public static int maximumSubarray(int[] values)
    {
        int maxHere = 0;
        int totalMax = 0;
        for (int x : values)
        {
            maxHere = max(0, maxHere + x);
            totalMax = max(totalMax, maxHere);
        }
        return totalMax;
    }
```

Listing 19: Counting the number of ways to make change (order doesn't matter)

```java
    public static long[] getUnorderedWays(int[] values, int max)
    {
        long[][] ways = new long[values.length][max + 1];
        for (int i = 0; i < values.length; i++)
        {
            int cur = values[i];
            ways[i][0] = 1;
            for (int j = 1; j <= max; j++)
            {
                int prev = j - cur;
                ways[i][j] = 0;
                if (prev >= 0)
                    ways[i][j] += ways[i][prev];
                if (i > 0)
                    ways[i][j] += ways[i - 1][j];
            }
        }
        return ways[values.length - 1];
    }
```

Listing 20: Counting the number of ways to make change (order matters)

```java
    public static long[] getOrderedWays(int[] values, int max)
    {
        long ways[] = new long[max + 1];
        int cur;
        ways[0] = 1;
        for (int j = 1; j <= max; j++)
        {
            cur = 0;
            for (int i = 0; i < values.length; i++)
            {
                int prev = j - values[i];
                if (prev >= 0)
                    cur += ways[prev];
            }
            ways[j] = cur;
        }
        return ways;
    }
```

## 5 Graph Theory

Listing 21: Basic code in Graph class

```java
public class Graph
{
    private int[][] weights;
    private boolean[][] conn;

    public Graph(int n)
    {
        weights = new int[n][n];
        conn = new boolean[n][n];
        for (int i = 0; i < n; i++)
        {
            Arrays.fill(weights[i], -1);
            Arrays.fill(conn[i], false);
        }
    }

    public void addEdge(int source, int dest, int weight)
    {
        conn[source][dest] = true;
        weights[source][dest] = weight;
        conn[dest][source] = true;
        weights[dest][source] = weight;
    }

    public void removeEdge(int source, int dest)
    {
        conn[source][dest] = false;
        conn[dest][source] = false;
    }

    public int size()
    {
        return conn.length;
    }
}
```

## 5.1 Shortest Path

Listing 22: Dijkstra's algorithm for finding shortest path between two vertices

```
// finds shortest path from source to dest
public int dijkstra(int source, int dest)
{
    boolean[] visited = new boolean[size()];
    int[] value = new int[size()];
    int[] prev = new int[size()];
    int[] dists = new int[size()];
    Arrays.fill(visited, false);
    Arrays.fill(value, Integer.MAX_VALUE);
    value[source] = 0;
    prev[source] = -1;
    dists[source] = 1;
    int current = source;

    while (!visited[dest])
    {
        for (int i = 0; i < size(); i++)
        {
            if (visited[i] || i == current)
                continue;
            if (!conn[current][i])
                continue;

            int dist = value[current] + weights[current][i];
            if (dist < value[i])
            {
                value[i] = dist;
                prev[i] = current;
                dists[i] = dists[current] + 1;
            }
        }

        visited[current] = true;
        current = dest;
        for (int i = 0; i < size(); i++)
        {
            if (!visited[i] && value[i] < value[current])
                current = i;
        }
    }

    int[] path = new int[dists[dest]];
    current = dest;
    for (int i = dists[dest] - 1; i >= 0; i--)
    {
        path[i] = prev[current];
        current = prev[current];
    }

    // path contains the path
    // return path;

    // value is the distance
    return value[dest];
}
```

## 5.2 Minimal Spanning Tree

Listing 23: Kruskal's algorithm for finding minimal spanning tree

```
    private static class Edge implements Comparable<Edge>
    {
        public int source, dest, weight;

        public Edge(int source, int dest, int weight)
        {
            this.source = source;
            this.dest = dest;
            this.weight = weight;
        }

        @Override
        public int compareTo(Edge arg0)
        {
            return weight - arg0.weight;
        }
    }

    // Ax + B transformation of weights
    // finds minimum-cost spanning tree
    public Graph kruskal(int A, int B)
    {
        Graph g = new Graph(size());
        PriorityQueue<Edge> Q = new PriorityQueue<Edge>();
        boolean[][] connected = new boolean[size()][size()];
        for (int i = 0; i < size(); i++)
        {
            Arrays.fill(connected[i], false);
            connected[i][i] = true;
            for (int j = 0; j < i; j++)
            {
                if (conn[i][j])
                    Q.add(new Edge(i, j, A * weights[i][j] + B));
            }
        }

        while (!Q.isEmpty())
        {
            Edge e = Q.poll();
            if (!connected[e.source][e.dest])
            {
                g.addEdge(e.source, e.dest, e.weight);
                for (int i = 0; i < size(); i++)
                {
                    if (connected[e.source][i] || connected[e.dest][i])
                    {
                        connected[e.source][i] = true;
                        connected[i][e.source] = true;
                        connected[i][e.dest] = true;
                        connected[e.dest][i] = true;
                    }
                }
            }
        }

        return g;
    }
```