Discrete Structures for Part IA Discrete Mathematics

Note Discrete structures are only briefly touched upon in the course. This handout gives a bit more depth in some of the definitions covered and in the end an example in Java is presented on how discrete structures can be used in algorithm design.

The Part II Cryptography course covers these in much greater depth. Most textbooks on *group theory* or *abstract algebra* should cover the topics here in sufficient depth.

Basic Group Theory

Definition 1. Let S be a set and $\cdot : S \times S \to S$, then $(S, \cdot)$ is a group if it satisfies the following axioms:

1. (**Closure**) For all $a, b \in S$, the result $a \cdot b \in S$ (Note: this is guaranteed by the definition of $\cdot$)
2. (**Associativity**) For all $a, b, c \in S$, $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
3. (**Identity element**) There exists $e \in S$ such that $a \cdot e = e \cdot a = a$ for all $a \in S$.
4. (**Inverse element**) For every $a \in S$, there exists $a^{-1} \in S$ such that $a \cdot a^{-1} = a^{-1} \cdot a = e$.

Example 1 (Identity is unique)**.** Show that the identity element in a group is unique.

Proof. Assume that there are two identities e and e', then using the definition of an identity for e gives:

$$e \cdot e' = e' \cdot e = e \tag{1}$$

The definition of an identity for e' gives:

$$e' \cdot e = e \cdot e' = e' \tag{2}$$

Combining (1) and (2), we get $e = e'$. □

Example 2 (Inverses are unique)**.** Show that the inverse for each element in a group are unique.

Proof. Assume that element $a \in G$ has two inverses a_1 and a_2. By definition of a_1 being an inverse of a, we have

$$a \cdot a_1 = e \tag{3}$$

By definition of a_2 being an inverse of a,

$$a \cdot a_2 = e \tag{4}$$

Combining 3 and 4, gives

$a \cdot a_1 = a \cdot a_2 \Rightarrow a_1 \cdot (a \cdot a_1) = a_1 \cdot (a \cdot a_2) \Rightarrow (a_1 \cdot a) \cdot a_1 = (a_1 \cdot a) \cdot a_2 \Rightarrow e \cdot a_1 = e \cdot a_2 \Rightarrow a_1 = a_2$ □

Example 3 (Inverse of the inverse)**.** Let $(G, \cdot)$ be a group, then

1. $(a^{-1})^{-1} = a$
2. $(ab)^{-1} = b^{-1}a^{-1}$

Proof.(1) By definition of $(a^{-1})^{-1}$ being the inverse of a^{-1}, it satisfies

$a^{-1} \cdot (a^{-1})^{-1} = (a^{-1})^{-1} \cdot a^{-1} = e$

But these are also satisfied for by a (since a^{-1} is its inverse). By the uniqueness of inverses, it follows that $a = (a^{-1})^{-1}$.

(2) $a \cdot b \cdot (b^{-1} \cdot a^{-1}) = a \cdot (b \cdot b^{-1}) \cdot a^{-1} = a \cdot (e) \cdot a^{-1} = a \cdot (e \cdot a^{-1}) = a \cdot a^{-1} = e$ □

Example 4. Let $(G, \cdot)$ be a group. For all $a, b \in G$, if $a \cdot b = e$, then $a = b^{-1}$ and $b = a^{-1}$.

Proof.$a \cdot b = e \Rightarrow a^{-1} \cdot a \cdot b = a^{-1} \cdot e \Rightarrow e \cdot b = a^{-1} \Rightarrow b = a^{-1}$

Similarly,

$b \cdot a = e \Rightarrow b^{-1} \cdot b \cdot a = b^{-1} \cdot e \Rightarrow e \cdot a = b^{-1} \Rightarrow a = b^{-1}$ □

Example 5. Let $(G, \cdot)$ be a group. If $a^2 = a$, then $a = e$.

Proof.

$$\begin{aligned} a^2 = a &\Rightarrow a^{-1} \cdot (a \cdot a) = a^{-1} \cdot a \text{ (By right multiplying by } a^{-1}) \\ &\Rightarrow (a^{-1} \cdot a) \cdot a = a^{-1} \cdot a \text{ (By associativity)} \\ &\Rightarrow e \cdot a = e \text{ (By inverse properties)} \\ &\Rightarrow a = e \text{ (By properties of the identity)} \end{aligned}$$

□

Example 6 (Cancellation Law)**.** Let $(G, \cdot)$ be a group. For $a, b, c \in G$, if $a \cdot b = a \cdot c$ or $b \cdot a = c \cdot a$, then $b = c$.

*Proof.*Consider the inverse a^{-1} of a, then

$$\begin{aligned} a \cdot b &= a \cdot c \Rightarrow \\ a^{-1} \cdot (a \cdot b) &= a^{-1} \cdot (a \cdot c) \text{ (by left multiplication)} \\ (a^{-1} \cdot a) \cdot b &= (a^{-1} \cdot a) \cdot c \text{ (by associativity)} \\ e \cdot b &= e \cdot c \text{ (by inverse property)} \\ b &= c \text{ (by inverse property)} \end{aligned}$$

By using right multiplication, we also prove the second result. □

Exercise 1 (Subgroup)**.** A subset $H \subseteq G$ of a group $(G, \cdot)$ is a subgroup iff $(H, \cdot)$ is a group. Show that $(H, \cdot)$ being a subgroup of $(G, \cdot)$ is equivalent to $e_G \in H$ and $\forall a, b \in H a \cdot b^{-1} \in H$.

Note: Strictly speaking $\cdot$ is the operator constrained on H when referred to the group $(H, \cdot)$.

Exercise 2 (Exponents in groups)**.** Let $(G, \cdot)$ be a group and $a \in G$. For convenience,

- for $n \in \mathbb{Z}^+$, we write $a^n = \underbrace{a \cdot \ldots \cdot a}_{n \text{ terms}}$
- for $n \in \mathbb{Z}^+$, we write $a^{-n} = \underbrace{a^{-1} \cdot \ldots \cdot a^{-1}}_{n \text{ terms}}$, where a^{-1} is the inverse of a in $(G, \cdot)$.
- for $n = 0$, we write $a^0 = e$, where e is the identity of the group.

Prove that:

(a) $(a^{-1})^m = a^{-m} = (a^m)^{-1}$

(b) $a^{n+m} = a^n \cdot a^m$

(c) $(a^n)^m = a^{n \cdot m}$

Exercise 3 (Cyclic groups)**.** : A group is **cyclic** if $\exists a. \forall b. \exists n \in \mathbb{Z}. b = a^n$. The element a is called a **generator** of G.

Given a group $(G, \cdot)$ and an element $a \in G$, we define $\langle a \rangle = \{a^n : n \in \mathbb{Z}\}$. Show that $\langle a \rangle$ forms a subgroup of $(G, \cdot)$.

Abelian groups

Definition 2. A group $(G, \cdot)$ is Abelian if $\cdot$ is commutative, i.e. for every $a, b \in G$, $a \cdot b = b \cdot a$.

Exercise 4. Let $(G, \cdot)$ be a group. Show that if for every $a \in G$, $a^2 = e$ (where e is the identity), then the group operator is commutative.

Exercise 5. Let $(G, \cdot)$ be a group. If for all $a, b \in G$ we have that $(a \cdot b)^2 = a^2 \cdot b^2$ then $(G, \cdot)$ is an abelian group.

Exercise 6. Let $(G, \cdot)$ be a group. Then G is an abelian group if and only if for all $a, b \in G$ we have that $(a \cdot b)^{-1} = a^{-1} b^{-1}$.

Structures which are less specified than groups

For completeness we give the structures that are less specified than groups. You may find it hard to remember the names. In the next section we show how to create an exponentiation algorithm which becomes incrementally more powerful for each of these structures.

Definition 3. Let S be a set and $\cdot : S \times S \to S$, then $(S, \cdot)$ is a magma if it satisfies the following axioms:

1. (**Closure**) For all $a, b \in S$, the result $a \cdot b \in S$ (Note: this is guaranteed by the definition of $\cdot$)

Definition 4. Let S be a set and $\cdot : S \times S \to S$, then $(S, \cdot)$ is a semigroup if it satisfies the following axioms:

1. (**Closure**) For all $a, b \in S$, the result $a \cdot b \in S$ (Note: this is guaranteed by the definition of $\cdot$)
2. (**Associativity**) For all $a, b, c \in S$, $a \cdot (b \cdot c) = (a \cdot b) \cdot c$

Definition 5. Let S be a set and $\cdot : S \times S \to S$, then $(S, \cdot)$ is a monoid if it satisfies the following axioms:

1. (**Closure**) For all $a, b \in S$, the result $a \cdot b \in S$ (Note: this is guaranteed by the definition of $\cdot$)
2. (**Associativity**) For all $a, b, c \in S$, $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
3. (**Identity element**) There exists $e \in S$ such that $a \cdot e = e \cdot a = a$ for all $a \in S$.

Rings

Definition 6. A ring is a set R equipped with a two binary operations $+ : R \times R \to R$ and $\cdot : R \times R \to R$, which satisfy the following three sets of axioms:

1. $(R, +)$ is an Abelian group, meaning that:
   - $(a + b) + c = a + (b + c)$
   - $a + b = b + a$
   - There exists $0 \in R$, such that $a + 0 = 0 + a = a$ for all $a \in R$ (**additive identity**)
   - For each $a \in R$, there exists $a^{-1} \in R$ such that $a \cdot a^{-1} = a^{-1}a = e$.
2. $(R, \cdot)$ is a monoid.
   - $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
   - There exists $1 \in R$ such that $a \cdot 1 = 1 \cdot a = a$ for all $a \in R$
3. Multiplication is distributive with respect to addition, meaning that, for all $a \in R$:
   - $a \cdot (b + c) = a \cdot b + a \cdot c$
   - $(b + c) \cdot a = b \cdot a + c \cdot a$

Property 1. Let $(R, +, \cdot)$ be a ring, then $a \cdot 0 = 0$ for every $a \in R$.

Proof. $a \cdot a = a \cdot (a + 0) = a \cdot a + a \cdot 0 \Rightarrow a \cdot 0 = 0$ □

Property 2. Let $(R, +, \cdot)$ be a ring, then $a \cdot (-b) = -(a \cdot b)$ for every $a, b \in R$.

*Proof.*Using the distributive property and the previous property,
$a \cdot (b + -b) = 0 \Rightarrow a \cdot b + a \cdot (-b) = 0 \Rightarrow -(a \cdot b) = a \cdot (-b)$

□

Fields

Definition 7. A field is is a commutative ring where $0 \neq 1$ and all non-zero elements are invertible.

This is equivalent to the following definition:

Definition 8. A filed is a set R equipped with a two binary operations $+ : R \times R \to R$ and $\cdot : R \times R \to R$, which satisfy the following three sets of axioms:

1. $(R, +)$ is an Abelian group with 0 as the identity
2. $(R \setminus \{0\}, \cdot)$ is an Abelian group.
3. Multiplication distributes over addition.

Exercise 7. Show that the rationals form a field under the normal addition and multiplication.

Number Theoretic Structures

Show that the **additive group** $(\mathbb{Z}_n, +_n)$ is an Abelian group.

Show that the **multiplicative group** $(\mathbb{Z}_n^*, \cdot_n)$ is an Abelian group, where $\mathbb{Z}_n^*$ is the set of naturals $< n$ that are co-prime to n.

Show that for p prime, $(\mathbb{Z}_p, +_p, \cdot_p)$ is a field.

Is this true when n is not prime?

Further reading: You may want to read more here about an abstract-algebraic interpretation of the Chinese Remainder Theorem.

Past papers

COMPUTER SCIENCE TRIPOS Part IA – 2017 – Paper 2

9 Discrete Mathematics (MPF)

(*b*) Recall that a commutative monoid is a structure $(M, \mathtt{1}, *)$ where M is a set, $\mathtt{1}$ is an element of M, and $*$ is a binary operation on M such that

$$x * \mathtt{1} = x \ , \quad x * y = y * x \ , \quad (x * y) * z = x * (y * z)$$

for all x, y, z in M.

For a commutative monoid $(M, \mathtt{1}, *)$, consider the structure $(\mathcal{P}(M), I, \circledast)$ where $\mathcal{P}(M)$ is the powerset of M, I in $\mathcal{P}(M)$ is the singleton set $\{\mathtt{1}\}$, and $\circledast$ is the binary operation on $\mathcal{P}(M)$ given by

$$X \circledast Y = \{\, m \in M \mid \exists\, x \in X.\ \exists\, y \in Y.\ m = x * y \,\}$$

for all X and Y in $\mathcal{P}(M)$.

Prove that $(\mathcal{P}(M), I, \circledast)$ is a commutative monoid. [10 marks]

(optional) Discrete structures in algorithm design

Exponentiation `npow`

In the following sections we show how to create an incrementally more powerful algorithm for efficient exponentiation in a discrete structure. You may find it more helpful to see the code directly starting with "Main.java" and going through the examples, and use this document only for clarifications.

Interfaces for structures

We define the Magma interface as:

```
public interface Magma<T> {

        /* The operation. */
        T times(T a, T b);

}
```

The Semigroup interface as:

```
public interface Semigroup<T> extends Magma<T> {

}
```

There is no way to ensure at compile time that the operation is associative, so the Semigroup interface is just directly inheriting Magma. The name just signifies that the overloaded operation is expected to be associative.

The Monoid interface adds a function that returns the identity element:

```
public interface Monoid<T> extends Semigroup<T> {

	/* Returns the identity of the structure. */
	T identity();

}
```

The Group interface adds a function that returns the inverse:

```
public interface Group<T> extends Monoid<T> {

	/* Returns the inverse of an element. */
	T inverse(T a);

}
```

Example structures

We now introduce the example structures that we are going to use. You can verify yourself whether these are magmas, semigroups, monoids or groups.

String pairing magma: This is an artificial structure, so that the operations are not associative. Given two strings "abc" and "def", the result of the operation is "(abc, def)". The result is still a string, so the operation is closed.

```
public class StringPairingMagma implements Magma<String> {

	@Override
	public String times(String a, String b) {
		return "(" + a + "," + b + ")";
	}

}
```

Even integers with multiplication: Note that this structure does not have an identity and that multiplication is still closed, so it is a semigroup.

```
public class EvenIntegerNumber {

	public final int x;

	public EvenIntegerNumber(int x) {
		if (x % 2 == 1) {
			throw new IllegalArgumentException(x + " is an odd number.");
		}
		this.x = x;
	}

	@Override
	public String toString() {
		return Integer.toString(x);
	}

}
public class EvenIntegersWithMultiplications implements Semigroup<EvenIntegerNumber> {

	@Override
	public EvenIntegerNumber times(EvenIntegerNumber a, EvenIntegerNumber b) {
		return new EvenIntegerNumber(a.x * b.x);
	}

}
```

Integers with multiplication: This is a monoid.

```
public class IntegerNumber {

	public final int x;
```

```
        public IntegerNumber(int x) {
                this.x = x;
        }

        @Override
        public String toString() {
                return Integer.toString(x);
        }

}
public class IntegerNumbersWithMultiplication implements Monoid<IntegerNumber> {

        @Override
        public IntegerNumber times(IntegerNumber a, IntegerNumber b) {
                return new IntegerNumber(a.x * b.x);
        }

        @Override
        public IntegerNumber identity() {
                return new IntegerNumber(1);
        }

}
```

Strings with concatenation: This is a monoid.

```
public class StringsWithConcatenation implements Monoid<String> {

        @Override
        public String times(String a, String b) {
                return a + b;
        }

        @Override
        public String identity() {
                return "";
        }

}
```

Rational numbers with multiplication and without zero: This is a group.

```
public class RationalNumbersWithoutZero implements Group<RationalNumber> {

        @Override
        public RationalNumber times(RationalNumber a, RationalNumber b) {
                return RationalNumber.multiply(a, b);
        }

        @Override
        public RationalNumber inverse(RationalNumber a) {
                return a.findInverse();
        }

        @Override
        public RationalNumber identity() {
                return RationalNumber.ONE;
        }

}
```

Integers with addition: This is a group.

```
public class IntegerNumbersWithAddition implements Group<IntegerNumber> {

        @Override
        public IntegerNumber times(IntegerNumber a, IntegerNumber b) {
                return new IntegerNumber(a.x + b.x);
        }

        @Override
```

```
        public IntegerNumber inverse(IntegerNumber a) {
                return new IntegerNumber(-a.x);
        }

        @Override
        public IntegerNumber identity() {
                return new IntegerNumber(0);
        }

}
```

Invertible 2D matrices: This is a group.

```
/*
 * Class for 2D matrices.
 */
public class InvertibleMatrix2d {

        /* The multiplication identity matrix. */
        public static final InvertibleMatrix2d ID =
                        new InvertibleMatrix2d(new double[][]{{1, 0}, {0, 1}});

        private final double m[][];

        public InvertibleMatrix2d(double m[][]) {
                double det = m[0][0] * m[1][1] - m[0][1] * m[1][0];
                if (det == 0.0) {
                        throw new IllegalArgumentException("Given matrix is not invertible.");
                }
                // Ideally, we should be checking dimension here.
                this.m = m;
        }

        /*
         * Multiply two matrices. The product of two invertible matrices
         * is invertible (assuming no loss of precision).
         */
        public static InvertibleMatrix2d multiply(InvertibleMatrix2d a, InvertibleMatrix2d b) {
                double m[][] = new double[2][2];
                m[0][0] = a.m[0][0] * b.m[0][0] + a.m[0][1] * b.m[1][0];
                m[0][1] = a.m[0][0] * b.m[0][1] + a.m[0][1] * b.m[1][1];
                m[1][0] = a.m[1][0] * b.m[0][0] + a.m[1][1] * b.m[1][0];
                m[1][1] = a.m[1][0] * b.m[0][1] + a.m[1][1] * b.m[1][1];
                return new InvertibleMatrix2d(m);
        }

        /*
         * Inverse of a 2D matrix.
         */
        public InvertibleMatrix2d invert() {
                double invM[][] = new double[2][2];
                double det = m[0][0] * m[1][1] - m[0][1] * m[1][0];
                double invDet = 1.0 / (det);
                invM[0][0] = m[1][1] * invDet;
                invM[0][1] = -m[0][1] * invDet;
                invM[1][0] = -m[1][0] * invDet;
                invM[1][1] = m[0][0] * invDet;

                return new InvertibleMatrix2d(invM);
        }

        @Override
        public String toString() {
                return "[ [" + m[0][0] + " " + m[0][1] + "] [" + m[1][0] + " " + m[1][1] + "]]";
        }

}

public class Invertible2dMatrices implements Group<InvertibleMatrix2d> {
```

```
        @Override
        public InvertibleMatrix2d times(InvertibleMatrix2d a, InvertibleMatrix2d b) {
                return InvertibleMatrix2d.multiply(a, b);
        }

        @Override
        public InvertibleMatrix2d inverse(InvertibleMatrix2d a) {
                return a.invert();
        }

        @Override
        public InvertibleMatrix2d identity() {
                return InvertibleMatrix2d.ID;
        }

}
```

`npow` for magmas

For magmas, the only thing we can do for exponentiation a^n for $n \geq 1$, is to compute $a \cdot (a \cdot (\ldots))$. So, an example implementation is as follows:

```
public class MagmaExponentiation<T> {

        private final Magma<T> magma;

        public MagmaExponentiation(Magma<T> magma) {
                this.magma = magma;
        }

        public T exponentiate(T element, int n) {
                if (n <= 0) {
                        throw new IllegalArgumentException("n should be positive.");
                }
                T cur = element;
                for (int i = 1; i < n; ++i) {
                        cur = magma.times(cur, element);
                }
                return cur;
        }

}
```

For the string pairing magma, this gives

```
public static void runStringPairingOperation() {
   MagmaExponentiation<String> fastExp =
         new MagmaExponentiation<>(new StringPairingMagma());
   System.out.println(fastExp.exponentiate("a", 5));
   System.out.println(fastExp.exponentiate("b", 7));
   System.out.println(fastExp.exponentiate("c", 1));
}
/*
String pairing operation:
((((a,a),a),a),a)
((((((b,b),b),b),b),b),b)
c
*/
```

Note that we can also execute this algorithm for any of the other structures.

`npow` for semigroups

For semigroups, we know that we can change the parentheses of the exponentiation, e.g. we can write $\underbrace{a \cdot \ldots \cdot a}_{2n \text{ times}} = \underbrace{(a \cdot a) \cdot \ldots \cdot (a \cdot a)}_{n \text{ times}} = \underbrace{a^2 \cdot \ldots \cdot a^2}_{n \text{ times}}$. So, the code for $n \geq 1$ is as follows:

```
public class SemigroupExponentiation<T> {
```

```
        private final Semigroup<T> semigroup;

        public SemigroupExponentiation(Semigroup<T> semigroup) {
                this.semigroup = semigroup;
        }

        public T exponentiate(T element, int n) {
                if (n <= 0) {
                        throw new IllegalArgumentException("n should be positive.");
                } else if (n == 1) {
                        return element;
                }
                T square = semigroup.times(element, element);
                T rec = exponentiate(square, n / 2);
                if (n % 2 == 0) {
                        return rec;
                }
                return semigroup.times(element, rec);
        }

}
```

For even integers we get:

```
public static void runOddIntgerExponentiation() {
   SemigroupExponentiation<EvenIntegerNumber> fastExp =
         new SemigroupExponentiation<>(new EvenIntegersWithMultiplications());
   System.out.println(fastExp.exponentiate(new EvenIntegerNumber(4), 5));
   System.out.println(fastExp.exponentiate(new EvenIntegerNumber(2), 7));
   System.out.println(fastExp.exponentiate(new EvenIntegerNumber(6), 1));
}
/*
Odd integer number exponentiation:
1024
128
6
*/
```

Note: Thsi algorithm cannot be applied to the string-pairing operation, but the Magma-exponentiation can be applied for semigroups (since semigroups are also magmas).

`npow` for monoids

For monoids, there is also an identity element, so we can assign $a^0 = e$ (if this element is unique).

```
public class MonoidExponentiation<T> {

        private final Monoid<T> monoid;

        public MonoidExponentiation(Monoid<T> monoid) {
                this.monoid = monoid;
        }

        public T exponentiate(T element, int n) {
                if (n < 0) {
                        throw new IllegalArgumentException("n should be non-negative.");
                } else if (n == 0) {
                        return monoid.identity();
                }
                T square = monoid.times(element, element);
                T rec = exponentiate(square, n / 2);
                if (n % 2 == 0) {
                        return rec;
                }
                return monoid.times(element, rec);
        }

}
```

For integers with multiplications, we get

```
public static void runNaturalNumbersExponentiation() {
   MonoidExponentiation<IntegerNumber> fastExp =
         new MonoidExponentiation<>(new IntegerNumbersWithMultiplication());
   System.out.println(fastExp.exponentiate(new IntegerNumber(4), 5));
   System.out.println(fastExp.exponentiate(new IntegerNumber(2), 7));
   System.out.println(fastExp.exponentiate(new IntegerNumber(10), 0));
}
/*
Natural numbers exponentiation:
1024
128
1
*/
```

For string concatenation, we get:

```
public static void runStringConcatenation() {
   MonoidExponentiation<String> fastExp =
         new MonoidExponentiation<>(new StringsWithConcatenation());
   System.out.println(fastExp.exponentiate("abc", 5));
   System.out.println(fastExp.exponentiate("cd", 7));
   System.out.println(fastExp.exponentiate("abc", 0));
}
/*
String concatenation:
abcabcabcabcabc
cdcdcdcdcdcdcd
*/
```

Note: This exponentiation method is not always more efficient. Consider the case for string concatenation.

`npow` for groups

Because of Exercise 2, we can also compute negative powers, so we modify the code as follows:

```
public class GroupExponentiation<T> {

        private final Group<T> group;

        public GroupExponentiation(Group<T> group) {
                this.group = group;
        }

        public T exponentiate(T element, int n) {
                if (n == 0) return group.identity();
                if (n < 0) return exponentiate(group.inverse(element), -n);
                T square = group.times(element, element);
                T rec = exponentiate(square, n / 2);
                if (n % 2 == 0) {
                        return rec;
                }
                return group.times(element, rec);
        }

}
```

For integers with addition, we have:

```
public static void runNaturalNumbersMultiplication() {
   GroupExponentiation<IntegerNumber> fastExp =
         new GroupExponentiation<>(new IntegerNumbersWithAddition());
   System.out.println(fastExp.exponentiate(new IntegerNumber(4), 5));
   System.out.println(fastExp.exponentiate(new IntegerNumber(2), -5));
   System.out.println(fastExp.exponentiate(new IntegerNumber(10), 0));
}
/*
Natural numbers multiplication:
20
-10
0
*/
```

For rational numbers without zero for example, we have:

```
public static void runRationalNumbersExponentiation() {
   GroupExponentiation<RationalNumber> fastExp =
         new GroupExponentiation<>(new RationalNumbersWithoutZero());
   System.out.println(fastExp.exponentiate(new RationalNumber(4, 3), 5));
   System.out.println(fastExp.exponentiate(new RationalNumber(4, 3), -5));
   System.out.println(fastExp.exponentiate(new RationalNumber(4, 3), 0));
}
/*
Rational numbers exponentiation:
1024/243
243/1024
1/1
*/
```

For invertible matrices, we have:

```
public static void runMatrixExponentiation() {
   GroupExponentiation<InvertibleMatrix2d> fastExp =
         new GroupExponentiation<>(new Invertible2dMatrices());
   System.out.println(fastExp.exponentiate(
         new InvertibleMatrix2d(new double[][] {{1, 1}, {1, 0}}), 5));
   System.out.println(fastExp.exponentiate(
         new InvertibleMatrix2d(new double[][] {{1, 1}, {1, 0}}), -5));
   System.out.println(fastExp.exponentiate(
         new InvertibleMatrix2d(new double[][] {{1, 1}, {1, 0}}), 0));
}
/*
Invertible matrices exponentiation:
[ [8.0 5.0] [5.0 3.0]]
[ [-3.0 5.0] [5.0 -8.0]]
[ [1.0 0.0] [0.0 1.0]]
*/
```

Semirings in Shortest Paths (read after Part IA algorithms)

See the attached zip file for how to generalise the Bellman-Ford algorithm to semirings for finding the widest path (i.e. the path with the lightest heaviest edge), the most reliable path (i.e. the path with the highest probability of reaching the destination). Can you think how to create a semiring for finding the k-th shortest path? What if we want to find the widest first and in case there are many equal ones choose the shortest one?