Lab 3: Set operations using bitwise operations

Abdelrahman Elsayed Saeed Saad **22010869**

Abdelrahman Fat'hy Ahmed El-Lawaty **19015920**

1 Part 1

1.1 Requirements

This part focuses on implementing a unit to handle the basic bitwise operations to be used in the set operations later. Here are the required operations:

1. getBit(int number, int position): returns the bit value at position *position*'.
2. setBit(int number, int position): sets the bit at position *position*'
3. clearBit(int number, int position): clears the bit at position *position*.
4. updateBit(int number, int position, boolean value): sets the value 'value' for the bit at position *position*.

1.2 Solution Design

"BitNumOperations" package contains **"BitNum"** data structure that implements all the required operations in addition to some more to be used in set operations.

Data Structures

1. 32-bit primitive integer to hold the decimal value of the BitNum.
2. User-defined BitNum DS to implement all bit operations dynamically.

1.3 Demo Runs

"part1.java" implements a demo run with some test cases assserted at run time as shown in figure.

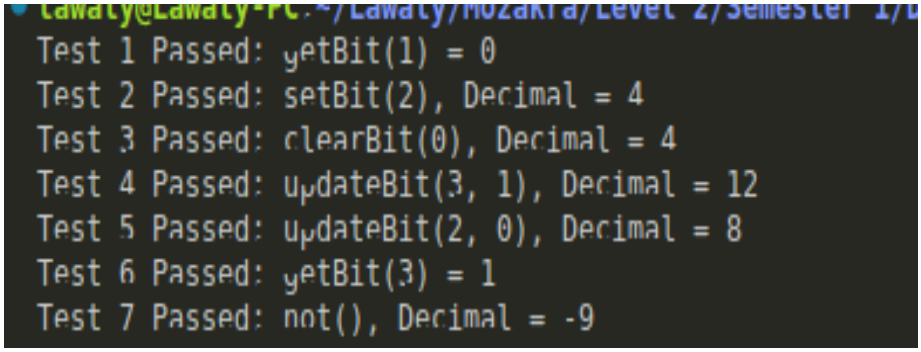

Figure 1: Bitwise Operations Sample Run

2 Part 2

2.1 Requirements

Using the BitNum data structure we have built earlier, it is required to prompt some sets from the user and implement some set operations on them. The universe will first be prompted as a **list of strings** and **use the BitNum data structure to store the elements in each set** as a bit map for it.

2.2 Solution Design

"BitSet" is another data structure that is coupled with the BitNum Class through its interface IbitNum to store the elements as a bitmap. It also stores the "Universe" inside to be able perform set operations like complement and others.

"Universe" is a simple data structure holds all the Sample space elements in an ArrayList. It provides some operation to traverse the universe elements and stuff like that. Here are all the implemented functionalities inside BitSet.

1. add(String element): Adds an element to the set.
2. union(BitSet otherSet): returns a set of all elements inside both sets.
3. intersection(BitSet otherSet): returns a set of elements that exist in both sets.
4. difference(BitSet otherSet): returns a set of all elements inside the first set and not in the other
5. complement(): returns a new set of all elements in the universe that are outside the set
6. cardinality(): returns the number of elements in the set.
7. getElements(): returns a List<string> of all elements in the set.

2.3 Test Runs

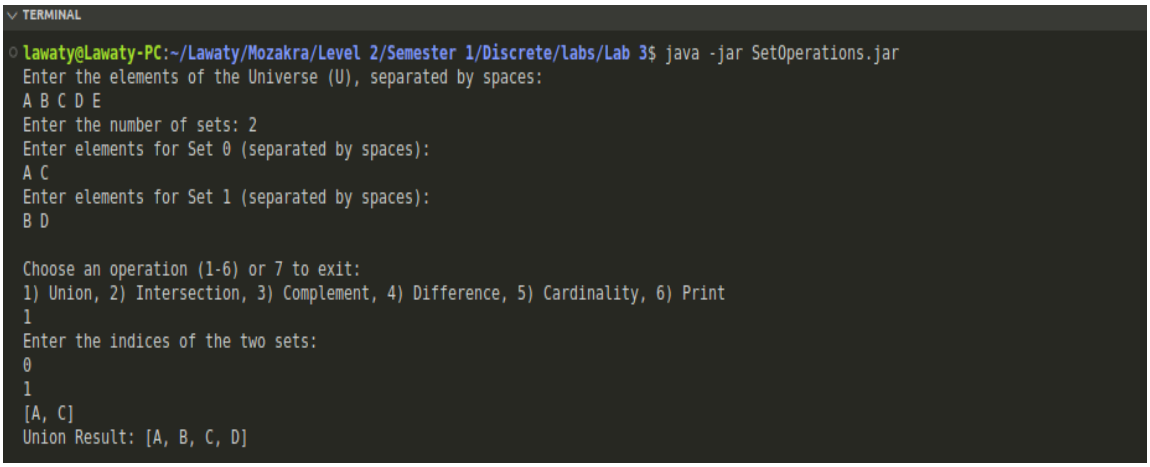

Figure 2: Union

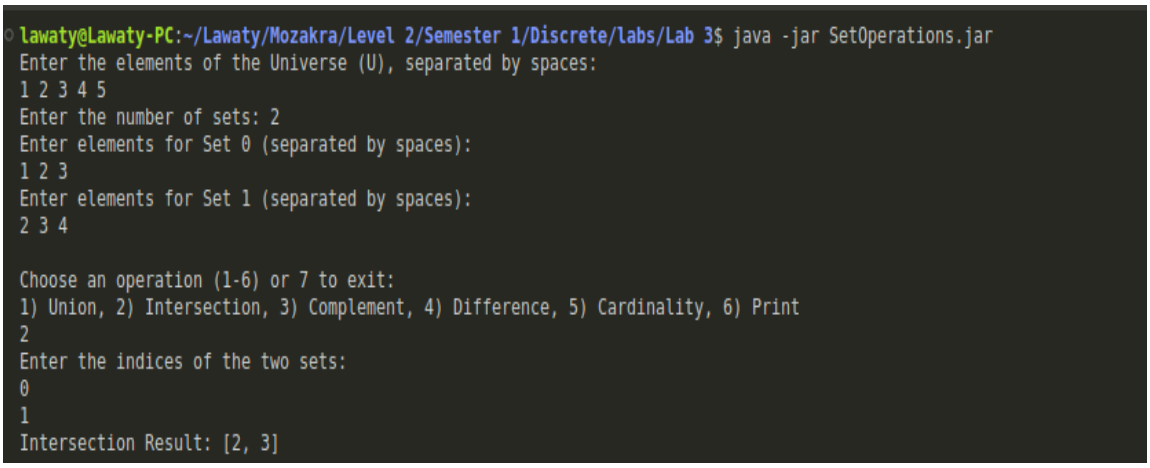

Figure 3: Intersection

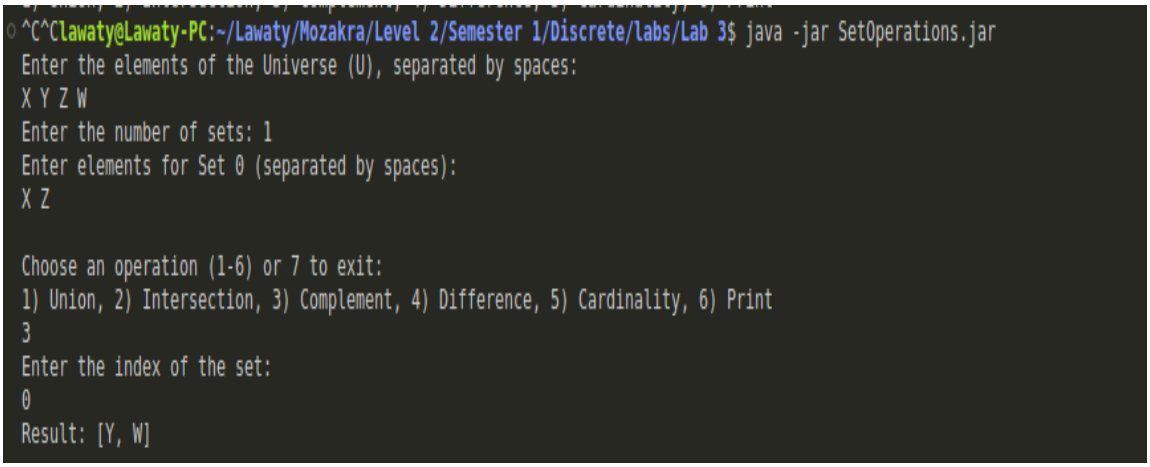

Figure 4: Complement

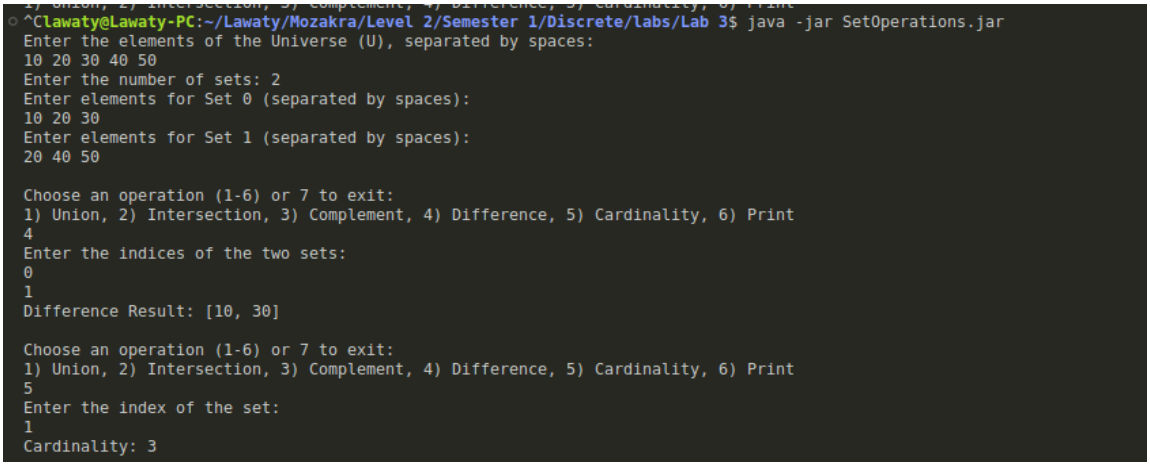

Figure 5: Difference + Cardinality

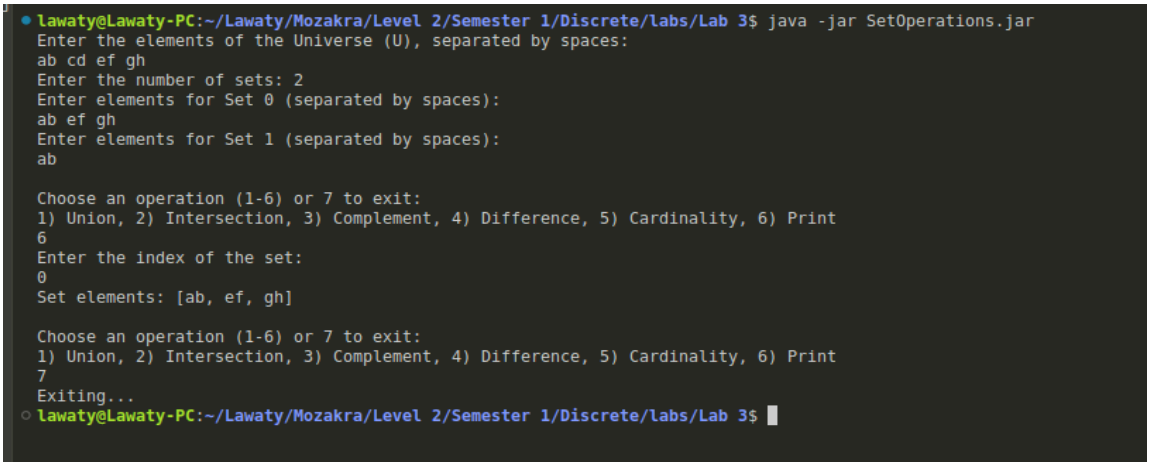

Figure 6: Print + Exit

"Thank You"