ASSIGNMENT 2

Fares Wael 202201260

Introduction:

This report explains the Java program that implements the LZ78 compression and decompression algorithm. The LZ78 algorithm is a dictionary-based lossless data compression technique. The program reads an input file, compresses its contents using LZ78 encoding, and writes the compressed data to an output file. It also provides a decompression function to reconstruct the original data from the compressed file.

Code Structure

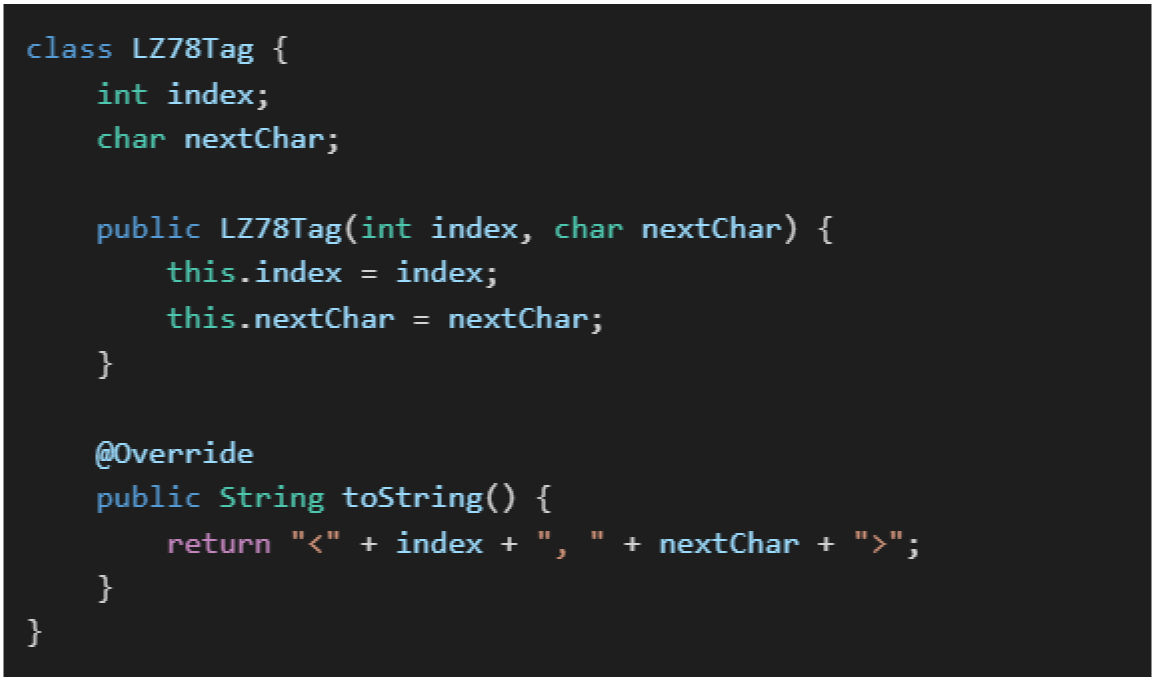
The program consists of the following main components:

- LZ78Tag Class: Represents a single compressed tag containing an index and the next character.
- Compression Method: Reads the input file, processes it into LZ78 tags, and writes the compressed data.
- Decompression Method: Reads the compressed file and reconstructs the original data.
- Main Method: Provides a menu-driven interface for users to choose between compression and decompression.

LZ78Tag class:

This class defines the structure of LZ78 compression tags. Each tag consists of:

- An index: References a previously stored dictionary entry.
- A next character: The next unique character encountered.

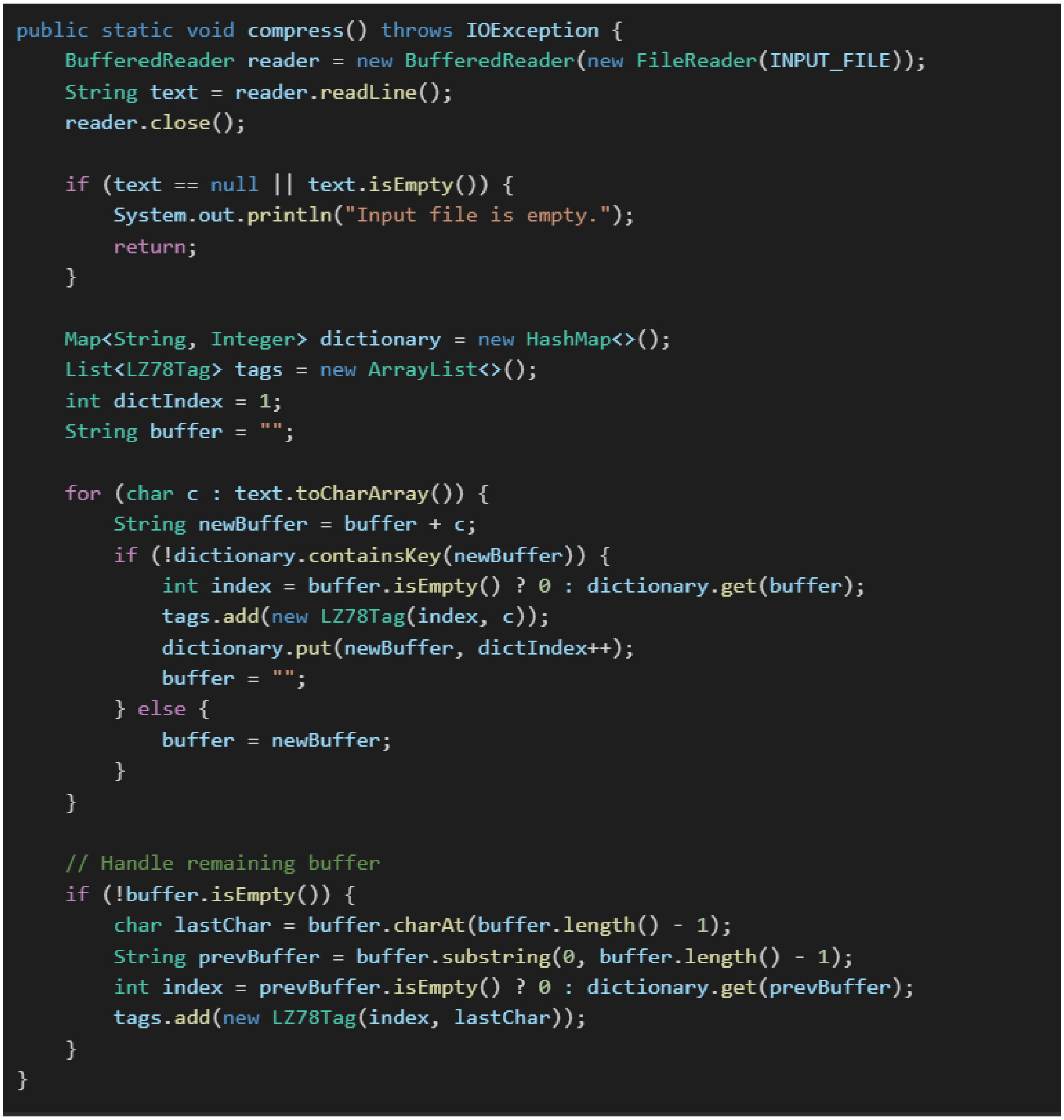


```
class LZ78Tag {
    int index;
    char nextChar;

    public LZ78Tag(int index, char nextChar) {
        this.index = index;
        this.nextChar = nextChar;
    }

    @Override
    public String toString() {
        return "<" + index + ", " + nextChar + ">";
    }
}
```

Compression:

The compress method follows these steps:

- Read the input file.
- Initializes a dictionary to store previously encountered substrings.
- Iterates through the input text, building substrings and assigning indexes.
- Writes the tags to a text file (output_of_compression_tags.txt).
- Calculates the number of bits required for storing indexes.
- Writes compressed binary data to output.bin.
- Writes additional compression calculations to calculations.txt.

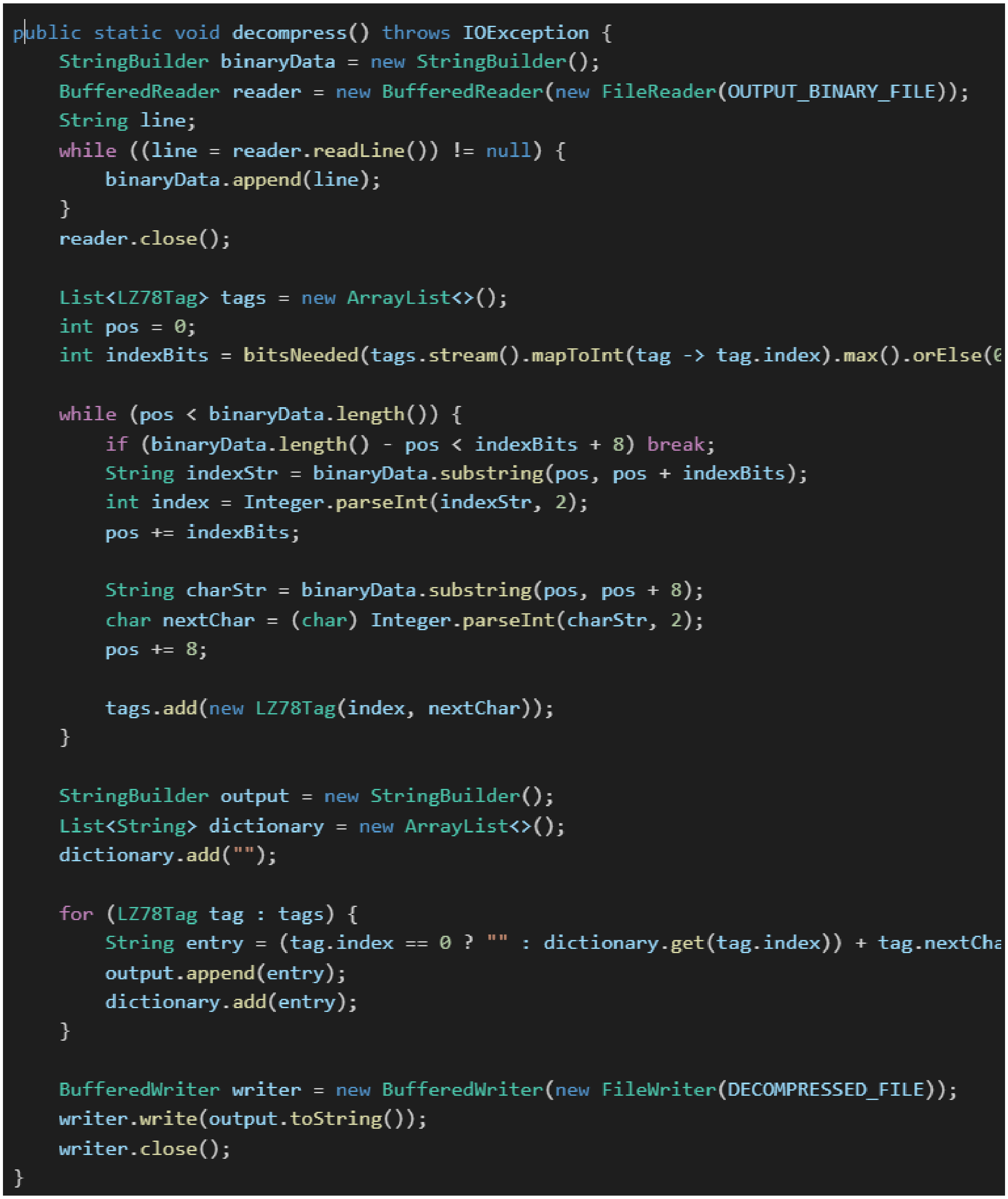


```java
public static void compress() throws IOException {
    BufferedReader reader = new BufferedReader(new FileReader(INPUT_FILE));
    String text = reader.readLine();
    reader.close();

    if (text == null || text.isEmpty()) {
        System.out.println("Input file is empty.");
        return;
    }

    Map<String, Integer> dictionary = new HashMap<>();
    List<LZ78Tag> tags = new ArrayList<>();
    int dictIndex = 1;
    String buffer = "";

    for (char c : text.toCharArray()) {
        String newBuffer = buffer + c;
        if (!dictionary.containsKey(newBuffer)) {
            int index = buffer.isEmpty() ? 0 : dictionary.get(buffer);
            tags.add(new LZ78Tag(index, c));
            dictionary.put(newBuffer, dictIndex++);
            buffer = "";
        } else {
            buffer = newBuffer;
        }
    }

    // Handle remaining buffer
    if (!buffer.isEmpty()) {
        char lastChar = buffer.charAt(buffer.length() - 1);
        String prevBuffer = buffer.substring(0, buffer.length() - 1);
        int index = prevBuffer.isEmpty() ? 0 : dictionary.get(prevBuffer);
        tags.add(new LZ78Tag(index, lastChar));
    }
}
```

Decompression:

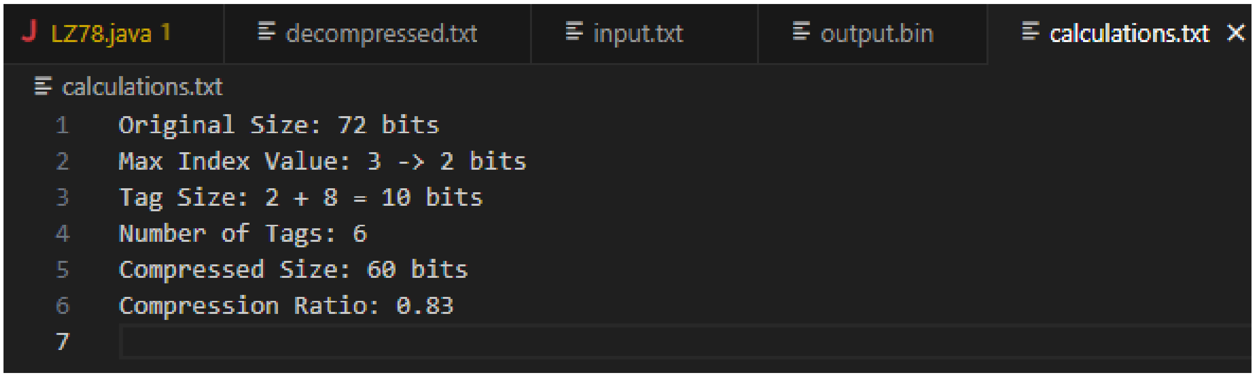
The decompress method follows these steps:

- Reads the binary compressed file (output.bin).
- Extracts the encoded tags.
- Rebuilds the original text using a dictionary.
- Writes the decompressed output to decompressed.txt.



```
public static void decompress() throws IOException {
    StringBuilder binaryData = new StringBuilder();
    BufferedReader reader = new BufferedReader(new FileReader(OUTPUT_BINARY_FILE));
    String line;
    while ((line = reader.readLine()) != null) {
        binaryData.append(line);
    }
    reader.close();

    List<LZ78Tag> tags = new ArrayList<>();
    int pos = 0;
    int indexBits = bitsNeeded(tags.stream().mapToInt(tag -> tag.index).max().orElse(0

    while (pos < binaryData.length()) {
        if (binaryData.length() - pos < indexBits + 8) break;
        String indexStr = binaryData.substring(pos, pos + indexBits);
        int index = Integer.parseInt(indexStr, 2);
        pos += indexBits;

        String charStr = binaryData.substring(pos, pos + 8);
        char nextChar = (char) Integer.parseInt(charStr, 2);
        pos += 8;

        tags.add(new LZ78Tag(index, nextChar));
    }

    StringBuilder output = new StringBuilder();
    List<String> dictionary = new ArrayList<>();
    dictionary.add("");

    for (LZ78Tag tag : tags) {
        String entry = (tag.index == 0 ? "" : dictionary.get(tag.index)) + tag.nextCha
        output.append(entry);
        dictionary.add(entry);
    }

    BufferedWriter writer = new BufferedWriter(new FileWriter(DECOMPRESSED_FILE));
    writer.write(output.toString());
    writer.close();
}
```

Calculations:

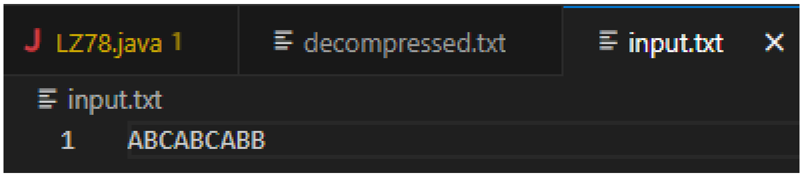


```
Original Size: 72 bits
Max Index Value: 3 -> 2 bits
Tag Size: 2 + 8 = 10 bits
Number of Tags: 6
Compressed Size: 60 bits
Compression Ratio: 0.83
```

Test Cases:

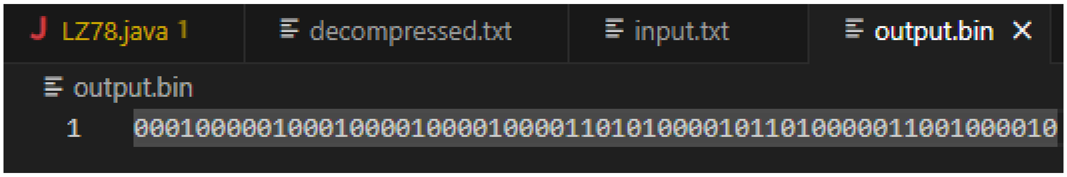
- ## Test case 1:

INPUT:



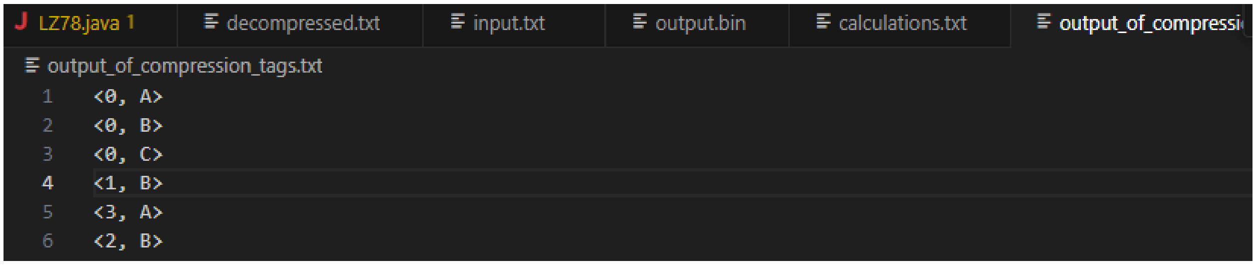
```
ABCABCABB
```

BINARY:



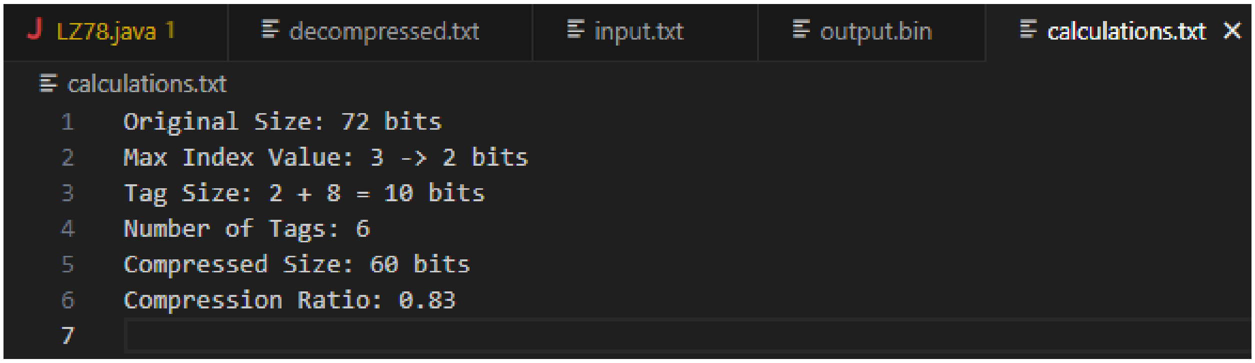
```
000100000100010000100001000011010100001011010000011001000010
```

COMPRESSION TAGS:



```
<0, A>
<0, B>
<0, C>
<1, B>
<3, A>
<2, B>
```

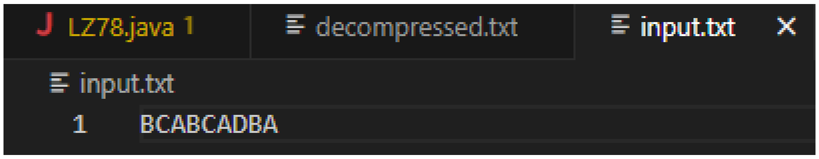
CALCULATIONS:



```
Original Size: 72 bits
Max Index Value: 3 -> 2 bits
Tag Size: 2 + 8 = 10 bits
Number of Tags: 6
Compressed Size: 60 bits
Compression Ratio: 0.83
```

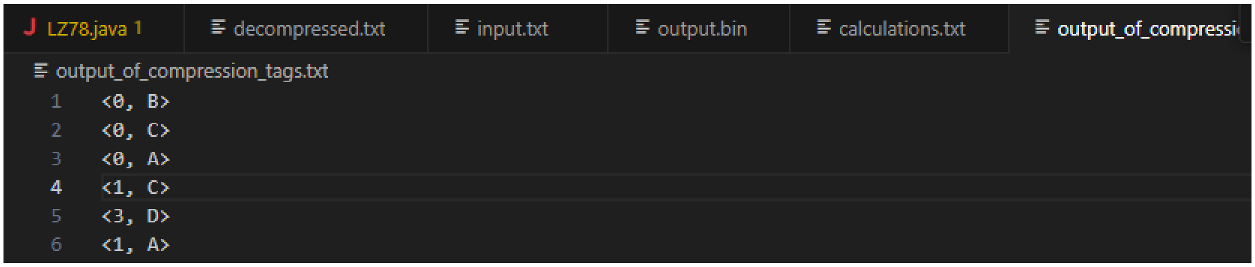
- **Test case 2:**

INPUT:



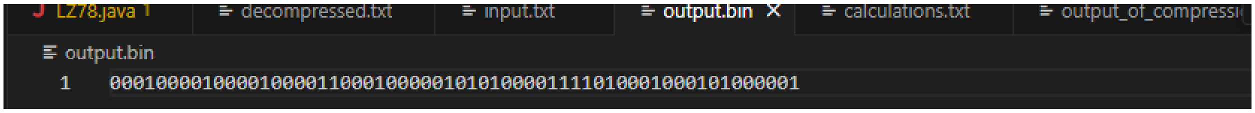
```
BCABCADBA
```

OUTPUT TAGS:



```
<0, B>
<0, C>
<0, A>
<1, C>
<3, D>
<1, A>
```

OUTPUT IN BINARY:



```
000100001000010000110001000001010100001111010001000101000001
```