HW #4- Convolutional codes

Responsible T.A: Yair Mazal, mazaly@post.bgu.ac.il
Submission date: 17/6/21 (last day of semester). In order to accommodate with possible requests for extension, students may submit their work late, without any penalty until July 4th.

Intro

Convolutional codes are used as building blocks of some practical error correction codes used for instance in cellular and Wi-Fi communications (they use turbo codes which are based on convolutional codes). They are also used in satellite communications.
In this homework you will implement this error correction scheme in a class called "ConvolutionalCode". You will use the implementation to encode and decode a byte stream.

This brief will not go into the details of convolutional codes and will not teach you how to implement them. Instead, here are several sources which you are advised to go over. Note that they cover the same topics, and therefore there is no need to go over all of them, choose the ones you're most comfortable with. Alternatively, there are many other online sources.

What to read about:

1. Convolutional codes in general.
2. Encoding a bit stream.
3. Decoding a bit stream using the Viterbi algorithm (hard decisions)

What not to read about (you're welcome to, but out of the scope of this work):

1. Recursive convolutional codes.
2. Soft decision decoding (we will do only hard decisions).
3. BCJR algorithm or other decoding algorithms (we do only Viterbi).

Reading sources:

- Wiki
- Intro and encoding – Medium blog
- Decoding – Medium blog
- Intro and encoding – MIT notes
- Decoding – MIT notes

For those who prefer YouTube:

- Intro and encoding
- Decoding

What will you do?

You will implement a class using the skeleton file provided. The class must implement the constructor and required methods. You may add methods and classes as you see fit.
Required methods:

1. **Constructor** – The constructor receives a single argument, a tuple. Each element in the tuple will be an integer representing a single generator polynomial of the code. To clarify, the integer (in binary form) represents the coefficients of delays. For instance, an input of 5=b101, corresponds to the polynomial $g(x) = 1 \cdot x^2 + 0 \cdot x + 1 = x^2 + 1$, and 3=b11, corresponds to the polynomial $g(x) = 1 \cdot x + 1 = x + 1$.
2. **Encode** – The method receives a bytes object as input and returns its encoded version as output (a list of 0's and 1's). Don't forget the zero tailed termination (details below).
3. **Decode** – The method receives a list of 0's and 1's (bits) as input and returns its decoded bytes (a bytes object) as output using the Viterbi algorithm. It also returns the number of errors corrected while decoding (bit flips). Don't forget to discard termination bits.

Clarifications

- There are such things as recursive convolutional codes. They are out of scope.
- There are other decoding algorithms except for Viterbi. They are out of scope.
- There are convolutional codes which receive two bits in parallel as input. This is out of scope (i.e., $k = 1$ for this work).
- To easily choose the likely path termination is usually employed. **We will use zero tailed termination**. This means you need to assume/ verify that the encoder is always at the zero state at the beginning and end of encoding. This is done by appending a *constraint length* number of 0 bits at the end of the input bits sequence, prior to encoding.
- On the decoder end, the zero tailed termination, implies you can always start the paths on the trellis form the zero state and discard all paths which don't end at a zero state.
-

What you're allowed to do?

- You may add methods and classes (but you must submit a single .py file)
- You may import all built-in modules (no pip install whatsover).

What you're **not** allowed to do?

Doing any of these will fail your code. Resulting, grade could be as low as zero.

- Don't try to import any modules other than built-in ones. They won't be installed, and your code will crash upon import.
- Don't change the prototype of provided methods, our testing code will only call these.
- Don't split your code to multiple files, we cannot guarantee execution.

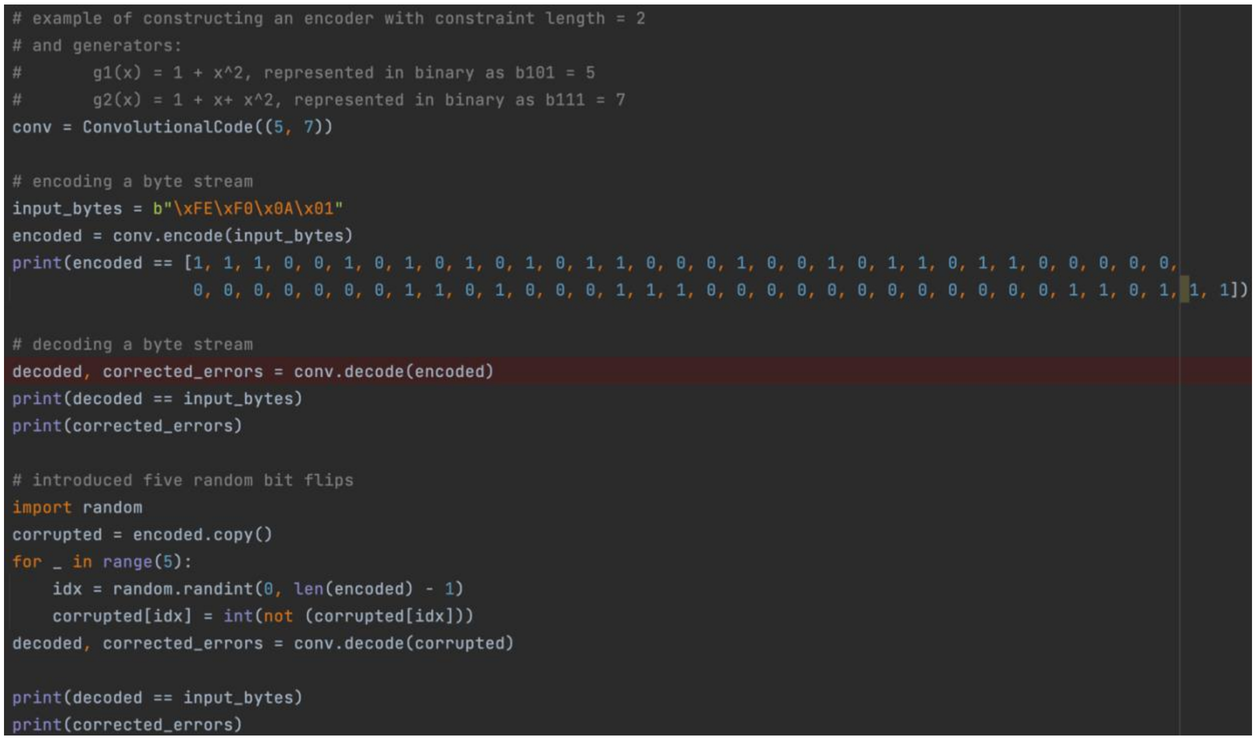
How will we test your code?

Testing will be similar to tests provided prior to submission. Basically, build various codes using the constructor, and call encoding and decoding methods. Grading will be based on comparing the methods output with expected outputs.

Examples

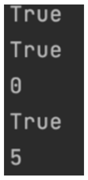
For examples, see skeleton file. The same examples are shown here:

Prints:



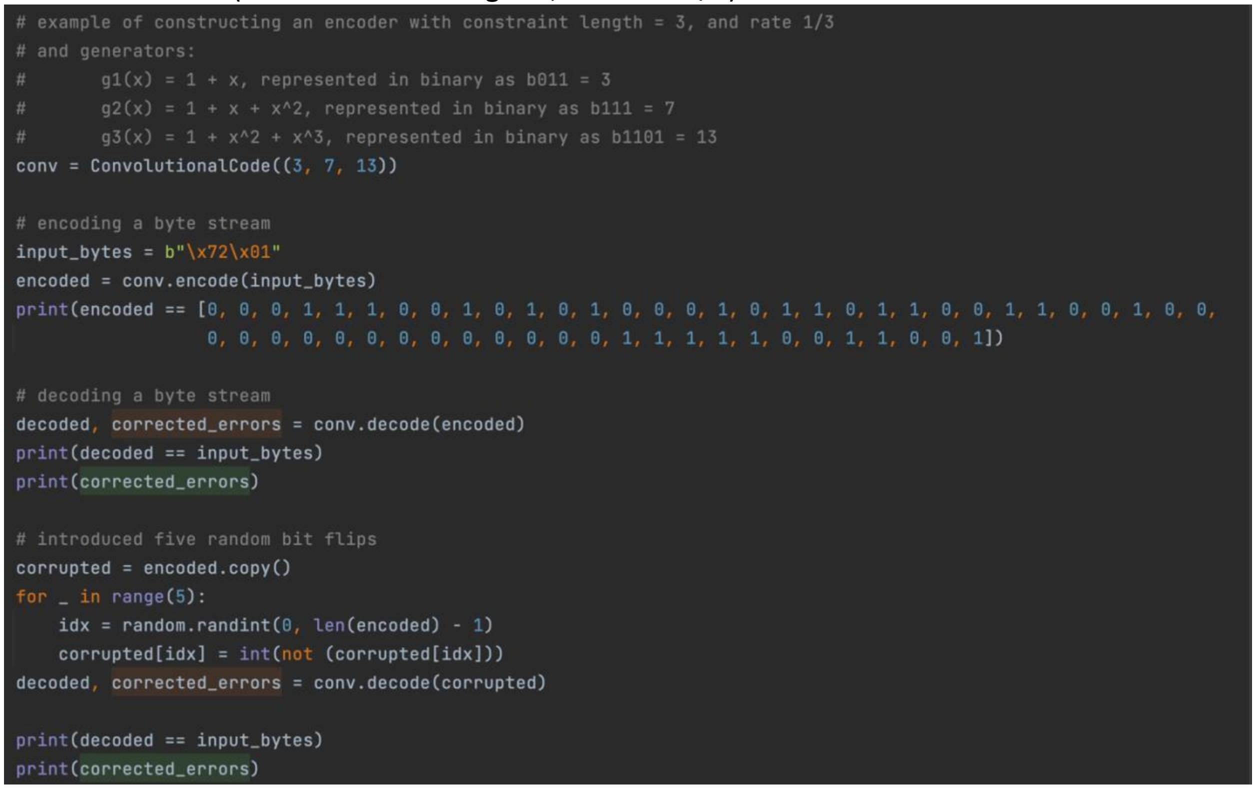
```python
# example of constructing an encoder with constraint length = 2
# and generators:
#       g1(x) = 1 + x^2, represented in binary as b101 = 5
#       g2(x) = 1 + x+ x^2, represented in binary as b111 = 7
conv = ConvolutionalCode((5, 7))

# encoding a byte stream
input_bytes = b"\xFE\xF0\x0A\x01"
encoded = conv.encode(input_bytes)
print(encoded == [1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0,
                  0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1])

# decoding a byte stream
decoded, corrected_errors = conv.decode(encoded)
print(decoded == input_bytes)
print(corrected_errors)

# introduced five random bit flips
import random
corrupted = encoded.copy()
for _ in range(5):
    idx = random.randint(0, len(encoded) - 1)
    corrupted[idx] = int(not (corrupted[idx]))
decoded, corrected_errors = conv.decode(corrupted)

print(decoded == input_bytes)
print(corrected_errors)
```

Prints:



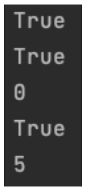
```
True
True
0
True
5
```

Another encoder (with constraint length 3, and rate 1/3):



```python
# example of constructing an encoder with constraint length = 3, and rate 1/3
# and generators:
#       g1(x) = 1 + x, represented in binary as b011 = 3
#       g2(x) = 1 + x + x^2, represented in binary as b111 = 7
#       g3(x) = 1 + x^2 + x^3, represented in binary as b1101 = 13
conv = ConvolutionalCode((3, 7, 13))

# encoding a byte stream
input_bytes = b"\x72\x01"
encoded = conv.encode(input_bytes)
print(encoded == [0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0,
                  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1])

# decoding a byte stream
decoded, corrected_errors = conv.decode(encoded)
print(decoded == input_bytes)
print(corrected_errors)

# introduced five random bit flips
corrupted = encoded.copy()
for _ in range(5):
    idx = random.randint(0, len(encoded) - 1)
    corrupted[idx] = int(not (corrupted[idx]))
decoded, corrected_errors = conv.decode(corrupted)

print(decoded == input_bytes)
print(corrected_errors)
```

Prints:



```
True
True
0
True
5
```