Coduri maximale binare – documentație

Dându-se n (lungimea) și d (distanța) să se genereze un cod maximal de tip (n,M,d) și găsește o valoare potențială pentru M. Proiectul conține:

**CodeWord.py**

* Conține clasa *CodeWord* ce reprezintă un cuvânt cod
* Membrii:
  + bits\_list – reprezintă lista de biți din care este alcătuit cuvântul-cod
* Metode:
  + **\_\_init\_\_ (self, binary\_representation)**

Constructorul clasei, ce primește ca parametru un șir de biți și construiește lista de biți din care este format cuvântul-cod.

* + **get\_digits(self)**

Getter pentru bits \_list*.*

* + **get\_length(self)**

Getter pentru lungimea listei de biți.

* + **get\_distance(word1, word2)**

Funcție statică ce calculează distanța (numărul de biți diferiți) dintre cuvintele primite ca parametru: *word1* și *word2.*

**CodeGenerator.py**

* Conține clasa *CodeGenerator* care se ocupă cu generarea codului maximal
* Membrii:
  + length – reprezintă lungimea codului
  + distance – reprezintă distanța codului
  + code\_words – reprezintă lista de cuvinte-cod ce va fi generată
  + word\_generator – este un obiect de tip generator care este folosit pentru a genera un nou cuvând-cod
* Funcții:
  + **\_\_init\_\_(self, length, distance)**

Constructorul clasei, ce inițializează lungimea și distanța codului cu parametrii primiți. Tot aici se inițializează obiectul generator de coduri, prin inițializarea cu o funcție generator. Acesta este *­word\_generator()* iar sintaxa de inițializare a obiectului generator este: *iter(self.word\_generation()).*

* + **word\_generation(self)**

Funcția construiește și returnează pe rând toate cuvintele-cod de lungime n. Acestea sunt reprezentate de scrierile binare ale numerelor de la 0 la 2n-1, și pentru fiecare astfel de scriere se va returna un obiect de tipul *CodeWord* ce primește în constructor numărul în forma binară. Funcția este o funcție de tip generator deoarece ea returnează cuvintele folosind ***yield****,* procedeu prin care se returnează la fiecare apel tot următorul cuvânt din listă, fără a se memora lista în memorie. Când nu mai sunt cuvinte, se aruncă excepție.

* + **generate\_word(self)**

Funcția returnează un nou cuvânt generat de obiectul-generator, prin sintaxa *next(self.word\_generator)*.

* + **generate\_code(self)**

Funcția construiește lista de cuvinte-cod care respectă lungimea și distanța dată. Inițial se adaugă în listă primul cuvânt generat, după care am o buclă infinită. Aici îmi generez un cuvânt cod posibil, și verific dacă acesta este valid (prin verificarea distanțelor dintre el și cuvintele deja generate din listă). Dacă este valid, atunci se adaugă la listă. Acest procedeu este conținut într-un bloc *try*, și dacă nu mai sunt cuvinte de generat, se aruncă o excepție. În acest moment execuția sare la blocul *except* care întrerupe bucla infinită. Funcția returnează lista de cuvinte generate.

* + **valid\_word(self, possible\_word)**

Funcția verifică dacă un cuvânt-cod posibil poate fi adăugat listei de cuvinte generate astfel încât să se respecte distanța. În acest scop, calculez distanțele de la cuvântul posibil la restul cuvintelor și dacă una din acestea este mai mică decât

**main.py**

* conține punctul de pornire al aplicației, funcția *\_\_main\_\_*
* funcții:
  + **run\_tests()**

Funcția execută testele din fișierul Tests.py.

* + **read\_inputs()**

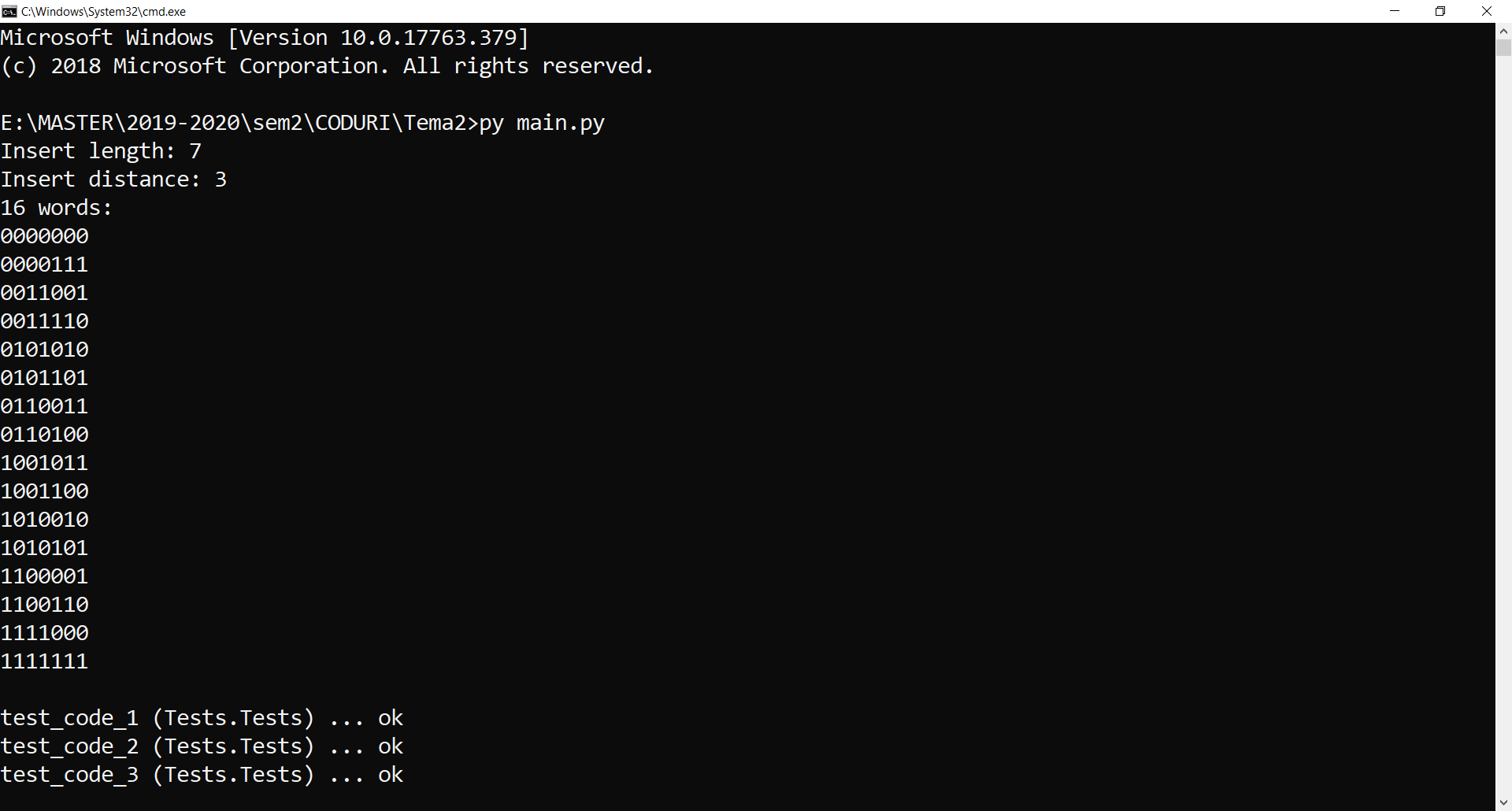
Funcția citește de la tastatură și returnează lungimea și distanța codului.

* + **\_\_main\_\_()**

Aici se citesc lungimea și distanța codului apelând *read\_inputs()*, se instanțiază un obiect de tip *CodeGenerator,* se apelează funcția *generate\_code()* care returnează lista de cuvinte-cod. Dacă lista conține un singur cuvânt (primul cuvânt generat) înseamnă că nu se pot genera cuvinte. Altfel afișez cuvintele generate.

**Tests.py**

* Fișierul conține clasa *Tests*, are mai multe teste ce verifică corectitudinea cuvintelor-cod generate de clasa *CodeGenerator*, în funcție de lungimea și distanța setată.

**Output**