In this assignment, we want to establish **bounds** on the size of **optimal edit metric** **error-correcting codes** using a chosen computational approach.

**We want to show that a (n, d) q code with more than M codewords cannot exist, for some chosen values n, M, d, and q (by an exhaustive search for such a code)**

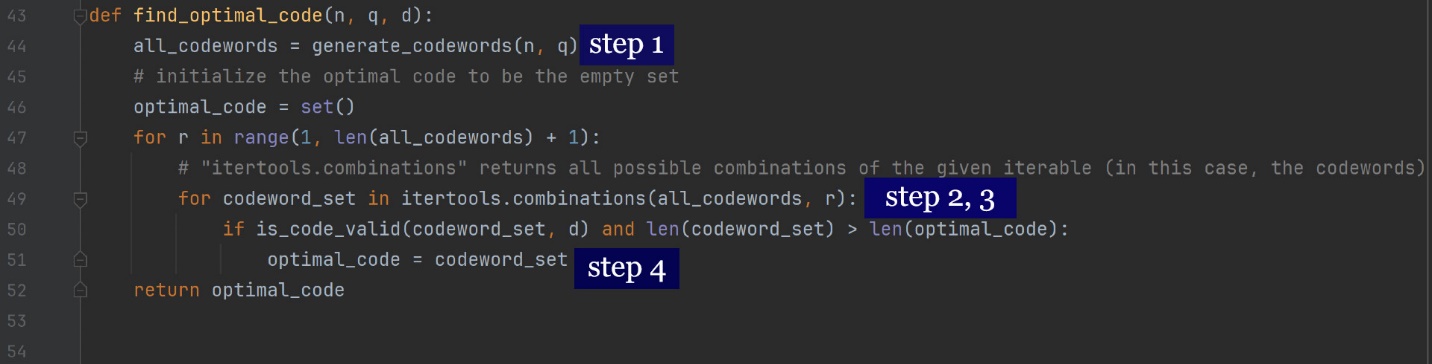
Assumptions:

* We will work with **fixed-length** (n, M, d) q​ **edit metric codes**.
* These are sets of M q-ary codewords of length n where every pair of codewords is at least d edit distance apart.
* We want to determine the largest possible value of M for a given n, d, and q such that there exists an (n, M, d) q code.

We start with the **simplest possible** (maybe the worst in terms of computational time ☹) approach and then try to improve it step by step. For the simplest approach to solve this problem, we can:

1. Generate all possible q-ary codewords of length n
2. Create all possible combination sets of those codewords
3. Check whether all pairs of codewords in each set have an edit distance >= d and if they do, they are valid.
4. Keep the valid set with the greatest number of codewords as an optimal edit metric error-correcting code with M codewords.

In the **simplest.py** file, you will find my code following the above steps as below:



Notes:

* The “generate\_codewords” function generates all possible q-ary codewords of length n
* The “is\_code\_valid” function checks whether the given code is a valid code with minimum edit distance >= d between any two codewords. For that purpose, it uses the “calculate\_edit\_distance” function implementing the algorithm in *slide 13- week7* and using a [*dynamic programming technique*](https://www.youtube.com/watch?v=We3YDTzNXEk).

**Analysis of the simplest approach**

The “find\_optimal\_code” function finds the optimal code of length n and size q with minimum edit distance d between any two codewords by exhaustively checking all the possible sets of codewords.

In this approach, we must check the space of sets of codewords. (where is the number of possible codewords and in that case, we have subsets )

For each set, we need to check the edit distance between every pair of codewords. In the worst case, this is comparisons per set, where M is the maximum number of codewords in a set.

Each comparison takes O(m \* n) time, where m is the length of codeword1 and n is the length of codeword2. (In our assumptions we know that we are working on **fixed-length** (n, M, d) q​ **edit metric codes**. So, we can say each comparison takes O(n2) time)

As a result, the total time complexity is O( \* \* n2) which obviously needs improvements. Let’s see how far we can go with this approach and find the optimal codes with different parameters.

**q = 2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N \ d | 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 1 | 1 | 1 | 1 |
| 2 | 4 | 2 | 1 | 1 | 1 |
| 3 | 8 | 4 | 2 | 1 | 1 |
| 4 | 16 | 8 | 2 | 2 | 1 |
| 5 | **?** | **?** | **?** | **?** | **?** |

* + - For n = 4, d = 1 it took about a second
    - For n = 4, d = 2 it took 1.36s
    - For n = 5, d = 1 it took too much time :(

**q = 3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N \ d | 1 | 2 | 3 | 4 | 5 |
| 1 | 3 | 1 | 1 | 1 | 1 |
| 2 | 9 | 3 | 1 | 1 | 1 |
| 3 | **?** | **?** | **?** | **?** | **?** |

* + - For n = 2, d = 1 it took 0.02s
    - For n = 3, d = 1 it took too much time :(

**q = 4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N \ d | 1 | 2 | 3 | 4 | 5 |
| 1 | 4 | 1 | 1 | 1 | 1 |
| 2 | 16 | 4 | 1 | 1 | 1 |
| 3 | **?** | **?** | **?** | **?** | **?** |

* + - For n = 2, d = 1 it took 10.71s
    - For n = 2, d = 2 it took 0.5s
    - For n = 3 it took too much time :(

**Conclusion:** It is obvious that we cannot go beyond n = 4 or n = 5 in a reasonable amount of time. So, we start improving our approach.

To improve our approach, we can use **the backtracking algorithm** introduced in *week5slides*. Backtracking is a more efficient approach compared to a simple exhaustive search. By using backtracking, we can eliminate large portions of the search space that cannot possibly contain a solution, thereby reducing computation time significantly.