**Structure Pattern**

*We need to experiment on various atomic structures for the crystals we've harvested, but our equipment is in rough shape needs improvement. For starts you should know how to recognize simple row structures even without details: the atoms comprising the crystalline structure are represented as digits and letters. The first patterns we check will be simple and the system will only need to recognize the order of the digits and letters. Unfortunately, our computer can only store integers and convert them into binary format for comparison. As a result we need a program which can compare a pattern with a given atomic structure.*

You are given a pattern as a positive integer and you are also given a row structure as a word. For comparison, the recognition system should convert the integer pattern into binary form. It needs to append zeros to left to match the structure length and compare this combination with the structure.

**1 is a letter and 0 is a digit.**

If the pattern and the structure match, then return True, else return False. If the pattern is longer than a given structure, then they are not a match (Example: 8 = 1000 and "a").

Here's an example: the given pattern is 42 and the structure is "12a0b3e4". 42 == 101010 in binary form, but this is not long enough to match the structure. Append zeroes to the left to get "00101010". Now compare the two:

 42 == 00101010

12a0b3e4 == DDLDLDLD

--------------------

 True VVVVVVVV

Here's one more example -- 101 and "ab23b4zz":

 101 == 01100101

ab23b4zz == LLDDLDLL

--------------------

 False XVXVXXXV

**Input:** A pattern as a positive integer and a command as a string. The third argument is optional with default value defines a level of patterns.

**Output:** Determination if the pattern matches the command as a boolean.

**Example:**

check\_structure(42, "12a0b3e4") == True

check\_structure(101, "ab23b4zz") == False

**Precondition:**

structure matches by "\A[A-Za-z0-9]{1, 32}\Z" regexp expression.

pattern\_level = 2

**Rank 2**:

In default pattern level (level = 2) we using binary form. If the patter level equals 3, then convert the integer pattern into ternary numeral system (base 3). **0 is a digit, 1 is a lowercase letter, 2 is an uppercase letter.**

*Precondition rank 2:*

structure matches by "\A[a-zA-Z0-9]{1, 32}\Z" regexp expression.

pattern\_level ∈ {2, 3}

**Rank 3**: If the pattern level equals 4, then convert the integer pattern into Quaternary numeral system (base 4). **0 is a digit, 1 is a lowercase letter, 2 is an uppercase letter, 3 is a whitespace.**

*Precondition rank 3:*

structure matches by "\A[ a-zA-Z0-9]{1, 32}\Z" regexp expression.

pattern\_level ∈ {2, 3, 4}

**How it is used:**

In this mission you learn how to store complex data in simple numbers and how to work with simple patterns. You can expand this concept to take on more complex patterns with different or more complex numbering systems.

def check\_structure(pattern, structure, pattern\_level=2):

 pattern=convert(pattern,pattern\_level)

 if len(pattern)>len(structure):

 return False

 if len(pattern)<len(structure):

 pattern="0"\*(len(structure)-len(pattern))+pattern

 if pattern\_level==2:

 structure=structure.lower()

 for i in range(0,len(structure)):

 if pattern[i]=="0" and not(structure[i].isdigit()):

 return False

 if pattern[i]=="1" and not(structure[i].islower()):

 return False

 if pattern[i]=="2" and not(structure[i].isupper()):

 return False

 if pattern[i]=="3" and not(structure[i].isspace()):

 return False

 return True