TP - Pattern Matching

RANDO Tomás

1)

```
def existChar(String, c):
    for i in range(len(String)):
        if String[i] == c:
            return True
    return False
```

2)

```
def isPalindrome(String):
    n = len(String)
    for i in range(0, round(n/2)):
        if String[i] != String[-(i+1)]:
            return False
    return True
```

4)

```
def getBiggestIslandLen(String):
    biggest = 1
    char = String[0]
    aux = 1
    for i in range(1, len(String)):
        if String[i] != char:
            if aux > biggest:
                biggest = aux
            char = String[i]
            aux = 1
        else:
            aux += 1
    return biggest
```

5)

```
def isAnagram(String, String2):
    if len(String) != len(String2):
        return False
    m = ord("z") - ord("a")
    hashtable = dictionary.dictionary(m)
    for i in range(len(String)):
        dictionary.insert(hashtable, ord(String[i]), String[i])
    for i in range(len(String2)):
        if dictionary.search(hashtable, ord(String2[i])) == None:
            return False
        else:
            dictionary.delete(hashtable, ord(String2[i]))
    return True
```

6)

```
def verifyBalancedParentheses(String):
    n = len(String)
    contador = 0
    for i in range(0, n):
        if String[i] == "(":
            contador += 1
        if String[i] == ")":
            contador -= 1
        if contador < 0:
            return False
    if contador == 0:
        return True
    else:
        return False
```

7)

```
def reduceLen(String):
    n = len(String)
    newString = ""
    aux = String[0]
    for i in range(1, n):
        if String[i] != aux:
            if aux is not None:
                newString = newString + aux
            aux = String[i]
            if i == n-1:
                newString = newString + String[i]
        else:
            aux = None
    return newString
```

8)

```
def isContained(String, String2):
    n = len(String)
    for i in range(0, n):
        if String[i] == String2[0]:
            String2 = String2[1:]
            if String2 == "":
                return True
    return False
```

9)

```
def isPatternContained(String, String2, c):
    n = len(String)
    for i in range(0, n):
        if String[i] == String2[0]:
            String2 = String2[1:]
        elif String2[0] == c:
            String2 = String2[1:]

        if String2 == "":
            return True
    return False
```

10)

input	a	b	
0	1	0	a
1	2	0	a
2	2	3	b
3	4	0	a
4	2	5	b

Estado inicial	0	1	2	2	3	4	0	1	2	3	4	2	3	4	0	1	2
Estado final	1	2	2	3	4	5	1	2	3	4	2	3	4	5	1	2	3
Input	a	a	a	b	a	b	a	a	b	a	a	b	a	b	a	a	b

11)

```
def kmp2(P):
    m = len(P)
    pi = [0] * m
    pi[0] = 0
    k = 0
    for i in range (1, m):
        while k > 0 and P[k] != P[i]:
            k = pi[k-1]
        if P[k] == P[i]:
            k = k + 1
        pi[i] = k
    return pi

def biggestPrefix(T, P):
    m = len(P)
    n = len(T)
    pi = kmp2(P)
    k = 0
    maxK = 0
    for i in range(0, n):
        while k > 0 and T[i] != P[k]:
            k = pi[k]
        if T[i] == P[k]:
            k = k + 1
        if k > maxK:
            maxK = k
        if k == m:
            return P
            k = pi[k]
    return P[0:maxK]
```

12)

```
def computeTransitionFunction(P, alphabet):
    n = len(P) + 1
    matrix = [[0] * len(alphabet) for i in range(n)]
    for i in range(0, n):
        for j in range(0, len(alphabet)):
            k = min(n + 1, i + 1)
            while k > 0 and suffix(P[:k], P[:i] + alphabet[j]) == False:
                k = k - 1
            matrix[i][j] = k
    return matrix

def finiteAutomatonMatcher(T, P):
    n = len(T)
    m = len(P)
    alphabet = []
    for i in P:
        if i not in alphabet:
            alphabet.append(i)
    d = computeTransitionFunction(P, alphabet)
    q = 0
    for i in range(0, n):
        if T[i] in alphabet:
            k = alphabet.index(T[i])
            q = d[q][k]
        else:
            q = 0
        if q == m:
            return True
    return False
```

```
def suffix(S1, S2):
    n = len(S2)
    m = len(S1)
    if S2[n-m:] == S1:
        condition = True
    else:
        condition = False
    return condition
```

13)

```
def rabinKarp(T, P):
    n = len(T)
    m = len(P)
    hp = hash(P, None, None)
    ts = None
    firstChar = None
    for i in range(0, n - m + 1):
        string = T[i: i + len(P)]
        ts = hash(string, ts, firstChar)
        firstChar = T[i]
        if ts == hp:
            if string == P:
                return True
    return False

def hash(ST, key, firstChar):
    n = len(ST)
    if key == None:
        hashKey = 0
        for i in range(0, n):
            hashKey = hashKey + ((10 ** (n- 1 - i)) * ord(ST[i]))
    else:
        hashKey = 10 * (key - ((10 ** (n-1) )* ord(firstChar))) + ord(ST[n-1])
    return hashKey
```

14)

```python
def computePrefixFunction(P):
    m = len(P)
    pi = [0] * m
    pi[0] = 0
    k = 0
    for i in range (1, m):
        while k > 0 and P[k] != P[i]:
            k = pi[k-1]
        if P[k] == P[i]:
            k = k + 1
        pi[i] = k
    return pi

def KMP(T, P):
    m = len(P)
    n = len(T)
    pi = kmp2(P)
    k = 0
    list = []
    for i in range(0, n):
        while k > 0 and T[i] != P[k]:
            list.pop(0)
            k = pi[k]
        if T[i] == P[k]:
            list.append(i)
            k = k + 1
        if k == m:
            return list
            k = pi[k]
    return False
```