UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO

FACULTAD
DE INGENIERÍA

UNIVERSIDAD NACIONAL DE CUYO

FACULTAD DE INGENIERÍA

ALGORITMOS Y ESTRUCTURAS DE DATOS II

Pattern Matching

Yacante Daniel

Leg: 10341

Ejercicio 7

Para eliminar los pares de caracteres iguales, mediante un bucle while recorro la cadena desde la posición 0 hasta la penúltima viendo si el carácter de la posición actual es igual al de la siguiente, si es asi, aumento el contador en 2 (para saltarme el carácter repetido) y sigo recorriendo la cadena. Si el carácter actual no es igual al siguiente, paso el carácter actual a la variable que va guardando la cadena sin caracteres repetidos.

```python
def reduceLen(cadena):
    newCadena=""
    i=0
    l=len(cadena)-1
    while i<l:
        if not cadena[i]==cadena[i+1]:
            newCadena+=cadena[i]
            i+=1
        else:
            i+=2
            if i>=l:
                newCadena+=cadena[l]
    return newCadena
```

Ejercicio 8

Para ver si una palabra esta contenida en otra bajo la premisa impuesta que los caracteres tienen que estar en el mismo orden que en la palabra a buscar utilizo un contador que va aumentando a medida que encuentro caracteres de P en T.

```python
15  def isContained(pattern,cadena):
16      i=0
17      for c in cadena:
18          if c==pattern[i]:
19              if i==len(pattern)-1:
20                  return True
21              else:
22                  i+=1
```

Ejercicio 9

Para encontrar el patron con los comodines, divido a la cadena con el patron en partes que esten delimitadas por el comodin, luego para cada una de las partes me fijo si desde la posición actual (al principio 0) hasta el fin de la cadena a buscar el patron, aparece la primer porción del patron, si se encuentra, actualizo la posición desde la cual voy a buscar en la cadena el siguiente pedazo de patron, y asi con cada uno de los trozos de patron. Si alguno de los trozos no se encuentra en lo que queda de cadena, se regresa False.

```python
24  def isPatternContained(patter:str,cadena:str,comodin:str):
25      partes=patter.split(comodin)
26      indAnt=0
27      for part in partes:
28          ind=cadena[indAnt:].find(part)
29          if ind==-1:
30              return False
31          else:
32              indAnt+=ind+len(part)
33      return True
```

Ejercicio 10

Diagrama de transición del automata:

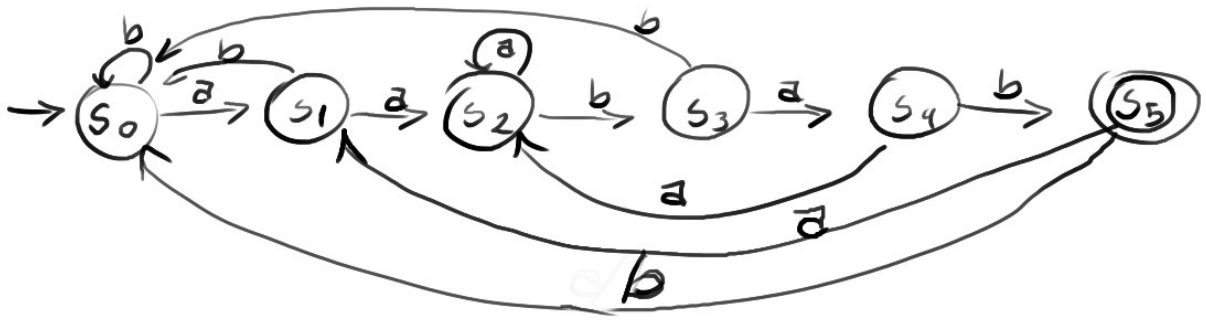
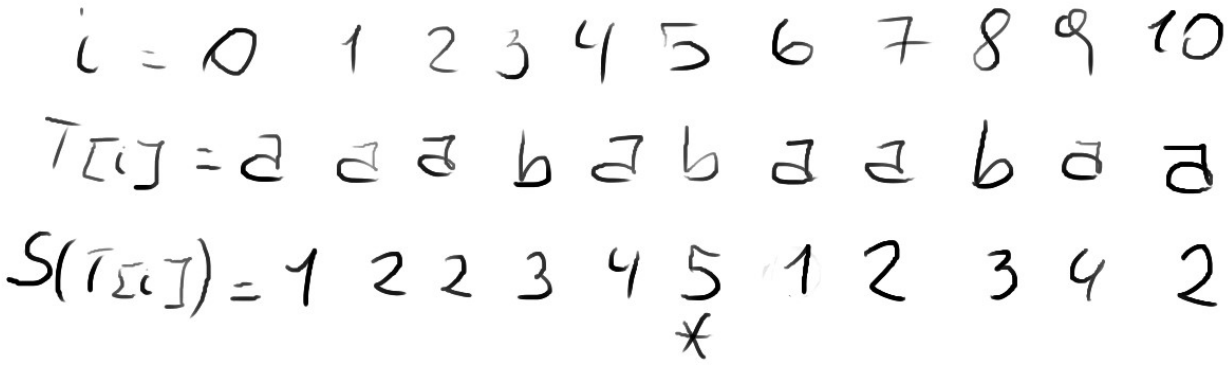


Tabla con funciones de Transferencia entre estados.

S	a	b
→0	1	0
1	2	0
2	2	3
3	4	0
4	2	5
*5	1	0

Prueba en cadena T:



Ejercicio 11

Se podría encontrar todos los posibles prefijos y guardarlos en un array, y luego mediante un autómata finito que reconozca P, en cada uno de los estados para los cuales uno se encuentra en un prefijo de P, marcar en el array en que índice se encuentra, si el índice que se encuentra es mayor al que tenia previamente, se actualiza dicho índice y se continua, de no ser asi, no se actualiza dicho índice ya que hay un prefijo que ha sido mayor. Al finalizar de recorrer la cadena T, si entramos al arrary de los prefijos con el mayor índice que quedo obtendremos el mayor de los prefijos de P en T.

Ejercicio 13

```python
def rabinKarp(cad1:str,cad2:str):
    m=len(cad1)#Patron P
    n=len(cad2)#Cadena T
    hashP=0
    hashT=0
    for i in range(len(cad1)):
        e=m-i-1
        hashP+=ord(cad1[i])*pow(128,e)
    for i in range(len(cad1)):
        hashT+=ord(cad2[i])*pow(128,m-i-1)
    fhash=lambda hAnt,a,b,l: 128*(hAnt-pow(128,l)*ord(a))+ord(b)
    for i in range(len(cad2)-len(cad1)):
        subC=cad2[i:i+m]
        if hashT==hashP:
            if cad1==subC:
                return True
        else:
            hashT=fhash(hashT,cad2[i],cad2[i+m],m-1)
    return False
```