Tecnológico de Monterrey

Implementación de métodos computacionales (Gpo 603)

Evaluación 3.2. Implementación PDA palíndromos con backtracking

Equipo 3

Diego Alejandro Espejo López A01638341

Julieta Carolina Arteaga Legorreta A0637444

Marcela Beatriz De La Rosa Barrios A01637239

Pablo Heredia Sahagún A01637103

Adela Alejandra Solorio Alcázar A01637205

22/04/2024

Evaluación 3.2. Implementación PDA palíndromos con backtracking

EXPLICACIÓN DEL CÓDIGO

El código es una implementación de un autómata de pila (PDA) en Python para verificar si una palabra es un palíndromo, este utiliza un stack y realiza transiciones de estado basadas en los símbolos de entrada y el contenido del stack.

- Función **is_palindrome(palabra)**: función principal que determina si la palabra es un palíndromo o no (True or False). Llama a la función **q0** con el índice inicial 0 y un stack vacío.
- Funciones Internas **q0** y **q1**: funciones representan los **estados del autómata** y manejan la lógica de adición y eliminación en/del stack para verificar la propiedad de palíndromo.

Función q0(i, stack)

- Prepara el stack y explora posibles caminos para formar un palíndromo, ya sea añadiendo caracteres al stack o avanzando sin modificarla.
- Step by step:
  - Si el índice i llega al final de la palabra, retorna False porque no se ha confirmado que sea un palíndromo.
  - Evalúa tres posibles caminos para determinar si la palabra puede ser un palíndromo:
    - Primera llamada a **q1**: checa si la palabra es un palíndromo comenzando desde el índice actual i con el stack actual.
    - Segunda llamada a **q1**: ignora el carácter en el índice actual i y verifica si la palabra es un palíndromo comenzando desde el índice i+1.
    - Llamada recursiva a **q0**: añade el carácter actual a la stacky verifica si la palabra es un palíndromo con el nuevo estado de la stack.
- Regresa True en caso de que alguna de las llamadas anteriores confirme que la configuración de caracteres permite la formación de un palíndromo.

Función q1(i, stack)

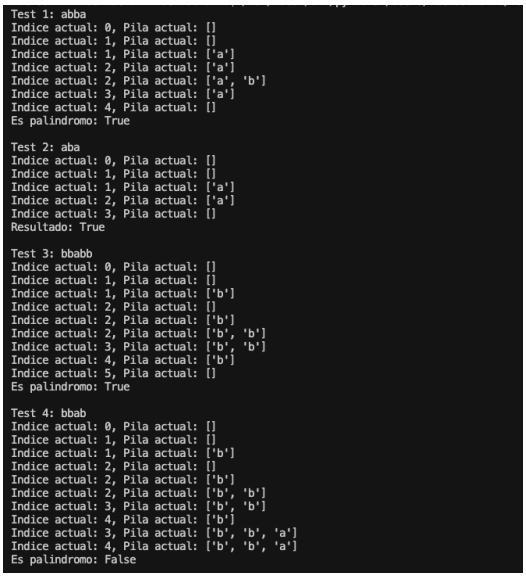
- Verifica si los caracteres que quedan en la palabra coinciden con los elementos del stack uno por uno
- Step by step:
  - Imprime el índice y el estado actual del stack.

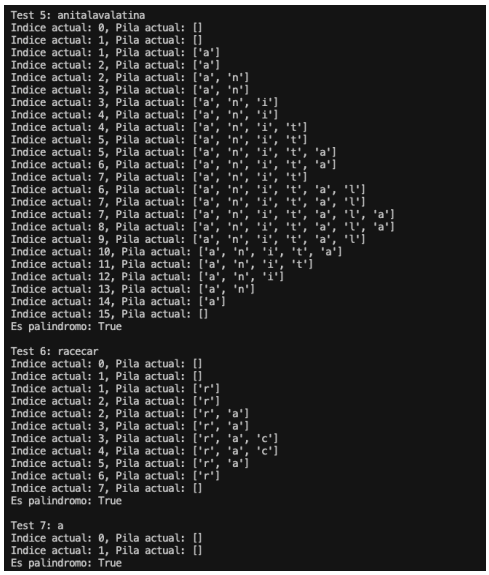
- Si el índice i es igual al tamaño de la palabra y el stack está vacío, retorna True porque todos los caracteres han sido validados como un palíndromo.
- Si el índice es mayor o igual al tamaño de la palabra pero el stack no está vacío, retorna False.
- Si el carácter en el índice actual i coincide con el top of stack, elimina ese carácter del stack y verifica el siguiente carácter con el estado actualizado del stack a (q1(i + 1, stack2)).
- Si no hay coincidencia, retorna False.

CÓDIGO FINAL ESTILIZADO

```
# Estado q1: Elimina las letras de la pila y verifica si la palabra es palíndromo
(si la pila está vacía y el índice llega al final de la palabra)

def is_palindrome(palabra):
    def q1(i, stack):
        stack2 = stack[:] # Copia la pila para no modificar la original
        print(f"Índice actual: {i}, Pila actual: {stack2}") # print índice & stack
        # If índice llega al final de palabra y stack está vacío, regresa True (palíndromo)
        if i == len(palabra) and len(stack2) == 0:
            return True
        # if índice llega al final de la palabra pero la pila no está vacía, regresa False
        if i >= len(palabra):
            return False
        if stack2 and (stack2[-1] == palabra[i]): # if stack not empty y stack.top
coincide con el carácter en el índice i
            stack2.pop()  #elimina el top de stack y continua verificando
            return q1(i + 1, stack2)
        return False   # Si no, regresa False (no es palindromo)

# Estado q0: Añade las letras a la pila o ignora un carácter (en caso de ser impar)
    def q0(i, stack):
        stack2 = stack[:]  # Copia la pila para evitar modificar la original
        if i == len(palabra): # Si el índice llega al final de la palabra, regresa False
            return False

        # Si cumple con las transiciones de q0 y q1, regresa True (es palindromo)
        res = False # Inicializa la variable de resultado (el estado final) en False
        # Checa si la letra actual a partir del índice i es un palíndromo con la pila actual
        res = res or q1(i, stack2)
        # Checa si la siguiente letra en la palabra forma un palíndromo con la pila actual
        res = res or q1(i + 1, stack2)
        # Añade la letra actual a la pila y checa si la palabra resultante es un palíndromo
```

```
        stack2.append(palabra[i])
        res = res or q0(i + 1, stack2)
        # Regresa el resultado final
        return res
    return q0(0, []) # Regresa el resultado de la función q1

def test_palindrome():
    print("Test 1: abba")
    test = is_palindrome("abba")
    print(f"Es palindromo: {test}\n")

    print("Test 3: bbabb")
    test = is_palindrome("bbabb")
    print(f"Es palindromo: {test}\n")

    print("Test 4: bbab")
    test = is_palindrome("bbab")
    print(f"Es palindromo: {test}\n")

    print("Test 5: anitalavalatina")
    test = is_palindrome("anitalavalatina")
    print(f"Es palindromo: {test}\n")

    print("Test 7: a")
    test = is_palindrome("a")
    print(f"Es palindromo: {test}\n")

if __name__ == "__main__":
    test_palindrome()
```





TEST CASES: Capturas de pantalla

BASE PARA EL CÓDIGO FINAL (LÓGICA CON COMENTARIOS)

```
palabra = "babab"

def d(i, s): #ESTADO Q1
    s2 = s[:] #copia el stack para no modificar el OG
    print(i,s2) #imprime el índice y el stack
    if i == len(palabra) and len(s2)==0: #caso base (i=palabra & stack = empty)
        return True #es palíndromo
    if i >= len(palabra): #si i es mayor o igual a la longitud de la palabra
        return False #no es palíndromo
    if s2 and (s2[-1] == palabra[i]): #TRANSICIÓN, si el elemento del top of stack es igual a la letra actual
        s2.pop() #quita el último elemento de la lista
        return d(i+1,s2)  #pasa a la siguiente letra
    return False #si no se cumple la condición anterior, no es palíndromo

def f(i,s): #ESTADO Q0
    s2 = s[:] #copia del stack
    if i==len(palabra): #si i es igual a la longitud de la palabra y no haz pop nada
        return False #no es palíndromo, estado de agregación no es final
    # si una de las sig 3 líneas pasa, es palindromo
    res = False
    res = res or d(i,s2) #transición cuando pasamos al delete con E/E/E
    res = res or d(i+1,s2) #transición cuando pasamos al delete con a/E/E o b/E/E
    s2.append(palabra[i]) #agrega la letra actual al stack, PT DE TRANSICIÓN (lo q metes)
    res = res or f(i+1,s2) #transición en la que te quedas en el mismo estado de agregación a/E/a o b/E/b
    return res #si no se cumple ninguna de las condiciones anteriores, no es palíndromo

print("El string",palabra,"es palíndromo?")
print(f(0,[]))
```

```
El string babab es palíndromo?
0 []
1 []
1 ['b']
2 ['b']
3 []
2 ['b', 'a']
3 ['b', 'a']
4 ['b']
5 []
True
```

Otros casos de prueba:

```
El string abbba es palíndromo?
0 []
1 []
1 ['a']
2 ['a']
2 ['a', 'b']
3 ['a']
3 ['a', 'b']
4 ['a']
5 []
True
```

```
El string abbab es palíndromo?
0 []
1 []
1 ['a']
2 ['a']
2 ['a', 'b']
3 ['a']
4 []
3 ['a', 'b']
3 ['a', 'b', 'b']
4 ['a', 'b', 'b']
5 ['a', 'b']
4 ['a', 'b', 'b', 'a']
5 ['a', 'b', 'b', 'a']
False
```

```
El string aa es palíndromo?
0 []
1 []
1 ['a']
2 []
True
```

```
El string bab es palíndromo?
0 []
1 []
1 ['b']
2 ['b']
3 []
True
```

```
El string a es palíndromo?
0 []
1 []
True
```