Guía de Strings para Algoritmos

Referencia completa de métodos y técnicas en Python

M.H. Alberto

1 NIVEL BÁSICO

1.1 Acceso y Recorrido

1.1.1 Indexación Básica

Operaciones fundamentales de acceso directo.

- `len(s)` → longitud total del string
- `s[i]` → carácter en posición i (índice desde 0)
- `s[0]` → primer carácter
- `s[-1]` → último carácter
- `s[-2]` → penúltimo carácter
- `s[len(s)-1]` → equivalente a `s[-1]`

Nota: El acceso directo es O(1). Los índices negativos cuentan desde el final.

1.1.2 Iteración y Búsqueda

Recorrido directo:

```
for char in s:
    print(char)  # carácter por carácter
```

Con índice:

```
for i in range(len(s)):
    print(i, s[i])  # posición y carácter
```

Con enumerate:

```
for i, char in enumerate(s):
    print(i, char)  # más pythónico
```

Membresía y búsqueda:
- `"x" in s` → True si existe (O(n))
- `"xyz" in s` → funciona con subcadenas
- `"x" not in s` → verificar ausencia

1.1.3 Slicing Avanzado

Sintaxis general: `s[start:end:step]`

Extracción básica:
- `s[2:5]` → caracteres en posiciones 2, 3, 4
- `s[:3]` → primeros 3 caracteres
- `s[3:]` → desde posición 3 hasta el final
- `s[:]` → copia completa del string

Técnicas avanzadas:
- `s[::-1]` → invertir string completamente
- `s[::2]` → caracteres en posiciones pares (0,2,4...)
- `s[1::2]` → caracteres en posiciones impares
- `s[::-2]` → cada segundo carácter en reversa
- `s[2:8:2]` → del 2 al 7, saltando de 2 en 2

Rotación de strings:

```
# Rotar n posiciones a la izquierda
s[n:] + s[:n]

# Rotar n posiciones a la derecha
s[-n:] + s[:-n]
```

1.2 Normalización de Datos

1.2.1 Limpieza de Espacios

Eliminar whitespace:
- `s.strip()` → quita espacios, tabs, enters de ambos lados
- `s.lstrip()` → solo del lado izquierdo
- `s.rstrip()` → solo del lado derecho

Eliminar caracteres específicos:
- `s.strip(".,!?")` → quita puntuación de extremos
- `s.strip("xyz")` → quita cualquier combinación
- `s.lstrip("0")` → útil para eliminar ceros a la izquierda

Importante: `strip()` no modifica el string original, retorna uno nuevo.

1.2.2 Transformación de Caso

Cambios completos:
- `s.lower()` → todo en minúsculas
- `s.upper()` → todo en MAYÚSCULAS
- `s.swapcase()` → invierte el caso (a→A, A→a)

Capitalización:
- `s.capitalize()` → solo primera letra mayúscula
- `s.title()` → Primera Letra De Cada Palabra
- `"hola mundo".title()` → «Hola Mundo»

Uso común:

```
# Comparación case-insensitive
s1.lower() == s2.lower()

# Normalizar input de usuario
user_input.strip().lower()
```

1.2.3 Justificación y Relleno

Alineación:
- `s.center(width)` → centrar en ancho dado
- `s.ljust(width)` → alinear izquierda
- `s.rjust(width)` → alinear derecha

Con carácter de relleno:
- `s.center(10, "*")` → rellena con asteriscos

- `s.ljust(10, ".")` → puntos a la derecha
- `s.zfill(5)` → rellena con ceros a la izquierda

Ejemplo práctico:

```
# IDs con ceros: "42" → "00042"
id_str.zfill(5)

# Tabla alineada
f"{nombre.ljust(20)} {edad.rjust(3)}"
```

2 NIVEL INTERMEDIO

2.1 Búsqueda y Validación

2.1.1 Localización de Subcadenas

Búsqueda simple:
- `s.find("sub")` → índice de primera aparición o −1
- `s.rfind("sub")` → última aparición o −1
- `s.index("sub")` → índice o lanza ValueError
- `s.rindex("sub")` → desde el final o error

Conteo:
- `s.count("sub")` → número total de ocurrencias
- `s.count("x", start, end)` → contar en rango

Verificación de posición:
- `s.startswith("pre")` → True si empieza con
- `s.endswith("suf")` → True si termina con
- `s.startswith(("a", "b"))` → acepta tupla de opciones

Diferencia find vs index:

```
# find retorna -1 si no encuentra
pos = s.find("x")
if pos != -1:
    print("Encontrado en", pos)

# index lanza excepción
try:
    pos = s.index("x")
except ValueError:
    print("No encontrado")
```

2.1.2 Validación de Contenido

Tipos de caracteres:
- `s.isdigit()` → solo dígitos 0-9
- `s.isalpha()` → solo letras (a-z, A-Z)
- `s.isalnum()` → letras y/o números
- `s.isspace()` → solo whitespace
- `s.isascii()` → solo caracteres ASCII

Verificación de caso:
- `s.islower()` → todas minúsculas
- `s.isupper()` → todas MAYÚSCULAS
- `s.istitle()` → formato de título
- `s.isdecimal()` → dígitos decimales
- `s.isnumeric()` → números (incluye fracciones)

Casos de uso:

```
# Validar PIN numérico
if pin.isdigit() and len(pin) == 4:
    print("PIN válido")

# Verificar username alfanumérico
if username.isalnum() and len(username) >= 3:
    print("Username válido")
```

2.2 Transformación Estructural

2.2.1 División de Strings (Split)

Separación automática:
- `s.split()` → por cualquier whitespace
- `"a  b\tc\nd".split()` → [“a”, “b”, “c”, “d”]

Separador específico:
- `s.split(",")` → por comas (CSV)
- `s.split(";")` → por punto y coma
- `s.split("|")` → pipe como delimitador

Limitar divisiones:
- `s.split(",", maxsplit=2)` → máximo 2 splits
- `"a,b,c,d".split(",", 1)` → [“a”, “b,c,d”]

Métodos relacionados:
- `s.splitlines()` → divide por n, r, rn
- `s.partition("sep")` → tupla (antes, sep, después)
- `s.rpartition("sep")` → particiona desde el final

Ejemplo CSV:

```
csv = "Juan,25,México"
nombre, edad, pais = csv.split(",")
```

2.2.2 Unión de Listas (Join)

Sintaxis: `separador.join(lista)`

Formas comunes:
- `" ".join(palabras)` → unir con espacios
- `"".join(chars)` → concatenar sin separador
- `"\n".join(lineas)` → una por línea
- `", ".join(items)` → separados por coma+espacio
- `"-".join(partes)` → con guiones

Rendimiento:

```
# LENTO (crea muchos strings intermedios)
resultado = ""
for palabra in lista:
    resultado += palabra + " "

# RÁPIDO (una sola operación)
resultado = " ".join(lista)
```

Conversión de tipos:

```
# Convertir lista de números a string
numeros = [1, 2, 3, 4]
s = "".join(map(str, numeros))  # "1234"
s = ", ".join(map(str, numeros))  # "1, 2, 3, 4"
```

2.2.3 Reemplazo y Sustitución

Reemplazo simple:

- `s.replace("old", "new")` → todas las ocurrencias
- `s.replace("old", "new", count)` → limitar cantidad

Casos de uso:

```
# Eliminar espacios
s.replace(" ", "")

# Normalizar separadores
s.replace("\t", " ").replace("\n", " ")

# Censurar palabras
text.replace("palabra", "****")
```

Métodos de eliminación:
- `s.removeprefix("pre")` → Python 3.9+
- `s.removesuffix("suf")` → Python 3.9+

Traducción de caracteres:

```
# Tabla de traducción
tabla = str.maketrans("aeiou", "12345")
s.translate(tabla)  # vocales → números

# Eliminar caracteres
tabla = str.maketrans("", "", ".,!?")
s.translate(tabla)  # quita puntuación
```

2.3 Formato de Strings

2.3.1 f-strings (Python 3.6+)

Interpolación básica:

```
nombre = "Juan"
edad = 25
f"Me llamo {nombre} y tengo {edad} años"
```

Expresiones:

```
f"Suma: {a + b}"
f"Doble: {x * 2}"
f"Mayúsculas: {texto.upper()}"
```

Formato numérico:
- `f"{pi:.2f}"` → 2 decimales: 3.14
- `f"{num:.0f}"` → sin decimales: 42
- `f"{val:.2%}"` → porcentaje: 75.50%
- `f"{grande:,}"` → miles: 1,000,000

Alineación y relleno:
- `f"{val:>10}"` → alinear derecha en 10 espacios
- `f"{val:<10}"` → alinear izquierda
- `f"{val:^10}"` → centrar
- `f"{val:0>5}"` → rellenar con ceros: 00042
- `f"{val:*^10}"` → centrar con asteriscos

Debug (Python 3.8+):

```
x = 42
f"{x=}"  # "x=42"
f"{len(s)=}"  # "len(s)=10"
```

2.3.2 Método format()

Sintaxis clásica:

```
"Hola {}".format("mundo")
"Hola {0} {1}".format("Juan", "Pérez")
"Hola {nombre}".format(nombre="Juan")
```

Formato numérico:

```
"{:.2f}".format(3.14159)  # 3.14
"{:05d}".format(42)  # 00042
"{:,}".format(1000000)  # 1,000,000
```

3 NIVEL AVANZADO

3.1 Técnicas de Programación

3.1.1 List Comprehension con Strings

Filtrado:

```
# Solo dígitos
[c for c in s if c.isdigit()]

# Solo letras
[c for c in s if c.isalpha()]

# Sin vocales
[c for c in s if c.lower() not in "aeiou"]

# Caracteres únicos
[c for i, c in enumerate(s) if s.index(c) == i]
```

Transformación:

```
# Mayúsculas
[c.upper() for c in s]

# Códigos ASCII
[ord(c) for c in s]

# Duplicar cada carácter
[c*2 for c in s]
```

Filtrado + Transformación:

```
# Dígitos como enteros
[int(c) for c in s if c.isdigit()]

# Vocales en mayúsculas
[c.upper() if c in "aeiou" else c for c in s]
```

Reconstrucción:

```
# Limpiar string
"".join([c for c in s if c.isalnum()])

# Solo consonantes
"".join([c for c in s if c.isalpha() and c not
in "aeiou"])
```

3.1.2 Análisis de Frecuencias

Diccionario manual:

```
freq = {}
for char in s:
    if char in freq:
        freq[char] += 1
    else:
        freq[char] = 1

# O más compacto:
freq = {}
for char in s:
    freq[char] = freq.get(char, 0) + 1
```

Con defaultdict:

```
from collections import defaultdict

freq = defaultdict(int)
for char in s:
    freq[char] += 1
```

Con Counter:

```
from collections import Counter

# Crear frecuencias
freq = Counter(s)

# Métodos útiles
freq.most_common()  # todos ordenados
freq.most_common(3)  # top 3
freq.total()  # suma de valores
freq.elements()  # expandir a iterador
```

Operaciones con Counter:

```
c1 = Counter("hola")
c2 = Counter("adios")

c1 + c2  # sumar conteos
c1 - c2  # restar (mantiene positivos)
c1 & c2  # mínimo de cada elemento
c1 | c2  # máximo de cada elemento
```

3.1.3 Módulo string

Constantes útiles:

```
import string

string.ascii_lowercase  # 'abc...xyz'
string.ascii_uppercase  # 'ABC...XYZ'
string.ascii_letters    # a-z + A-Z
string.digits           # '0123456789'
string.punctuation      # '!"#$%&...'
string.whitespace       # ' \t\n\r...'
```

Casos de uso:

```
# Verificar si solo tiene letras
all(c in string.ascii_letters for c in s)

# Remover puntuación
"".join(c for c in s if c not in
string.punctuation)

# Generar alfabeto
list(string.ascii_lowercase)
```

3.2 Expresiones Regulares

3.2.1 Módulo re - Fundamentos

Importar:

```
import re
```

Funciones principales:

- `re.search(patron, s)` → encuentra primera coincidencia
- `re.match(patron, s)` → coincide desde el inicio
- `re.findall(patron, s)` → lista todas las coincidencias
- `re.finditer(patron, s)` → iterador de coincidencias
- `re.split(patron, s)` → dividir por patrón
- `re.sub(patron, repl, s)` → sustituir coincidencias

3.2.2 Patrones Básicos

Caracteres literales:

- `r"abc"` → busca «abc» exactamente
- `r"."` → cualquier carácter (excepto n)
- `r"\."` → punto literal (escapado)

Clases de caracteres:

- `r"\d"` → dígito [0-9]
- `r"\D"` → no-dígito
- `r"\w"` → palabra [a-zA-Z0-9]
- `r"\W"` → no-palabra
- `r"\s"` → whitespace
- `r"\S"` → no-whitespace

Conjuntos:

- `r"[abc]"` → a, b, o c
- `r"[a-z]"` → cualquier minúscula
- `r"[A-Z]"` → cualquier mayúscula
- `r"[0-9]"` → cualquier dígito
- `r"[^abc]"` → cualquier cosa EXCEPTO a, b, c

Cuantificadores:

- `r"a*"` → 0 o más a's
- `r"a+"` → 1 o más a's
- `r"a?"` → 0 o 1 a
- `r"a{3}"` → exactamente 3 a's
- `r"a{2,5}"` → entre 2 y 5 a's
- `r"a{3,}"` → 3 o más a's

3.2.3 Patrones Avanzados

Anclas:

- `r"^abc"` → empieza con abc
- `r"abc$"` → termina con abc

- `r"^abc$"` → exactamente abc
- `r"\b"` → límite de palabra

Agrupación:
- `r"(abc)+"` → repetir grupo
- `r"(a|b)"` → a o b
- `r"(?:abc)"` → grupo sin captura

Ejemplos prácticos:

```
# Extraer números
re.findall(r"\d+", "Hay 42 manzanas y 7 peras")
# ['42', '7']

# Extraer palabras
re.findall(r"\w+", "Hola, ¿cómo estás?")
# ['Hola', 'cómo', 'estás']

# Validar email simple
re.match(r"^\w+@\w+\.\w+$", email)

# Validar teléfono
re.match(r"^\d{3}-\d{3}-\d{4}$", "555-123-4567")

# Reemplazar múltiples espacios
re.sub(r"\s+", " ", "hola    mundo")
# "hola mundo"

# Eliminar no-alfanuméricos
re.sub(r"[^a-zA-Z0-9]", "", "Hola!123")
# "Hola123"
```

4 PATRONES DE PROBLEMAS

4.1 Palíndromos

Verificación simple:

```
s == s[::-1]
```

Ignorando espacios y puntuación:

```
# Con list comprehension
clean = "".join(c.lower() for c in s if
c.isalnum())
es_palindromo = clean == clean[::-1]

# Con regex
import re
clean = re.sub(r"[^a-z0-9]", "", s.lower())
es_palindromo = clean == clean[::-1]
```

Palíndromo con índices (sin reversa):

```
def es_palindromo(s):
    izq, der = 0, len(s) - 1
    while izq < der:
        if s[izq] != s[der]:
            return False
        izq += 1
        der -= 1
    return True
```

4.2 Anagramas

Con sorted (simple):

```
sorted(s1) == sorted(s2)
```

Con Counter (más eficiente):

```
from collections import Counter
Counter(s1) == Counter(s2)
```

Con diccionario manual:

```
def son_anagramas(s1, s2):
    if len(s1) != len(s2):
        return False

    freq = {}
    for c in s1:
        freq[c] = freq.get(c, 0) + 1

    for c in s2:
        if c not in freq:
            return False
        freq[c] -= 1
        if freq[c] < 0:
            return False

    return True
```

4.3 Subcadenas

Generar todas:

```
subcadenas = []
for i in range(len(s)):
    for j in range(i+1, len(s)+1):
        subcadenas.append(s[i:j])
```

Subcadenas de longitud k:

```
[s[i:i+k] for i in range(len(s)-k+1)]
```

Subcadenas únicas:

```
{s[i:j] for i in range(len(s))
        for j in range(i+1, len(s)+1)}
```

4.4 Rotaciones

Verificar si s2 es rotación de s1:

```
len(s1) == len(s2) and s2 in s1 + s1
```

Explicación: Si «mundo» es rotación de «hola», entonces «mundo» aparecerá en «holahola».

Generar todas las rotaciones:

```
rotaciones = [s[i:] + s[:i] for i in
range(len(s))]
```

4.5 Compresión de Strings

Run-Length Encoding:

```python
def comprimir(s):
    if not s:
        return ""

    resultado = []
    actual = s[0]
    conteo = 1

    for i in range(1, len(s)):
        if s[i] == actual:
            conteo += 1
        else:
            resultado.append(actual +
str(conteo))
            actual = s[i]
            conteo = 1

    resultado.append(actual + str(conteo))
    return "".join(resultado)

# "aaabbcccc" → "a3b2c4"
```

Descompresión:

```python
def descomprimir(s):
    resultado = []
    i = 0
    while i < len(s):
        char = s[i]
        i += 1
        count = ""
        while i < len(s) and s[i].isdigit():
            count += s[i]
            i += 1
        resultado.append(char * int(count))
    return "".join(resultado)

# "a3b2c4" → "aaabbcccc"
```