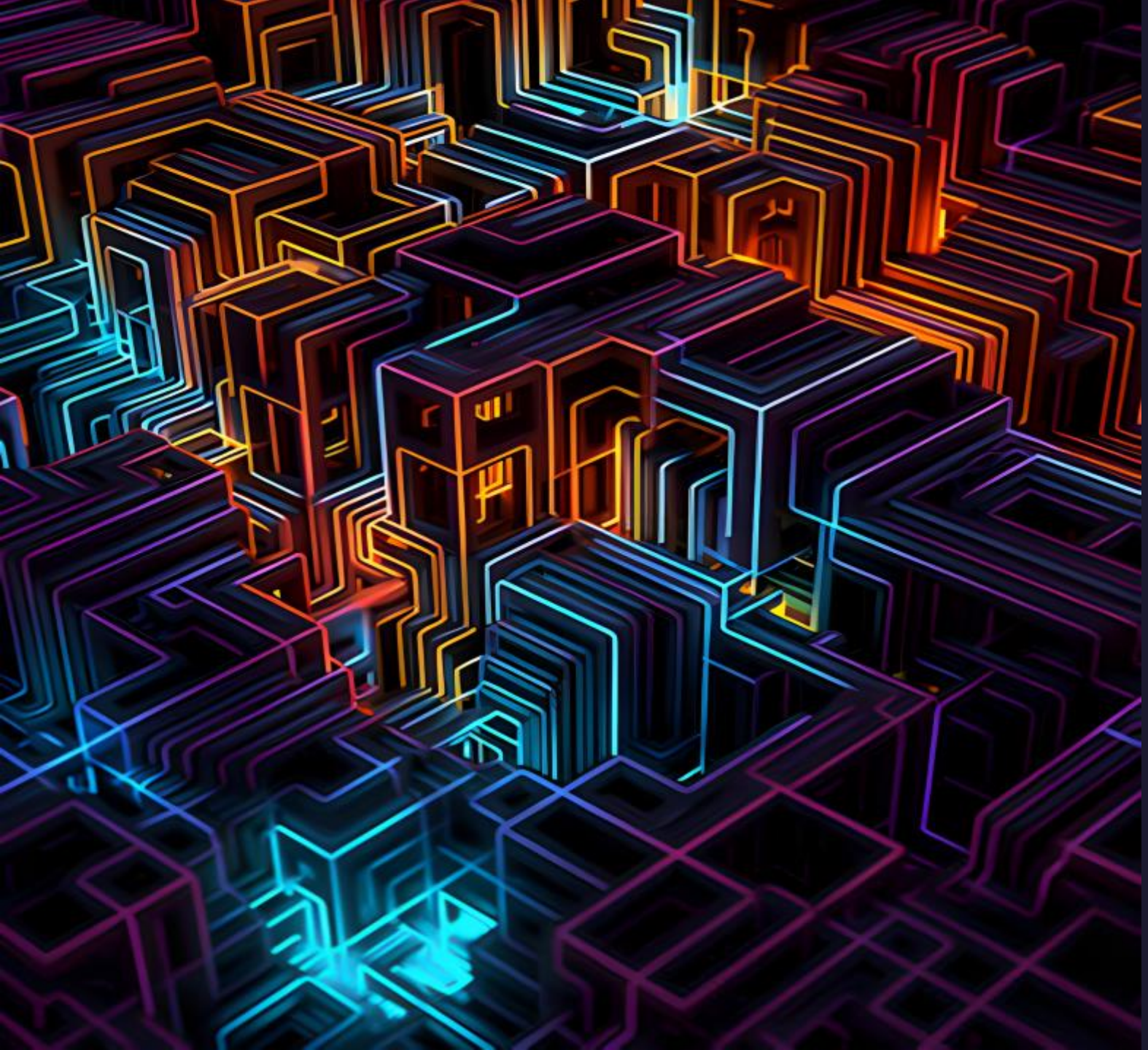


Procesamiento de strings

Análisis y diseño de algoritmos avanzados

Dra. Valentina Narváez Terán

Tecnológico de Monterrey

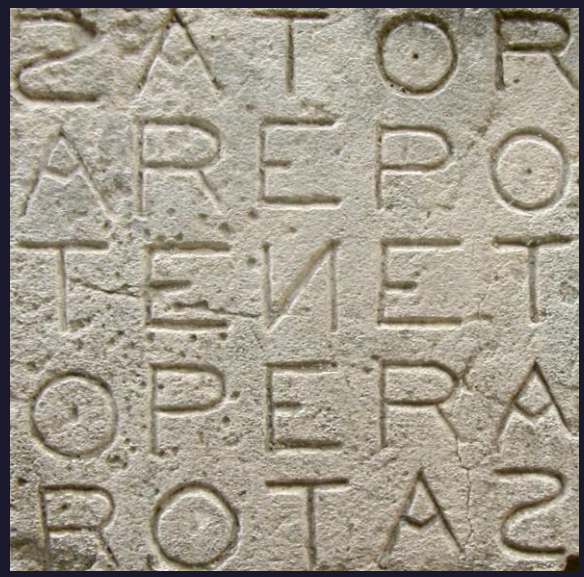
Problema: palíndromos mas largos

Los palíndromos son cadenas que se leen de la misma forma de izquierda a derecha, o viceversa

Palabras como: ojo, radar, reconocer, level, kayak, madam…

O frases:

- Anita lava la tina
- Somos o no somos
- Atar a la rata no es más que atar a la rata
- Was it a car or a cat I saw?



"sator arepo tenet opera rotas"
es un ejemplo en latín, encontrado en las ruinas de Pompeya

Problema: palíndromos mas largos

Algunos palíndromos contienen a otros palíndromos

s	o	m	o	s	o	n	o	s	o	m	o	s

s	o	m	o	s	o	n	o	s	o	m	o	s

s	o	m	o	s	o	n	o	s	o	m	o	s

s	o	m	o	s	o	n	o	s	o	m	o	s

Y también podemos encontrarlos como subcadenas de otras cadenas que no son palíndromos

a	n	a	b	a	n	a	n	a

¿Cuánto palíndromos s puedes encontrar en “anabanana”?

¿Cuál es el palíndromo mas largo?
¿Cómo encontrarlo?

Problema: palíndromos mas largos

Algunos palíndromos contienen a otros palíndromos

s	o	m	o	s	o	n	o	s	o	m	o	s

s	o	m	o	s	o	n	o	s	o	m	o	s

s	o	m	o	s	o	n	o	s	o	m	o	s

s	o	m	o	s	o	n	o	s	o	m	o	s

Y también podemos encontrarlos como subcadenas de otras cadenas que no son palíndromos

a	n	a	b	a	n	a	n	a

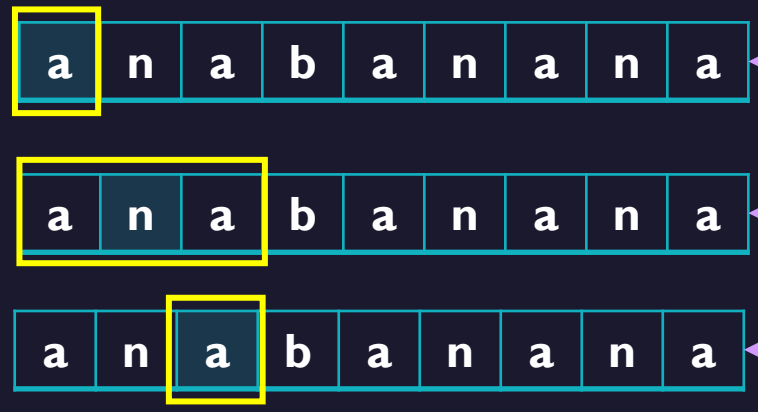
¿Cuánto palíndromos s puedes encontrar en “anabanana”?

ana, aba, naban, anana…

¿Cuál es el palíndromo mas largo?
¿Cómo encontrarlo?

El algoritmo de fuerza bruta

Un posible enfoque de fuerza bruta avanzaría, carácter por carácter, revisando cual es el palíndromo mas largo que se puede formar con ese carácter como centro



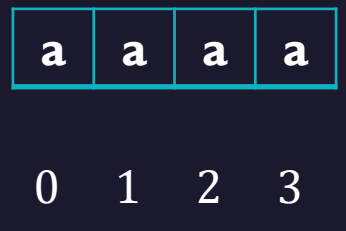
A partir de “a”, solo hay un palíndromo de longitud 1

A partir de “n”, hay un palíndromo de longitud 3

A partir de “a”, hay un palíndromo de longitud 1, etc, etc

Investiga ¿qué complejidad tiene este algoritmo?
¿Qué pasa con cadenas como “naan” ?

Algoritmo de Manacher



Un caso algo extremo… todas las subcadenas son parte de un palíndromo

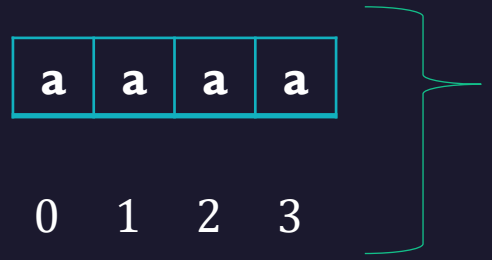
¿Cómo averiguamos que el mas largo es de longitud 4?

El algoritmo de Manacher consigue hacerlo, para cualquier cadena, en tiempo lineal $O(n)$

¿Cómo? Tomando ventaja del valor de elementos anteriores, y de la simetría de los palíndromos, para evitar calcular desde 0

¿A qué técnica de programación pertenece?

Algoritmo de Manacher



Un caso algo extremo… todas las subcadenas son parte de un palíndromo

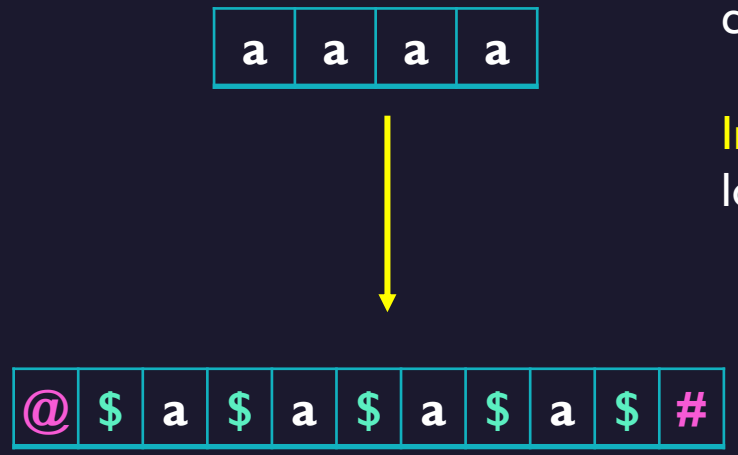
¿Cómo averiguamos que el mas largo es de longitud 4?

El algoritmo de Manacher consigue hacerlo, para cualquier cadena, en tiempo lineal $O(n)$

¿Cómo? Tomando ventaja del valor de elementos anteriores, y de la simetría de los palíndromos, para evitar calcular desde 0

¿A qué técnica de programación pertenece?
Programación dinámica

Algoritmo de Manacher



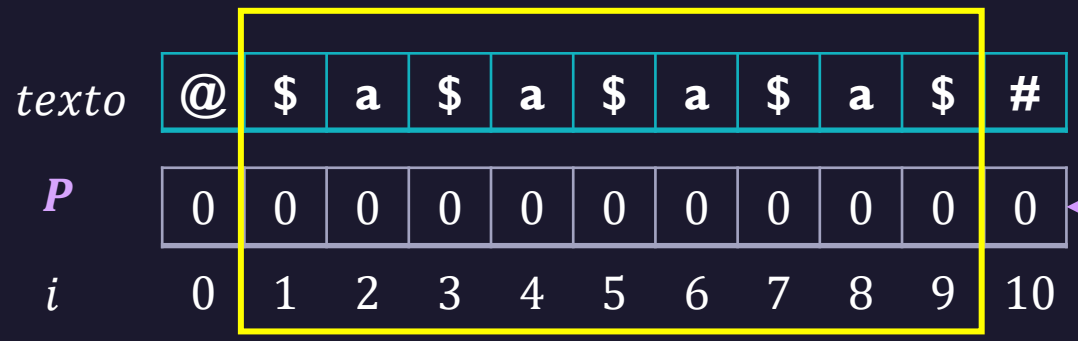
Primer paso:
Crear una nueva cadena con caracteres extra al inicio, al final y entre caracteres

Importante: los símbolos extra no deben ser parte del alfabeto original, y los caracteres de inicio y fin deben ser distintos

La nueva cadena tiene longitud $e = 2 * n + 3$

Esto es un trade-off de espacio por tiempo: usamos mas memoria para realizar menos pasos

Algoritmo de Manacher



El arreglo P contendrá la longitud del palíndromo que se forme, cuyo centro sea $texto[i]$

P tiene $e = 2 * n + 3$ elementos, y se inicializa en 0's

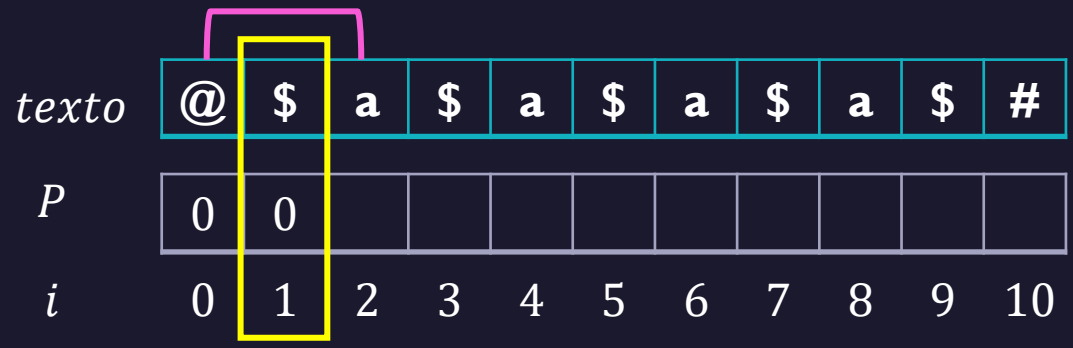
El calculo de P solo nos interesa de $i = 1$ a $e - 1$

Algoritmo de Manacher

```
centro = limite = 0

for i = 1 to e − 1
    if i < limite
        simetrica = 2 * centro − i
        P[i] = min(limite − i, P[simetrica])

    gap = P[i] + 1
    while texto[i − gap] == texto[i + gap]
        P[i] ++
        gap ++

    if i + P[i] > limite
        limite = i + P[i]
        centro = i
```

¿Cuál es el palíndromo mas grande con $texto[i]$ como centro?

¿Algo de lo procesado antes sirve para no calcularlo de cero?
No, porque no hemos encontrado ningún palíndromo aun



Entonces, revisamos si los dos caracteres antes y después de texto[i] son iguales

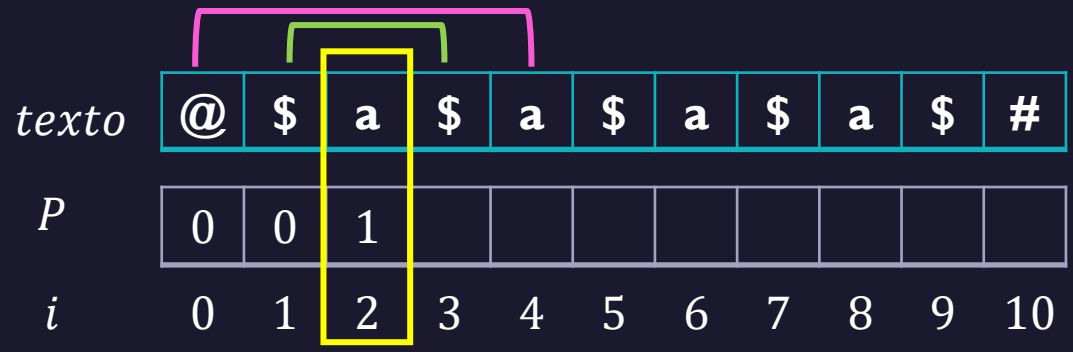
No lo son, así que nada mas ocurre.

Algoritmo de Manacher

```
centro = limite = 0

for i = 1 to e − 1
    if i < limite
        simetrica = 2 * centro − i
        P[i] = min(limite − i, P[simetrica])

    gap = P[i] + 1
    while texto[i − gap] == texto[i + gap]
        P[i] ++
        gap ++

    if i + P[i] > limite
        limite = i + P[i]
        centro = i
```

¿Cuál es el palíndromo mas grande con $texto[i]$ como centro?

¿Algo de lo procesado antes sirve para no calcularlo de cero?
Otra vez no. No hemos encontrado un palíndromo aun

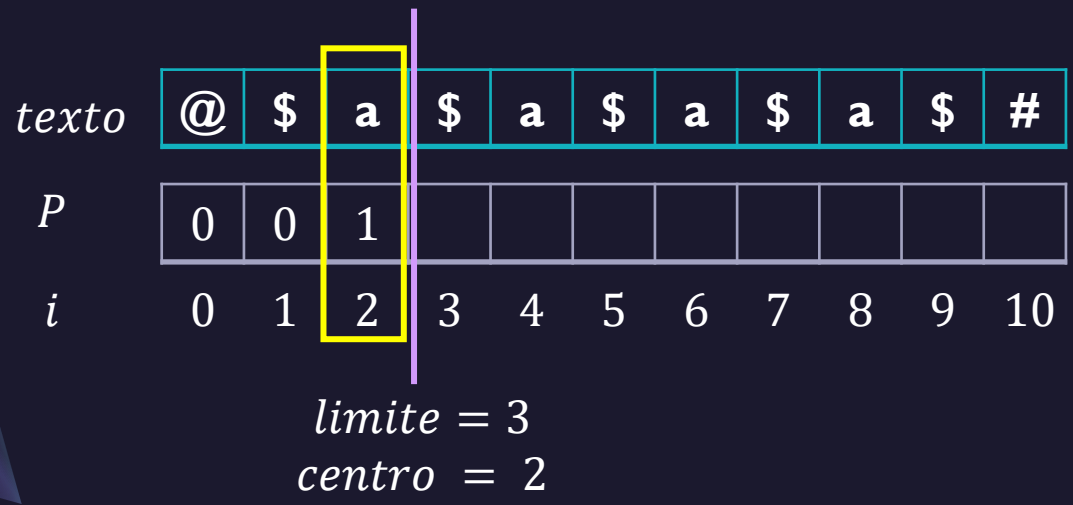


Revisamos si los caracteres antes y después de texto[i] son iguales. Lo son solo 1 vez.

Algoritmo de Manacher

```
centro = limite = 0

for i = 1 to e − 1
    if i < limite
        simetrica = 2 * centro − i
        P[i] = min(limite − i, P[simetrica])

    gap = P[i] + 1
    while texto[i − gap] == texto[i + gap]
        P[i] ++
        gap ++

    if i + P[i] > limite
        limite = i + P[i]
        centro = i
```

Ahora que encontramos un palíndromo, guardamos el *limite* (hasta donde llega) y cual fue su *centro*



Algoritmo de Manacher

$centro = limite = 0$

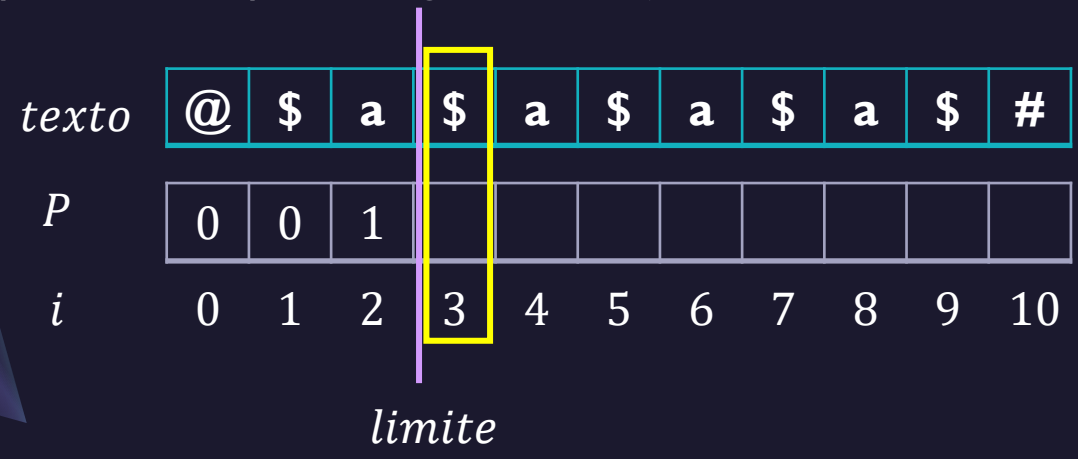
for $i = 1$ **to** $e - 1$
  if $i < limite$
    $simetrica = 2 * centro - i$
    $P[i] = \min(limite - i, P[simetrica])$

  $gap = P[i] + 1$
  while $texto[i - gap] == texto[i + gap]$
    $P[i] ++$
    $gap ++$

  if $i + P[i] > limite$
    $limite = i + P[i]$
    $centro = i$

¿El palíndromo anterior sirve de algo para el calculo de $P[3]$?

No, porque $i = 3$ no esta dentro del $limite$ del palíndromo que ha llegado mas lejos hasta ahora



Algoritmo de Manacher

$centro = limite = 0$

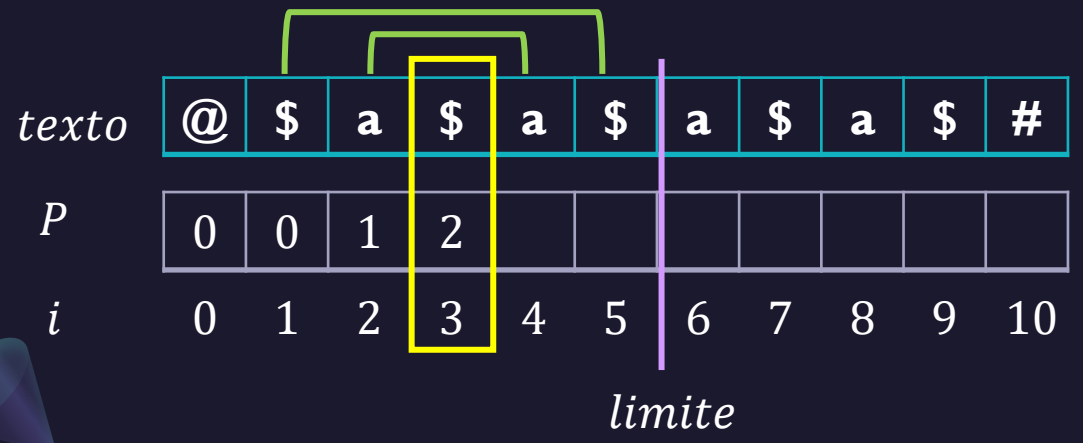
for $i = 1$ **to** $e - 1$
  if $i < limite$
    $simetrica = 2 * centro - i$
    $P[i] = \min(limite - i, P[simetrica])$

  $gap = P[i] + 1$
  while $texto[i - gap] == texto[i + gap]$
    $P[i] ++$
    $gap ++$

  if $i + P[i] > limite$
    $limite = i + P[i]$
    $centro = i$

Hay que hacer comparaciones desde cero (hay 2)

Este palíndromo llega mas lejos, así que actualizamos *limite* y *centro*

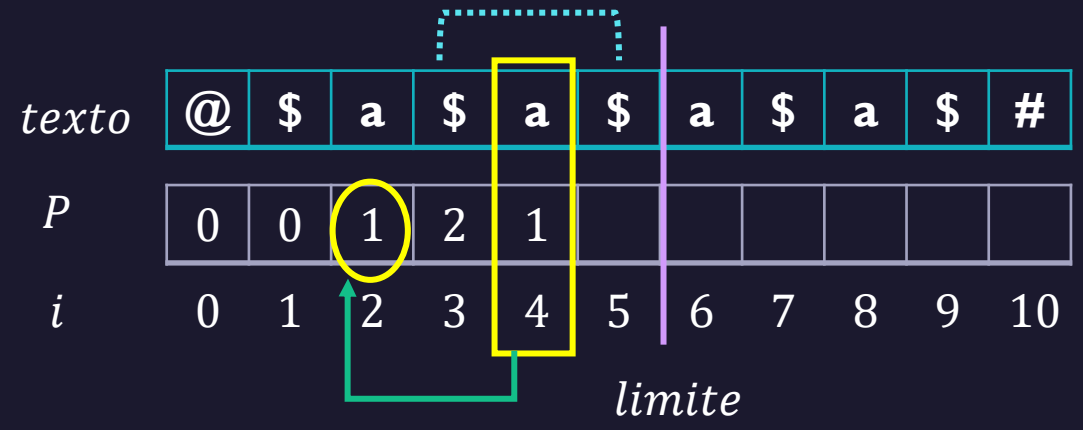


Algoritmo de Manacher

```
centro = limite = 0

for i = 1 to e − 1
    if i < limite
        simetrica = 2 * centro − i
        P[i] = min(limite − i, P[simetrica])

    gap = P[i] + 1
    while texto[i − gap] == texto[i + gap]
        P[i] ++
        gap ++

    if i + P[i] > limite
        limite = i + P[i]
        centro = i
```

En los palíndromos hay simetría: tiene que haber al menos las mismas coincidencias con respecto a la posición simétrica.

Para P[4]: El elemento i esta dentro del $limite$, podemos copiar el dato

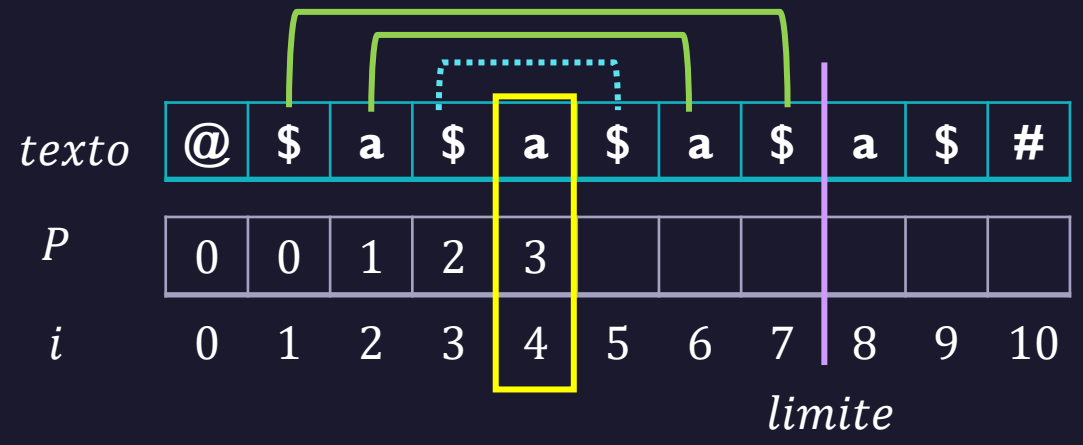


Así que copiamos el valor, sin tener que comparar las posiciones marcadas con la línea punteada

Algoritmo de Manacher

```
centro = limite = 0

for i = 1 to e − 1
    if i < limite
        simetrica = 2 * centro − i
        P[i] = min(limite − i, P[simetrica])

    gap = P[i] + 1
    while texto[i − gap] == texto[i + gap]
        P[i] ++
        gap ++

    if i + P[i] > limite
        limite = i + P[i]
        centro = i
```

Luego, seguimos buscando coincidencias.
En total, son 3 (1 copiada de $P[2]$ y 2 calculadas en el *while*)



El palíndromo llega mas allá del limite anterior,
limite y centro se actualizan

Algoritmo de Manacher

```
centro = limite = 0

for i = 1 to e − 1
    if i < limite
        simetrica = 2 * centro − i
        P[i] = min(limite − i, P[simetrica])

    gap = P[i] + 1
    while texto[i − gap] == texto[i + gap]
        P[i] ++
        gap ++

    if i + P[i] > limite
        limite = i + P[i]
        centro = i
```

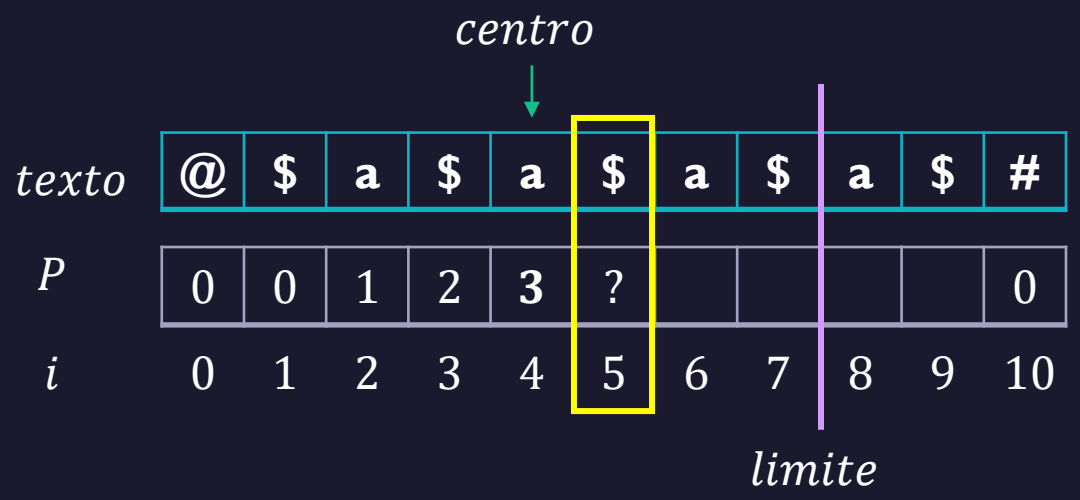
Revisa si i esta dentro del $limite$ del palíndromo que ha llegado mas lejos hasta ahora. Si lo esta, copia el dato de la posición $simétrica$ a i con respecto al $centro$

Sigue buscando coincidencias a partir de i, incrementando el gap

Al encontrar un palíndromo que sobrepase el $limite$, actualizamos $limite$ y $centro$

Algoritmo de Manacher

```
centro = limite = 0

for i = 1 to e − 1
    if i < limite
        simetrica = 2 * centro − i
        P[i] = min(limite − i, P[simetrica])

    gap = P[i] + 1
    while texto[i − gap] == texto[i + gap]
        P[i] ++
        gap ++

    if i + P[i] > limite
        limite = i + P[i]
        centro = i
```



Continua el ejemplo y programa el algoritmo.

Algoritmo de Manacher

```
centro = limite = 0

for i = 1 to e − 1
    if i < limite
        simetrica = 2 * centro − i
        P[i] = min(limite − i, P[simetrica])

    gap = P[i] + 1
    while texto[i − gap] == texto[i + gap]
        P[i] ++
        gap ++

    if i + P[i] > limite
        limite = i + P[i]
        centro = i
```

texto	@	$	a	$	a	$	a	$	a	$	#
P	0	0	1	2	**3**	?					0
i	0	1	2	3	4	5	6	7	8	9	10

Continua el ejemplo y programa el algoritmo.

Algoritmo de Manacher

$centro = limite = 0$

for $i = 1$ hasta $e - 1$
  if $i < limite$
    $simétrica = 2 * centro - i$
    $P[i] = \min(limite - i, P[simétrica])$

  $gap = P[i] + 1$
  while $texto[i - gap] == texto[i + gap]$
    $P[i] ++$
    $gap ++$

  if $i + P[i] > limite$
    $limite = i + P[i]$
    $centro = i$

El mayor palíndromo tiene longitud 4
Es fácil saberlo, consultando P

¿Pero cómo sabemos donde inicia en el texto original?
Usando el índice del valor máximo de P

$texto$	@	$	a	$	a	$	a	$	a	$	#
P	0	0	1	2	**3**	4	3	2	1	0	0
i	0	1	2	3	4	5	6	7	8	9	10

$maxIndex$

$inicio = \lfloor (maxIndex - P[maxIndex] - 1) / 2 \rfloor$

O sea, desde $inicio = 0$ de "aaaa"

Algoritmo de Manacher

```
centro = limite = 0

for i = 1 to e − 1
    if i < limite
        simetrica = 2 * centro − i
        P[i] = min(limite − i, P[simetrica])

    gap = P[i] + 1
    while texto[i − gap] == texto[i + gap]
        P[i] ++
        gap ++

    if i + P[i] > limite
        limite = i + P[i]
        centro = i
```

Otro tema: suffix Array

Para calcularlo, primero obtenemos todos los sufijos, y luego los ordenamos lexicográficamente

Ejemplo:

Los sufijos de "*anabanana*" son todas las subcadenas de i a n, con $0 \leq i < n$

i	$cadena[i:n]$
0	anabanana
1	nabanana
2	abanana
3	banana
4	anana
5	nana
6	ana
7	na
8	a

Ordenar

Suffix array de *anabanana*:
los índices de los sufijos ordenados

i	$cadena[i:]$
8	a
2	abanana
4	anana
	… etc.

Investiga e implementa
¿Complejidad?
¿Para que sirve?

Actividad 2.1: operaciones con strings

Para esta actividad implementa:

- Los algoritmos para string-matching de la clase anterior
  - KMP con LPS
  - Z-algorithm con Z-array
- Palíndromo mas largo
  - Algoritmo de Manacher
- Suffix array