Tecnológico de Monterrey.

Campus Querétaro.

TC2038. Análisis y diseño de Algoritmos A

M.C. Ramona Fuentes Valdéz

rfuentes@tec.mx

32

Técnicas de diseño de algoritmos

Manejo de strings

- texto T
- patrón P

Algoritmo Z

- El algoritmo Z se utiliza para encontrar la ocurrencia de un patrón en una cadena en tiempo lineal.
  - *Suponga que si la longitud de la cadena es n y el tamaño del patrón que se buscará es m, el tiempo necesario para resolverlo será del orden O(m + n).*
- Se crea una matriz Z, de la misma longitud que la cadena de texto T.
- Cada elemento de la matriz Z almacena la longitud de la subcadena más larga posible iniciando desde el carácter actual de la cadena original (T).
  - Para iniciar, el patrón P y el texto T se concatenan con un símbolo especial que no se encuentre presente dentro del texto y el patron, *por ejemplo:* $, obteniéndose la cadena PT.

Ejemplo:

Index:	0	1	2	3	4	5	6	7	8	9	10	11
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Z values:		1	0	0	3	1	0	0	2	2	1	0

string	a	a	a	b	a	a	a	b
posición	1	2	3	4	5	6	7	8
Valores Z	0	2	1	0	4	2	1	0

Z Algorithm Z algorithm Z algorithm (Linear time pattern searching Algorithm)

33

Técnicas de diseño de algoritmos

Manejo de strings

Algoritmo Z

- Dada una cadena S de longitud *n*, la función Z es un arreglo de longitud *n* donde el *i-ésimo* elemento es igual al máximo número de caracteres empezando desde la posición *i* que coincide con los primeros caracteres de *S*. En otras palabras, *Z[i]* es la longitud del prefijo común más largo entre *S* y el sufijo de *S* que empiezan en *i*, *S[i ..n]*.

  Ejemplo:

  - Dada la cadena S = “**aaabaaab**”, para cualquier *i > 1*, tenemos:

Posición 2:
S = “**aaabaaab**"
S[2..n] = “**aabaaab**"

Posición 3:
S = “**aaabaaab**"
S[3..n] = “**abaaab**"

Posición 4:
S = “**aaabaaab**"
S[4..n] = “**baaab**"

Algoritmo:

```
Z_FUNCTION
Input: S : String
  Z : Array[S.length]
  INIT(Z, 0)
  for i ← 1 to S.length do
    for j ← i to S.length do
      if S[i] = S[j - i] then
         Z[i] ← Z[i] + 1
      else
         break
      end if
     end for
  end for
return Z
```

string	a	a	a	b	a	a	a	b
Posición	1	2	3	4	5	6	7	8
Valores Z	0	2	1	0	4	2	1	0

Este algoritmo tiene una complejidad $O(n^2)$.

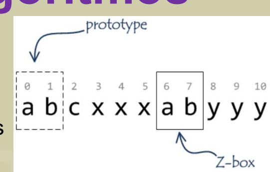
- Para obtener un algoritmo eficiente…
  Se calcularán los valores de Z[i] de 1 a (n - 1), pero al mismo tiempo, al calcular un nuevo valor, se intentará hacer el mejor uso posible de los valores previamente calculados.
  - Recuerda que se está buscando el máximo prefijo.
    - ¿Qué técnica podríamos emplear para este algoritmo?

Z-function and its calculation

Técnicas de diseño de algoritmos

Manejo de strings

Algoritmo Z



Implementación eficiente

- Se llamará “coincidencias de segmento” a las subcadenas que coinciden con el prefijo de S.

  Ejemplo:

  El valor de Z[i] es la longitud de la coincidencia de segmento que comienza en la posición *i* y que termina en la posición *i* + Z[i] − 1.

- Para hacer esto, mantendremos los índices [l, r] del segmento coincidente que se encuentra más hacia la derecha.
  - Es decir, entre todos los segmentos detectados mantendremos el que termina más a la derecha.
  - En cierto modo, el índice *r* puede verse como el límite de lo que se ha registrado. Todo lo que está más allá aún no se conoce.

- Para el índice actual, *i*, tenemos:
  - Si *i* > *r*: La posición está fuera de lo que ya hemos procesado.
  - Si *i* ≤ *r*, la posición está dentro del segmento de coincidencias actual [l , r].

Z-function and its calculation

Algoritmos TC2038, https://github.com/

Técnicas de diseño de algoritmos

Manejo de strings

Algoritmo Z

Implementación eficiente

- Para el índice actual, *i*, tenemos:
  - Si *i* > *r*: La posición está fuera de lo que ya hemos procesado.
    - Entonces, se utilizará la comparación carácter a carácter del algoritmo ingenuo.
    - Sin embargo, al final, si Z[i] > 0, se tendrán que actualizar los índices del segmento más a la derecha, porque el nuevo r = i + Z[i] − 1 es menor que el *r* anterior.
  - Si *i* ≤ *r*, la posición está dentro del segmento de coincidencias actual [l , r].
    - Entonces se pueden usar los valores Z ya calculados para inicializar el valor de Z[i] a algo (*mejor que comenzar desde cero*), tal vez incluso un número más grande. Para ello, se observa si las subcadenas S[l ..r] y S[0..(r − l)] coinciden. Eso significa que, como aproximación inicial para Z[i], podemos tomar el valor ya calculado del segmento correspondiente S[0..(r − l)], que es Z[i − l].
    - Sin embargo, el valor Z[i − l] podría ser demasiado grande: cuando se aplica a la posición *i*, ya que podría exceder el índice *r*. Esto no está permitido porque no sabemos nada sobre los caracteres a la derecha de *r* . Por lo tanto, como una buena aproximación, podemos decir que Z[i] = min(r − i + 1, Z[i − l]). Ahora solo falta calcular las posiciones que se encuentran a la derecha de *r*, y para ello se usa el algoritmo previo.

Técnicas de diseño de algoritmos

Manejo de strings

Algoritmo Z

Implementación eficiente

Algoritmo

```
Z_FUNCTION_DP
Input: S : String
  Z : Array[S.length]
  left ← 1, right ← 1, INIT(Z, 0)
  for i ← 2 to S.length do
    if i ≤ right then
      Z[i] ← MIN(right - i + 1, Z[i - left])
    end if
    while i + Z[i] ≤ S.length and S[Z[i]] = S[i + Z[i]]
    do
      Z[i] ← Z[i] + 1
    end while
    if i + Z[i] - 1 > right then
      left ← i,
      right ← i + Z[i] - 1
    end if
  end for
return Z
```

string	a	a	a	b	a	a	a	b
Posición	1	2	3	4	5	6	7	8
Valores Z	0	2	1	0	4	2	1	0

Ejemplo: **T = {aaabaaab}**

k=1, r=0, l=0.

Index:	0	1	2	3	4	5	6	7
Text:	a	a	a	b	a	a	a	b
Text:	a	a	a	b	a	a	a	b
Z values:								

Evaluating k=1.
Current index (k=1) is past end of last Z-box (r=0), so Z(1) must be computed explicitly.

Index:	0	1	2	3	4	5	6	7
Text:	a	a	a	b	a	a	a	b
Text:	a	a	a	b	a	a	a	b
Z values:		2						

Z(1) = 2, l=1, r=2, 3 comparisons so far.

Evaluating k=2.
Current index (k=2) is in a previously discovered substring (r=2).
The previously discovered substring starts at k-l (2-1), with Z(1)=2 which is not < the remaining substring S[2..2], so additional comparisons are needed.

Index:	0	1	2	3	4	5	6	7
Text:	a	a	a	b	a	a	a	b
Text:	a	a	a	b	a	a	a	b
Z values:		2	1					

Z(2) = 1, l=2, r=2, 4 comparisons so far.

Técnicas de diseño de algoritmos

Manejo de strings

Algoritmo Z

Ejemplo: T = {aaabaaab}

Evaluating k=3.
Current index (k=3) is past end of last Z-box (r=2), so Z(3) must be computed explicitly.

Index:	0	1	2	3	4	5	6	7
Text:	a	a	a	b	a	a	a	b
Text:	a	a	a	b	a	a	a	b
Z values:		2	1	0				

Z(3) = 0, l=2, r=2, 5 comparisons so far.

Evaluating k=4.
Current index (k=4) is past end of last Z-box (r=2), so Z(4) must be computed explicitly.

Index:	0	1	2	3	4	5	6	7
Text:	a	a	a	b	a	a	a	b
Text:	a	a	a	b	a	a	a	b
Z values:		2	1	0	4			

Z(4) = 4, l=4, r=7, 10 comparisons so far.

Evaluating k=5.
Current index (k=5) is in a previously discovered substring (r=7).
The previously discovered substring starts at k-l (5-4), with Z(1)=2 which is < the remaining substring S[5..7], so Z(5)=Z(1)=2.

Index:	0	1	2	3	4	5	6	7
Text:	a	a	a	b	a	a	a	b
Text:	a	a	a	b	a	a	a	b
Z values:		2	1	0	4	2		

Z(5) = 2, l=4, r=7, 10 comparisons so far.

Evaluating k=6.
Current index (k=6) is in a previously discovered substring (r=7).
The previously discovered substring starts at k-l (6-4), with Z(2)=1 which is < the remaining substring S[6..7], so Z(6)=Z(2)=1.
No additional comparisons needed.

Index:	0	1	2	3	4	5	6	7
Text:	a	a	a	b	a	a	a	b
Text:	a	a	a	b	a	a	a	b
Z values:		2	1	0	4	2	1	

Z(6) = 1, l=4, r=7, 10 comparisons so far.

Evaluating k=7.
Current index (k=7) is in a previously discovered substring (r=7).
The previously discovered substring starts at k-l (7-4), with Z(3)=0 which is < the remaining substring S[7..7], so Z(7)=Z(3)=0.

Index:	0	1	2	3	4	5	6	7
Text:	a	a	a	b	a	a	a	b
Text:	a	a	a	b	a	a	a	b
Z values:		2	1	0	4	2	1	0

Z(7) = 0, l=4, r=7, 10 comparisons so far.

The Z algorithm has completed with 10 comparisons on a text of size 8.

Index:	0	1	2	3	4	5	6	7
Text:	a	a	a	b	a	a	a	b
Z values:		2	1	0	4	2	1	0

Z Algorithm. https://personal.utdallas.edu/~besp/demo/John2010/z-algorithm.htm

Técnicas de diseño de algoritmos

Manejo de strings

Algoritmo Z

Ejemplo: T = {aabcaabxaaaz}

k=1, r=0, l=0.

Index:	0	1	2	3	4	5	6	7	8	9	10	11
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Z values:												

Evaluating k=1.
Current index (k=1) is past end of last Z-box (r=0), so Z(1) must be computed explicitly.

Index:	0	1	2	3	4	5	6	7	8	9	10	11
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Z values:		1										

Z(1) = 1, l=1, r=1, 2 comparisons so far.

Evaluating k=2.
Current index (k=2) is past end of last Z-box (r=1), so Z(2) must be computed explicitly.

Index:	0	1	2	3	4	5	6	7	8	9	10	11
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Z values:		1	0									

Z(2) = 0, l=1, r=1, 3 comparisons so far.

Evaluating k=3.
Current index (k=3) is past end of last Z-box (r=1), so Z(3) must be computed explicitly.

Index:	0	1	2	3	4	5	6	7	8	9	10	11
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Z values:		1	0	0								

Z(3) = 0, l=1, r=1, 4 comparisons so far.

Evaluating k=4.
Current index (k=4) is past end of last Z-box (r=1), so Z(4) must be computed explicitly.

Index:	0	1	2	3	4	5	6	7	8	9	10	11
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Z values:		1	0	0	3							

Z(4) = 3, l=4, r=6, 8 comparisons so far.

Evaluating k=5.
Current index (k=5) is in a previously discovered substring (r=6).
The previously discovered substring starts at k-l (5-4), with Z(1)=1 which is < the remaining substring S[5..6], so Z(5)=Z(1)=1.
No additional comparisons needed.

Index:	0	1	2	3	4	5	6	7	8	9	10	11
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Z values:		1	0	0	3	1						

Z(5) = 1, l=4, r=6, 8 comparisons so far.

Z Algorithm. https://personal.utdallas.edu/~besp/demo/John2010/z-algorithm.htm

Técnicas de diseño de algoritmos

Manejo de strings

Algoritmo Z

Ejemplo: **T = {aabcaabxaaaz}**

Evaluating k=6.
Current index (k=6) is in a previously discovered substring (r=6).
The previously discovered substring starts at k-l (6-4), with Z(2)=0 which is < the remaining substring S[6..6], so Z(6)=Z(2)=0.
No additional comparisons needed.

Index:	0	1	2	3	4	5	6	7	8	9	10	11
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Z values:		1	0	0	3	1	0					

Z(6) = 0, l=4, r=6, 8 comparisons so far.

Evaluating k=7.
Current index (k=7) is past end of last Z-box (r=6), so Z(7) must be computed explicitly.

Index:	0	1	2	3	4	5	6	7	8	9	10	11
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Z values:		1	0	0	3	1	0	0				

Z(7) = 0, l=4, r=6, 9 comparisons so far.

Evaluating k=8.
Current index (k=8) is past end of last Z-box (r=6), so Z(8) must be computed explicitly.

Index:	0	1	2	3	4	5	6	7	8	9	10	11
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Z values:		1	0	0	3	1	0	0	2			

Z(8) = 2, l=8, r=9, 12 comparisons so far.

Evaluating k=9.
Current index (k=9) is in a previously discovered substring (r=9).
The previously discovered substring starts at k-l (9-8), with Z(1)=2 which is not < the remaining substring S[9..9], so additional comparisons are needed.

Index:	0	1	2	3	4	5	6	7	8	9	10	11
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Z values:		1	0	0	3	1	0	0	2	2		

Z(9) = 2, l=9, r=10, 14 comparisons so far.

Evaluating k=10.
Current index (k=10) is in a previously discovered substring (r=10).
The previously discovered substring starts at k-l (10-9), with Z(1)=2 which is not < the remaining substring S[10..10], so additional comparisons are needed.

Index:	0	1	2	3	4	5	6	7	8	9	10	11
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Z values:		1	0	0	3	1	0	0	2	2	1	

Z(10) = 1, l=10, r=10, 15 comparisons so far.

Evaluating k=11.
Current index (k=11) is past end of last Z-box (r=10), so Z(11) must be computed explicitly.

Index:	0	1	2	3	4	5	6	7	8	9	10	11
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Text:	a	a	b	c	a	a	b	x	a	a	a	z
Z values:		1	0	0	3	1	0	0	2	2	1	0

Z(11) = 0, l=10, r=10, 16 comparisons so far.

The Z algorithm has completed with 16 comparisons on a text of size 12.

Z Algorithm, https://personal.utdallas.edu/~besp/demo/John2010/z-algorithm.htm

40

Técnicas de diseño de algoritmos

Manejo de strings

Algoritmo Z

Aplicaciones

- Búsqueda de una subcadena (o patrón).
- Número de subcadenas distintas en una cadena.
- Comprensión de cadenas.

Z-function and its calculation

41

Técnicas de diseño de algoritmos

Manejo de strings

- texto T
- patrón P

Algoritmo Z

Coincidencia de un patrón

- El problema de la coincidencia de un patrón en un texto (Pattern Matching Problem) también conocido como búsqueda de una subcadena en un texto (Substring Search), se define de la siguiente forma:
  - Dada una cadena P, llamada patrón (*pattern*) y una cadena más grande T llamada el texto, encontrar la primera ocurrencia del patrón P en el texto T y regresar la posición del texto T donde inicial el patrón o *-1* en caso de que el patrón no exista en el texto T.

Ejemplo:

- Para T = "bacacabcaca", P = "aca"
  - El patrón P se encuentra en T iniciando en las posiciones **2**, 4 y 9.

Fuerza bruta

Ir moviendo a la derecha (*shifting*) una ventana que contiene el patrón, llamada ventana de deslizamiento (*sliding windows*), casilla por casilla sobre la cadena T y, en cuanto se encuentre la coincidencia, se regresa el número de la casilla de inicio.

Alternativa

```
PATTERN_MATCHING
Input: P, T : String
  for i ← 1 to T.length - P.length do
    if P = T[i..(i + P.length)] then
      return i
    end if
  end for
return -1
```

Este algoritmo tiene una complejidad O(nm).

Existen varias formas de mejorar este algoritmo a tiempo lineal. Una de ellas es con la función Z se ejecuta en tiempo lineal. Para lograrlo, basta con concatenar el patrón al inicio del texto, separados por un carácter que no esté en el texto. Normalmente se utiliza el signo de pesos ($).

Z-function and its calculation Algoritmos TC2038, https://github.com/

42

Técnicas de diseño de algoritmos

Manejo de strings

- texto T
- patrón P

Algoritmo Z

- El algoritmo Z se utiliza para encontrar la ocurrencia de un patrón en una cadena en tiempo lineal.
  - *Suponga que si la longitud de la cadena es n y el tamaño del patrón que se buscará es m, el tiempo necesario para resolverlo será del orden O(m + n).*
- Se crea una matriz Z, de la misma longitud que la cadena de texto T.
- Cada elemento de la matriz Z almacena la longitud de la subcadena más larga posible iniciando desde el carácter actual de la cadena original (T).
  - Para iniciar, el patrón P y el texto T se concatenan con un símbolo especial que no se encuentre presente dentro del texto y el patron, *por ejemplo:* $, obteniéndose la cadena PT = P $ T.

Ejemplo:

- Para T = "bacacabcaca", P = "aca" ❭ PT = "aca$bacacabcaca"
  - El patrón P se encuentra en T iniciando en las posiciones **2**, 4 y 9.

string	a	c	a	$	b	a	c	a	c	a	b	c	a	c	a
posición	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Valores Z	0	0	1	0	0	3	0	3	0	1	0	0	3	0	1

Complejidad

- Tiene una complejidad de O(n)
- El tiempo que se toma para poder resolver el algoritmo es de orden O(m+n)

20 comparaciones se realizaron para generar los valores de Z.

Z Algorithm Exact Pattern Match,

Z Algorithm Z algorithm Z algorithm (Linear time pattern searching Algorithm)

43

Técnicas de diseño de algoritmos

Manejo de strings

"Yo dono rosas, oro no doy"
"Isaac no ronca así"
"Lavan esa base naval"
"No traces en ese cartón"
"¿Será lodo o dólares?"

Palíndromo

Conceptos básicos

- Un palíndromo es una cadena que se lee igual de izquierda a derecha que de derecha a izquierda.
- En algunas aplicaciones es necesario encontrar las subcadenas que son palíndromos, dentro de un *string* dado, o el palíndromo más grande que exista como subcadena de una cadena dada. s.

El problema…

- Dada una cadena *S*, se desea encontrar una subcadena que forme un palíndromo y que sea el más largo que se pueda encontrar, es decir, el que está formado con el mayor número de caracteres.

Ejemplo

Si T = {abaccab}

Subcadenas
- aba
- acca
- baccab

Técnicas de diseño de algoritmos

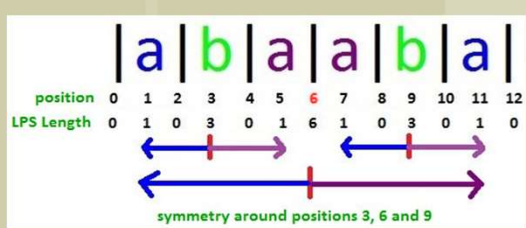
Manejo de strings

"Yo dono rosas, oro no doy"
"Isaac no ronca así"
"Lavan esa base naval"
"No traces en ese cartón"
"¿Será lodo o dólares?"

Palíndromo

Eficiencia

- La solución trivial puede ser tomar la letra inicial y la longitud de la palabra y verificar si es una palíndromo.
  - Esta solución es $O(n^3)$.
- Un mejor enfoque es fijar el centro de una palabra y expandir a ambos lados aumentando un contador cada vez que coinciden dos letras.
  - Este enfoque tiene una complejidad temporal de $O(n^2)$.
- El algoritmo de Manacher lo reduce a $O(n)$.
  - La razón de esta mejora es que el algoritmo de Manacher aprovecha las propiedades de una cadena palindrómica para evitar cálculos innecesarios.



Técnicas de diseño de algoritmos

Manejo de strings

Palíndromo

Fuerza bruta

- **Algoritmo**
  - Se recorren todas las posiciones, *i* , de la cadena, y para cada una de ellas se trata de expandir hacia la izquierda y a la derecha hasta encontrar la subcadena más grande, lo cual se debe hacer tanto para cadenas de longitud impar como para longitud par.
  - Complejidad O(n^2).

```
LONGEST_PALINDROME
Input: S : String
  maxLong ← 1
  startAt ← 0
  for i ← 1 to S.length do
    /* LONGITUD IMPAR */
    /* LONGITUD PAR */
  end for
return PAIR(startAt, maxLong)
```

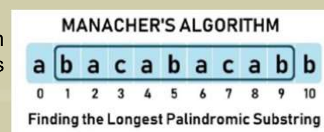
```
/* LONGITUD IMPAR */
j ← 1
while (i − j) ≥ 1 and (i + j) ≤ n and S[i − j] = S[i + j] do
  j ← j + 1
end while
if (2 ∗ j) − 1 > maxLong then
  maxLong ← (2 ∗ j) + 1
  startAt ← i − j + 1
end if
```

```
/* LONGITUD PAR */
j ← 1
while (i − j − 1) ≥ 1 and (i + j) ≤ n and S[i − j − 1] = S[i + j] do
  j ← j + 1
end while
if (2 ∗ j) > maxLong then
  maxLong ← 2 ∗ j
  startAt ← i − j
end if
```


Técnicas de diseño de algoritmos

Manejo de strings

Manacher (palíndromo más largo)



- Este algoritmo, propuesto por Glenn K. Manacher en 1975, logra resolver el problema del palíndromo más largo de una cadena *S*, en una forma muy eficiente, con una complejidad O(n), proporcional a la longitud *n* de la cadena.
- Al igual que el algoritmo de fuerza bruta, éste se basa en buscar un palíndromo centrado en *i*, con algunas diferencias:
  - En el algoritmo de fuerza bruta, se considera si la cadena es longitud par o impar. Por otra parte, el algoritmo de Manacher se basa en una **nueva cadena**, T, a la que *se le agregan posiciones intermedias*, una al inicio y otra al final de la cadena, lo que garantiza que todo el proceso se realizará sobre una cadena de longitud impar.
  - Los valores calculados se almacenan en un arreglo, L, que tiene la longitud de la nueva cadena.
  - Se utilizan los valores previamente calculados (a la izquierda de *i*) que están almacenados en L, para calcular los valores que siguen (a la derecha de *i*).

Técnicas de diseño de algoritmos

Manejo de strings

Manacher (palíndromo más largo)

Algoritmo

1) Se inicia con la construcción de la nueva cadena, *T*, agregando un carácter especial en medio de cada uno de los caracteres de la cadena original, además de uno al inicio y otro al final.
2) El carácter que se agrega puede ser cualquiera que no pertenezca al alfabeto de la cadena analizada.
   Ejemplo:
   - Cadena S = baab, de longitud 4
     - al agregar el carácter "|", la nueva cadena sería T = |b|a|a|b|, la cual tiene longitud de 9.
   - Cadena S = bacab, de longitud 5
     - al agregar el carácter "|", la nueva cadena sería T = |b|a|c|a|b|, la cual tiene longitud de 11.
3) Al finalizar el algoritmo, el valor en la posición *i* del arreglo L indica la longitud, en número de caracteres, que tiene el palíndromo más grande centrado en *i*, medido a la derecha o a la izquierda de *i*

<table>
<tr><td>Posición</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td></tr>
<tr><td>String</td><td>|</td><td>c</td><td>|</td><td>a</td><td>|</td><td>c</td><td>|</td><td>b</td><td>|</td><td>c</td><td>|</td><td>a</td><td>|</td><td>c</td><td>|</td></tr>
<tr><td>L</td><td>0</td><td>1</td><td>0</td><td>3</td><td>0</td><td>1</td><td>0</td><td>7</td><td>0</td><td>1</td><td>0</td><td>3</td><td>0</td><td>1</td><td>0</td></tr>
</table>

Técnicas de diseño de algoritmos

Manejo de strings

Manacher (palíndromo más largo)

Algoritmo

4) Manacher se dio cuenta que la propiedad de simetría de un palíndromo podría ayudar en el cálculo de L aprovechando los valores que ya se habían calculado anteriormente.
   - Sin embargo, las condiciones que se deben cumplir para poder disminuir la cantidad de cálculos no son triviales.

<table>
<tr><td>Posición</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td></tr>
<tr><td>String</td><td>|</td><td>x</td><td>|</td><td>x</td><td>|</td><td>x</td><td>|</td><td>x</td><td>|</td><td>x</td><td>|</td><td>x</td><td>|</td><td>x</td><td>|</td></tr>
<tr><td></td><td></td><td></td><td></td><td>Li</td><td></td><td></td><td></td><td>C</td><td></td><td></td><td></td><td>Ri</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>L[C]=D</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

Notación:
- C: La última posición calculada de L. Todos los valores en posiciones
- menores o iguales en C en L se conocen.
- Ri : Posición a la derecha de C a la cuál se le quiere calcular su valor L.
- Li : Posición a la izquierda de C que está separada exactamente el mismo
- número de caracteres de C que Ri . Esto implica que Ri – C = C – Li .

Técnicas de diseño de algoritmos

Manejo de strings

Manacher (palíndromo más largo)

Algoritmo

```
Input: S : String
  Q ← EXPAND(S)
  P ← Array(Q.length)
  center ← 1, right ← 1
  for i ← 2 to Q.length do
    iMirror ← center − (i − center)
    if right > i then
      P[i] ← min(right − i, P[iMirror])
    end if
    while Q[i + 1 + P[i]] = Q[i − 1 − P[i]] do
      P[i] ← P[i] + 1
    end while
    if i + P[i] > right then
      center ← i
      right ← i + P[i]
    end if
  end for
```

```
  maxPalindrome ← 0
  centerIndex ← 0
  for i ← 2 to Q.length do
    if maxPalindrome < P[i] then
      maxPalindrome ← P[i]
      centerIndex ← i
    end if
  end for
  return PAIR(((centerIndex − 1 − maxPlaindrome)/2), maxPalindrome)
```


```
Input: S : String
  Q ← EXPAND(S)
  P ← Array(Q.length)
  center ← 1, right ← 1
  for i ← 2 to Q.length do
    iMirror ← center − (i − center)
    if right > i then
      P[i] ← min(right − i, P[iMirror])
    end if
    while Q[i + 1 + P[i]] = Q[i − 1 − P[i]] do
      P[i] ← P[i] + 1
    end while
    if i + P[i] > right then
      center ← i
      right ← i + P[i]
    end if
  end for
```


Técnicas de diseño de algoritmos

Manejo de strings

Palíndromo

"Yo dono rosas, oro no doy"

"Isaac no ronca así"

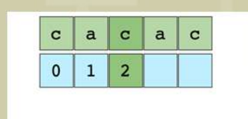
"Lavan esa base naval"

"No traces en ese cartón"

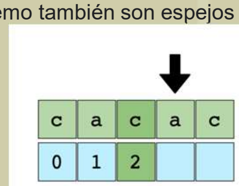
"¿Será todo o dólares?"

Ejemplo:

- Considera una cadena que contiene una subcadena palindrómica de longitud impar centrada alrededor del índice *c*. La mitad de los caracteres de la subcadena están a la derecha de *c* y la otra mitad a la izquierda.



- Consideremos la *a* señalada por la flecha, sería igual que la primera *a*.
- Esto se debe a que ya sabemos que la cadena anterior es un palíndromo alrededor de c y, por lo tanto, todos los caracteres en cualquier dirección son simétricos.
- Llamamos a estas dos a ser el espejo el uno del otro. De manera similar, ambas C en cada extremo también son espejos de cada uno.



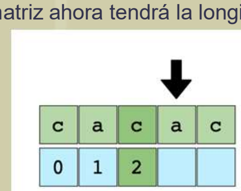
Técnicas de diseño de algoritmos

Manejo de strings

Palíndromo

Ejemplo:

- Ahora bien, no podemos decir que el palíndromo alrededor de *a*, señalado por la flecha, será el mismo que su espejo *a*, ya que el palíndromo del espejo *a* excede el rango cubierto por la subcadena palindrómica *cacac*.
- El palíndromo alrededor de *a*, señalado por la flecha, sería válido siempre que esté en el rango de *cacac*.
- Esto significa que, aunque el palíndromo alrededor del espejo *a* es más grande y excede el rango de *cacac*, la parte que se encuentra dentro del rango todavía estaría en el palíndromo del otro *a*.
- En este caso, se supuso que el palíndromo sería impar
- Si es par, transforma la cadena colocando un # inmediatamente antes y después de cada carácter.
  - Ejemplo, la palabra *deed* se convierte en #d#e#e#d#
  - La palabra *madam* se convierte en #m#a#d#a#m#.
- Esto también tiene otro beneficio; la matriz ahora tendrá la longitud del palíndromo alrededor de cada centro.



Técnicas de diseño de algoritmos

Manejo de strings

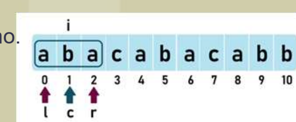
Manacher (palíndromo más largo)

Notación

- C: Es el centro del palindrome más grande que se ha encontrado hasta este momento.
- R : Posición a la derecha de C a la cuál se le quiere calcular su valor L.
- L : Posición a la izquierda de C que está separada exactamente el mismo número de caracteres de C que Ri . Esto implica que Ri – C = C – Li .

Ejemplo

Sea c el centro del palíndromo de mayor longitud que se ha encontrado hasta ahora.
Sean l y r los límites izquierdo y derecho de este palíndromo.
De izquierda a derecha, cuando i está en el índice 1, la subcadena palindrómica más larga es "aba" (longitud = 3)



La respuesta para la cadena dada es 9 cuando el palíndromo está centrado en el índice 5; c, l y r son los siguientes.

Técnicas de diseño de algoritmos

Manejo de strings

Manacher (palíndromo más largo)

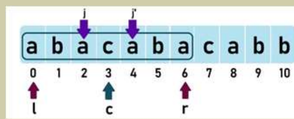
Mirror (espejo)

- C: Es el centro del palindrome más grande que se ha encontrado hasta este momento.
- R : Posición a la derecha de C a la cuál se le quiere calcular su valor L.
- L : Posición a la izquierda de C que está separada exactamente el mismo número de caracteres de C que Ri . Esto implica que Ri – C = C – Li .

Ejemplo

Índice de espejo: Para cualquier palíndromo centrado en c, el espejo del índice j es j', tal que: *j' = (2*c) - j*

Para palíndromo "abacaba" el espejo de j es j' y el espejo de j' es j



Técnicas de diseño de algoritmos

Manejo de strings

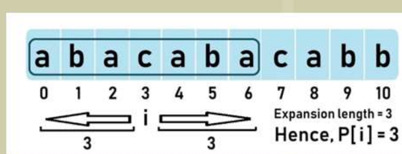
Manacher (palíndromo más largo)

Expandiendo

- C: Es el centro del palindrome más grande que se ha encontrado hasta este momento.
- R : Posición a la derecha de C a la cuál se le quiere calcular su valor L.
- L : Posición a la izquierda de C que está separada exactamente el mismo número de caracteres de C que Ri . Esto implica que Ri – C = C – Li .

Ejemplo

Al mover *i* de izquierda a derecha, intentamos expandir el palíndromo en cada *i*. ¿Hay un palíndromo centrado en i? y si es así, almacene la longitud de expansión hacia la izquierda o hacia la derecha en una matriz llamada P (a veces LPS).



Técnicas de diseño de algoritmos

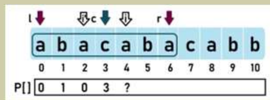
Manejo de strings

Manacher (palíndromo más largo)

Expandiendo

Ejemplo

- Considera este ejemplo, palíndromo "abacaba" centrado en *i= 3*



- La matriz P[] almacena la longitud de expansión del palíndromo centrado en cada índice.
- Pero no necesitamos ir manualmente a cada índice y expandirlo para verificar la longitud de la expansión cada vez.
- Aquí es exactamente donde el algoritmo de Manacher se optimiza mejor que la fuerza bruta, utilizando algunos conocimientos sobre cómo funcionan los palíndromos.
- Cuando i = 4, el índice está dentro del alcance del palíndromo más largo actual, es decir, i < r.
- Entonces, en lugar de expandir ingenuamente en *i*, queremos saber la longitud mínima de expansión que ciertamente es posible en *i*, de modo que podamos expandir ese P[i] mínimo y ver, en lugar de hacerlo desde el principio. Entonces, buscamos el espejo *i'*

Técnicas de diseño de algoritmos

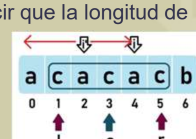
Manejo de strings

Manacher (palíndromo más largo)

Expandiendo

Ejemplo

- Si el índice i está dentro del rango
  - Mientras el palíndromo en el índice i' NO se expanda más allá del límite izquierdo (l) del palíndromo más largo actual, podemos decir que la longitud de expansión mínima ciertamente posible en i es P[i'].
  - En este caso, P[4]=P[2]=0
- Si el índice está fuera del rango
  - Si el palíndromo en el índice i' se expande más allá del límite izquierdo (l) del palíndromo más largo actual, podemos decir que la longitud de expansión mínima ciertamente posible en i es r-i.
  - Ejemplo: "acacacb"
  - P[4]=5-4 = 1



- Lo único que queda es expandir en *i* después de P[i], por lo que se verifican los caracteres del índice (P[i] + 1) y seguimos expandiendo en i.
- Si este palíndromo se expande más allá de r, actualice c a i y r a (i + P[i]).
- La matriz P[] ahora está llena y el valor máximo en esta matriz nos da la subcadena palindrómica más larga en la cadena dada.

Técnicas de diseño de algoritmos

Manejo de strings

Manacher (palíndromo más largo)

Pseudocódigo

```
R = -1
p = -1
for i = 0 to n-1:
   if i <= R:  A[i] = min(A[2*p - i], R-i)
   else:       A[i] = 0
   while S[i-A[i]-1] == S[i+A[i]+1]:
      A[i] = A[i] + 1
   if i+A[i] > R:
      R = i+A[i], p = i
```

Complejidad

- Time complexity *O(n)*
- Space complexity *O(n)*