Semana 4

Pedro O. Pérez M., PhD.

Análisis y diseño de algoritmos avanzados
Tecnológico de Monterrey

pperezm@tec.mx

08-2021

Palíndromo más largo

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 un cadena dada. El problema del palíndromo más largo y se define así:

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

Algoritmo de fuerza bruta

La forma más simple de resolver el problema del palíndromo más largo de una cadena S es usar fuerza bruta, en donde 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 $O(n^2)$.

Procedure 1 LONGEST_PALINDROME

Input: S : String

```
maxLong ← 1
startAt ← 0
for i ← 1 to S.length do
    /* LONGITUD IMPAR */
    NEXT_SLIDE
    /* LONGITUD PAR */
    NEXT_SLIDE
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
```

Algoritmo de Manacher

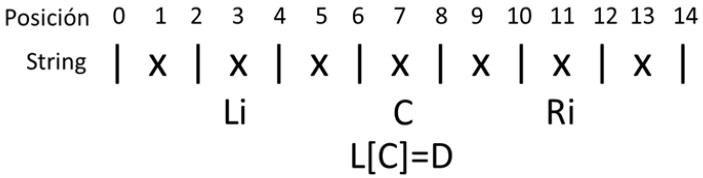
- Este algoritmo, propuesto por Glenn K. Manacher en 1975, logra resolver el problema del palíndromo más largo de una cadena S, en una forma muy eficiente, con una complejidad O(n), proporcional a la longitud n de la cadena.
- Al igual que el algoritmo a fuerza bruta, éste se basa en buscar un palíndromo centrado en i, con algunas diferencias:
  - En el algoritmo a 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).

- El algoritmo inicial 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 una al inicio y otro al final.
- El carácter que se agrega puede ser cualquiera que no pertenezca al alfabeto de la cadena analizada.
- Por ejemplo,
  - Si tenemos la 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.
  - Si tenemos la 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.

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.

Posición	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
String	|	c	|	a	|	c	|	b	|	c	|	a	|	c	|
L	0	1	0	3	0	1	0	7	0	1	0	3	0	1	0

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.



Notación:

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

Manacher encontró que sólo había 4 casos posibles y al implementarlos se reduce la complejidad. Es importante recordar que, dado que la longitud de la cadena original es n, la último posición de la cadena aumentada es $2n$.

1. Si $L[L_i] < (C + L[C]) - R_i$, entonces $L[R_i] = L[L_i]$.
2. Si $L[L_i] = (C + L[C]) - R_i$ y $(C + L[C]) = 2n$, entonces $L[R_i] = L[L_i]$.
3. Si $L[L_i] = (C + L[C]) - R_i$ y $(C + L[C]) < 2n$, entonces en este caso, no conocemos exactamente el valor de que debemos colocar en $L[R_i]$. Sin embargo, sabemos que ese valor debe ser al menos $L[L_i]$ y expandimos el palíndromo carácter por carácter.
4. Si $L[L_i] > (C + L[C]) - R_i$, entonces en este caso tampoco conocemos exactamente el valor de que debemos colocar en $L[R_i]$. Sin embargo, sabemos que ese valor debe ser al menos $L[L_i]$ y expandimos el palíndromo carácter por carácter.

Procedure 2 MANACHER

Input: S : String

$Q \leftarrow EXPAND(S)$
$P \leftarrow Array(Q.length)$
$center \leftarrow 0, right \leftarrow 0$
for $i \leftarrow 2$ **to** $Q.length$ **do**
  $iMirror \leftarrow center - (i - center)$
  if $right > i$ **then**
    $P[i] \leftarrow min(right - i, P[iMirror])$
  end if
  while $Q[i + 1 + P[i]] = Q[i - 1 - P[i]]$ **do**
    $P[i] \leftarrow P[i] + 1$
  end while
  if $i + P[i] > right$ **then**
    $center \leftarrow i$
    $right \leftarrow i + P[i]$
  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)
```

Introducción

- Los algoritmos de hashing son muy útiles en muchas áreas de las ciencias computacionales, sobre todo porque, usando una buena función de hash, es posible reducir la búsqueda a un tiempo casi constante $O(1)$.
- Por ejemplo, dadas las cadenas S_1, de longitud n_1, y S_2, de longitud n_2, indicar si son iguales o no. Si realizamos la comparación a fuerza bruta, el algoritmo resultante tiene un orden $O(min(n_1, n_2))$. Sin embargo, usando una buena función de hash podríamos logar un orden $O(1)$.

- La idea principal de una función hash para cadenas es convertir una cadena en un número entero. Para esto, la función debe cumplir la condición de que, si dos cadenas son iguales, les debe asignar el mismo número entero. Cuando la función le asigna el mismo número a dos cadenas diferentes se dicen que hubo una **colisión**. La función ideal sería aquella que a cada cadena le asignara un entero diferente.
- En realidad, es muy complicado hacer una función con tal condición, pero es suficiente con generar una función donde la probabilidad de asignarle el mismo entero a dos cadenas diferentes sea muy baja, y ese tipo de funciones sí se han podido generar.
- Un ejemplo es la llamada **Polynomial Rolling Hash Function**.

Polynomial Rolling Hash Function

Sea S, una cadena de longitud n, a la que se le quiere aplicar la función hash, y sean $S[0]$, $S[1]$, ..., $S[n-1]$ las representaciones en enteros de cada uno de los caracteres que lo forman, la función Polynomial Rolling Hash Function para la cadena S se define como:

$$\begin{aligned} prhs(S) &= (S[0] * p^0 + S[1] * p^1 + ... + S[n-1] * p^{n-1}) \mod m \\ &= (\sum_{i=0}^{n-1} S[i] * p^i) \mod n \end{aligned}$$

Donde, p y m son número enteros positivos que se deben seleccionar. Una forma común es que ambos sean números enteros primos.

- En el caso de P, se recomienda que sea un número primo cercano al número de caracteres en el alfabeto. Por ejemplo, para los caracteres en español, que son 27, el número primo más cercano es el 31, por lo que $p = 31$ es una buena opción. Si se incluyen también las mayúsculas, el conjunto sube a 54 y entonces $p = 53$ es una buena opción.
- En lo que respecta a m, debe ser un número muy grande, debido a que la probabilidad de que haya colisión es, aproximadamente, 1/m. Una buena selección ha sido un número primo muy grande, por ejemplo, $m = 10^9 + 9$, lo que nos da una probabilidad de colisión de alrededor de 10^{-9}.

- Si bien, una probabilidad de colisión de 10^{-9} es muy baja, si se hace una sola comparación. En algunos problemas es necesario comparar una cadena con un conjunto de otras cadenas. Si ese conjunto es de 10^6 elementos, la probabilidad de que suceda una colisión aumenta a 10^{-3}, y si se comparan las 10^6 cadenas entre si, la probabilidad se acerca peligrosamente a 1.
- Una forma muy común y simple de obtener mejores probabilidades de colisión, incluso con muchas comparaciones, es hacer dos funciones hash, con diferentes p y m. Se aplican las dos funciones hash y sólo se declaran iguales las cadenas si el resultado de ambas funciones son iguales. En este caso, si m para la segunda función es también cercana a 10^9, aplicar las dos funciones es equivalente a tener una m aproximadamente igual a 10^{18}.

```
long long prhf(const std::string &s) {
  int p = 31;
  int m = 1e9 + 9;
  long long result = 0;
  long long power = 1;

  for (int i = 0; i < s.size(); i++) {
        result = (result + ((s[i] - 'a' + 1) * power)) % m
        power = (power * p) % m;
  }
  return result;
}
```

Arreglos de Sufijos (Suffix Array)

El Arreglo de sufijos (Suffix Array) es una estructura de datos que fue propuesta por Udi Manber y Gene Myers en 1990. Es muy utilizada en varias aplicaciones relacionadas con el análisis de cadenas. Se define de la siguiente forma:

- Dada una cadena S, de longitud n, su arreglo de sufijos es un arreglo de enteros que contiene las posiciones que tienen los $n+1$ sufijos en la cadena $T = S\$$, ordenados lexicográficamente, considerando \$ como el primer carácter del alfabeto.

Procedure 3 SUFFIX_ARRAY

1. FORMAR LA CADENA DE DE TRABAJO $T \leftarrow S\$$
2. FORMAR TODOS LOS SUFIJOS DE T
3. ORDENAR LEXICOGRÁFICAMENTE LOS SUFIJOS ENCONTRADOS CONSIDERANDO A $ COMO PRIMER SÍMBOLO DEL ALFABETO
4. FORMAR EL ARREGLO A CON LAS POSICIONES DE INICIO EN LA CADENAS T DE CADA UNO DE LOS SUFIJOS

return A

¿Qué es una subsecuencia?

Considera una cadena A = [a, b, c, d]. Una subsecuencia(también llamada subcadena) se obtiene eliminando 0 o más símbolos (no necesariamente consecutivos) de A.
Por ejemplo:

- Los ejemplos [a, b, c, d] , [a], [b], [c], [d], [a, d], [a, c], [b, d] son subsecuencias de A.
- Pero [d, b], [d, a], [d, a, c] no son subsecuencias de A.

Subsecuencia creciente más larga

Dado un arreglo de n símbolos comparables, dar la subsecuencia creciente más larga. Por ejemplo, A = [-7, 10, 9, 2, 3, 8, 8, 1], la subsecuencia creciente más larga es [-7, 2, 3, 8].

Procedure 4 LIS

Input: $A : Array$

```
Aux : Array
Aux[1] ← 1
for i ← 2 to n do
    count ← 0
    for j ← 1 to (i − 1) do
        if A[j] < A[i] then
            count ← MAX(count, Aux[j])
        end if
    end for
    Aux[i] ← count + 1
end for
```

```
count ← 0
for i ← 1 to n do
    count ← MAX(count, Aux[i])
end for
return count
```

Testing the CATCHER [1]

Un contratista militar del Departamento de Defensa acaba de completar una serie de pruebas preliminares para un nuevo misil defensivo llamado FREEDOM, que es capaz de interceptar múltiples misiles ofensivos entrantes. Se supone que el FREEDOM es un notable misil defensivo. Puede moverse hacia adelante, lateralmente y hacia abajo a velocidades muy rápidas, y puede interceptar un misil ofensivo sin sufrir daños. Pero tiene un defecto importante. Aunque puede dispararse para alcanzar cualquier elevación inicial, no tiene poder para moverse más alto que el último misil que ha interceptado.
Las pruebas que completó el contratista fueron simulaciones por computadora de campo de batalla y condiciones de ataque hostil. Como solo eran preliminares, las simulaciones solo probaron la capacidad de movimiento vertical del FREEDOM. En cada simulación, el FREEDOM fue disparado contra una secuencia de misiles ofensivos que llegaban a intervalos de tiempo fijos. La única información disponible para el FREEDOM para cada misil entrante era su altura en el punto donde podía ser interceptada y donde aparecía en la secuencia de misiles. Cada misil entrante para una ejecución de prueba se representa en la secuencia solo una vez.

[1] https://onlinejudge.org/external/2/231.pdf

El resultado de cada prueba se informa como la secuencia de misiles entrantes y el número total de esos misiles que son interceptados por el FREEDOM en esa prueba.
La Oficina de Contabilidad General quiere asegurarse de que los resultados de la prueba de simulación presentados por el contratista militar sean alcanzables, dadas las limitaciones del FREEDOM. Escribir una solución que tome datos de entrada que representen el patrón de misiles entrantes para una prueba y genere la cantidad máxima de misiles que el FREEDOM puede interceptar para esa prueba. Para cualquier misil entrante en la prueba, el FREEDOM puede interceptarlo si y solo si cumple una de estas dos condiciones:
1. El misil entrante es el primer misil que se intercepta en esta prueba.
-o-
2. El misil fue disparado después del último misil que fue interceptado y no es más alto que el último misil que fue interceptado.

Entrada

Los datos de entrada para cualquier prueba consisten en una secuencia de uno o más enteros no negativos, todos los cuales son menores o iguales a 32,767, que representan las alturas de los misiles entrantes (el patrón de prueba). El último número en cada secuencia es -1, lo que significa el final de los datos para esa prueba.

Salida

La salida consta del número máximo de misiles entrantes que el FREEDOM posiblemente podría interceptar para la prueba. Nota: El número de misiles para cualquier prueba dada no está limitado. Si la solución se basa en un algoritmo ineficiente, es posible que no se ejecute en el tiempo asignado.

Ejemplo de entrada

389
207
155
300
299
170
158
65
-1

Ejemplo de salida

6

Subsecuencia común más larga

Dado dos cadenas, A y B, determinar la subsecuencia común más larga de ambas cadenas. Por ejemplo, si A = [c, d, c, c, f, g, e] y B = [e, c, c, e, g, f, e], ¿cuál la subsecuencia común más larga?

Ahora veamos como se plantea el problema: considere dos secuencias A = [a_1, a_2, ..., a_m] y B = [b_1, b_2, ..., b_n] . Para resolver el problema nos fijaremos en los últimos dos símbolos: a_m y b_n . Como podemos ver hay dos posibilidades:

- Caso 1: $a_m = b_n$. En este caso, la subsecuencia común más larga debe contener a_m . Simplemente basta encontrar la subsecuencia común más larga de [a_1, a_2, ..., a_{m-1}] y [b_1, b_2, ..., b_{n-1}].
- Caso 2: $a_m \neq b_n$. En este caso, puede hacerse corresponder [a_1, a_2, ..., a_m] con [b_1, b_2, ..., b_{n-1}] y también [a_1, a_2, ..., a_{m-1}] con [b_1, b_2, ..., b_n], y nos quedamos con el mayor de los dos resultados.

Procedure 5 LCS

Input: $A, B : Array$

$M : Matrix[0..A.length][0..B.length]$
$m \leftarrow A.length$
$n \leftarrow B.length$
$INIT(M, 0)$
for $i \leftarrow 1$ to m **do**
  for $j \leftarrow 1$ to n **do**
    if $A[i] = B[j]$ **then**
      $M[i][j] \leftarrow 1 + M[i-1][j-1]$
    else
      $M[i][j] \leftarrow MAX(M[i-1][j], M[i][j-1])$
    end if
  end for
end for
return $M[m][n]$

History Grading [2]

Muchos problemas en Ciencias de la Computación implican maximizar alguna medida de acuerdo con restricciones. Considere un examen de historia en el que se pide a los estudiantes que pongan varios eventos históricos en orden cronológico. Los estudiantes que ordenen todos los eventos correctamente recibirán crédito completo, pero ¿cómo se debe otorgar crédito parcial a los estudiantes que clasifiquen incorrectamente uno o más de los eventos históricos? Algunas posibilidades de crédito parcial incluyen:
1. 1 punto por cada evento cuyo rango coincida con su rango correcto.
2. 1 punto por cada evento en la secuencia más larga (no necesariamente contigua) de eventos que estén en el orden correcto entre sí.
Por ejemplo, si cuatro eventos están ordenados correctamente 1 2 3 4, entonces el orden 1 3 2 4 recibiría una puntuación de 2 usando el primer método (los eventos 1 y 4 están clasificados correctamente) y una puntuación de 3 usando el segundo método (evento las secuencias 1 2 4 y 1 3 4 están ambas en el orden correcto entre sí). En este problema, se le pide que escriba un programa para calificar tales preguntas utilizando el segundo método.

[2] https://onlinejudge.org/external/1/111.pdf

Dado el orden cronológico correcto de n eventos 1, 2, ..., n como c_1, c_2,. . . c_n donde $1 \geq c_i \geq$ n denota la clasificación del evento i en el orden cronológico correcto y una secuencia de respuestas de los estudiantes r_1, r_2, ..., r_n donde $1 \geq$ ri $\geq$ n denota el rango cronológico dado por el estudiante al evento i; determinar la longitud de la secuencia más larga (no necesariamente contigua) de eventos en las respuestas de los estudiantes que están en el orden cronológico correcto entre sí.

Entrada

El archivo de entrada contiene uno o más casos de prueba, cada uno de ellos como se describe a continuación.

La primera línea de la entrada constará de un entero n que indica el número de eventos con 2 $\geq$ n $\geq$ 20. La segunda línea contendrá n números enteros, indicando el orden cronológico correcto de n eventos. Las líneas restantes constarán cada una de n números enteros y cada línea representará el orden cronológico de los n eventos por parte del estudiante. Todas las líneas contendrán n números en el rango [1...n], con cada número apareciendo exactamente una vez por línea, y con cada número separado de otros números en la misma línea por uno o más espacios.

Salida

Para cada caso de prueba, la salida debe seguir la descripción a continuación Para cada clasificación de eventos de los estudiantes, su programa debe imprimir la puntuación de esa clasificación. Debe haber una línea de salida para cada clasificación de estudiantes.

Advertencia: lea atentamente la descripción y considere la diferencia entre .ºrdenarz çlasificar".

Ejemplo de entrada

```
4
4 2 3 1
1 3 2 4
3 2 1 4
2 3 4 1
10
3 1 2 4 9 5 10 6 8 7
1 2 3 4 5 6 7 8 9 10
4 7 2 3 10 6 9 1 5 8
3 1 2 4 9 5 10 6 8 7
2 10 1 3 8 4 9 5 7 6
```

Ejemplo de salida

```
1
2
3
6
5
10
9
```