Sobre Matching en Hileras

Victor David Coto Solano

Diego Quirós Artiñano

Derek Rojas Mendoza

Universidad Nacional de Costa Rica

EIF-203: Estructuras Discretas Grupo 01-10am

Carlos Loria-Saenz

Ciclo I 2022

Índice

Índice de fíguras

Índice de tablas

Índice de algoritmos

Introducción General

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris...

String Matching

¿Qué es String Matching?

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris...

Tipos de String Matching

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris...

Algoritmos

Introducción

algo

Fuerza Bruta (Naive Algorithm)

Introducción al Algoritmo de Fuerza Bruta

Implementación del Algoritmo de Fuerza Bruta

Algoritmo 1 Algoritmo de fuerza bruta

```
1:  procedure FUERZABRUTA((texto, patron))
2:      n ← len(texto)
3:      m ← len(patron)
4:      for i ≤ (n − m) do                ▷ Despues de n − m no puede ser el patron
5:          for j < m do                  ▷ Evalua el patron caracter por caracter
6:              if texto[i + j] ≠ patron[j] then   ▷ Evalua si los caracteres son los mismos
7:                  break
8:              end if
9:          end for
10:         if j = m then                 ▷ Si llega al final entonces existe y devuelve la posición
11:             return i
12:         end if
13:     end for
14:     return -1                         ▷ Si pasa por todo y no encuentra no existe devuelva -1
15: end procedure
```

Análisis del Algoritmo de Fuerza Bruta

Paso 1

Paso 2

Paso 3

Paso 4

Paso 5

Paso 6

Knuth-Morris-Pratt (KMP)

Introducción al Algoritmo de Knuth-Morris-Pratt

El algoritmo realiza la búsqueda usando información basada en fallos previos obtenidos del patrón, esto se hace creando una tabla de valores sobre su propio contenido. Esta tabla se crea para ver donde podría darse la siguiente coincidencia sin buscar más de 1 vez los caracteres del texto o cadena de caracteres donde se realiza la búsqueda.

La tabla de fallos se encarga de evitar que cada carácter del texto sea analizado más de 1 vez, esto lo logra comparando el patrón consigo mismo para ver que partes se repiten. Este método guarda una lista con números le indican al algoritmo cuando debe devolverse desde la posición actual una vez que el patrón no coincida con el texto.

El texto y el patrón van avanzando simultáneamente mientras ambos coincidan, si una vez coinciden del todo pero la letra siguiente sigue cumpliendo con el patrón, entonces el algoritmo mueve el patrón 1 a la derecha, si no coinciden, entonces el patrón se empieza a devolver para intentar hacerlo coincidir con el texto.

Implementación del Algoritmo de Knuth-Morris-Pratt

Algoritmo 2 Algoritmo de Knuth-Morris-Pratt

```
1: procedure KNUTH_MORRIS_PUTH((texto, patron))
2:     n ← len(texto)
3:     m ← len(patron)
4:     resultado = False
5:     listaDeIndices = []
6:     tablaDeFallo = [0] * m ▷ Tabla que se va a usar para devoluciones en los procesamientos
7:     i = 0
8:     j = 0
9:     procedure PROCESAMIENTO((patron, m, tablaDeFallo))
10:        longitud = 0                          ▷ longitud del sufijo del prefijo mas largo
11:        i = 1
12:        while i < m do
13:            if patron[i] == patron[longitud] then
14:                longitud+ = 1
15:                tablaDeFallo[i] = longitud
16:                i+ = 1
17:            else
18:                if longitud! = 0 then
19:                    longitud = tablaDeFallo[longitud − 1]
20:                else
21:                    tablaDeFallo[i] = 0
22:                    i+ = 1
23:                end if
24:            end if
25:        end while
26:    end procedure
27:    procedure BÚSQUEDA(())
28:        i = 0
29:        j = 0
30:        while i < n do
31:            if  thenpatron[j] == texto[i]
32:                i+ = 1
33:                j+ = 1
34:            end if
35:            if j == m then
36:                listaDeIndices+ = [i − j]
37:                resultado = True
38:                j = tablaDeFallo[j − 1]
39:            else if i < n ∧ patron[j]! = text[i] then
40:                if j! = 0 then
41:                    j = tablaDeFallo[j − 1]
42:                else
43:                    i+ = 1
44:                end if
```

```
            end if
        end while
    end procedure
    return listaDeIndices
end procedure
```

Análisis del Algoritmo de Knuth-Morris-Pratt

Paso 1

$n + m, n = len(texto) \land m = len(patron)$

Paso 2

Comparaciones entre patron y texto

Paso 3

$1 + T_{n-1} + 1 + T_{m-1}$

Paso 4

$T_{kmp} = n + m$

Paso 5

$O(n + m)$

Paso 6

Algoritmo de Boyer-Moore

Introducción del Algoritmo de Boyer-Moore

Implementación del Algoritmo de Boyer-Moore

Algoritmo 3 Algoritmo de Boyer_Moore

```
1: procedure BOYER-MOORE((patron, texto))
2:     sizeP = len(P)
3:     sizeT = len(T)
4:     boyerMooreBadChar = [0] * 256          ▷ 256 es el número generalmente aceptado como alfabéto
5:     for 0 ≤ i < sizeP − 1 do
6:         boyerMooreBadChar[ord(patron[i])] = sizeP − i − 1
7:     end for
8:     suff = [0] * sizeP
9:     f = 0
10:    g = sizeP − 1
11:    suff[sizeP − 1] = sizeP
12:    for sizeP − 2 ≥ i > −1 do
13:        if i > g ∧ suff[i + sizeP − 1 − f] < i − g then
14:            suff[i] = suff[i + sizeP − 1 − f]
15:        else
16:            if i < g then
17:                g = i
18:            end if
19:            f = i
20:            while g ≥ 0 ∧ P[g] == P[g + sizeP − 1 − f] do
21:                g− = 1
22:            end while
23:            suff[i] = f − g
24:        end if
25:    end for
26:    boyerMooreGoodSuffix = [sizeP] * sizeP
27:    for 0 ≤ i < sizeP do
28:        if suff[i] == i + 1 then
29:            for 0 ≤ j < sizeP − 1 − i do
30:                if boyerMooreGoodSuffix[j] == sizeP then
31:                    boyerMooreGoodSuffix[j] = sizeP − 1 − i
32:                end if
33:            end for
34:        end if
35:    end for
36:    for 0 ≤ i < sizeP − 1 do
37:        boyerMooreGoodSuffix[sizeP − 1 − suff[i]] = sizeP − 1 − i
38:    end for
39:    i = 0
```

```
40:     j = 0
41:     while j ≤ sizeT − sizeP do
42:         i = sizeP − 1
43:         while i! = −1 ∧ patron[i] == texto[i + j] do
44:             i− = 1
45:         end while
46:         if i < 0 then
47:             print(j)
48:             j+ = boyerMooreGoodSuffix[0]
49:         else
50:             j+ = max(boyerMooreGoodSuffix[i], boyerMooreBadChar[ord(T[i + j])] − sizeP + 1 + i)
51:         end if
52:     end while
53: end procedure
```

Análisis del Algoritmo de Boyer-Moore

Paso 1

Paso 2

Paso 3

Paso 4

Paso 5

Paso 6

Algoritmo de Karp-Rabin (o Rabin-Karp)

Introducción del Algoritmo de Karp-Rabin

Implementación del Algoritmo de Karp-Rabin

Algoritmo 4 Algoritmo de Karp-Rabin

```
1: procedure KARP_RABIN((patron, texto))
2:     n = len(patron)
3:     m = len(texto)
4:     d = 256          ▷ Este es el alfabeto defecto que sale en varios analisis (caracteres alfabeto inglés)
5:     q = 33554393     ▷ Cualquier número primo sirve, pero preferiblemente alto porque los pequeños solo hacen que el algoritmo corra como fuerza bruta porque más hashes concuerdan
6:     h = d^(m-1) mod(q)
7:     ValorHashPatron = 0
8:     ValorHashVentanaTexto = 0
9:     listaIndices = []
10:    for i = 0 <n do
11:        ValorHashPatron = (d * ValorHashPatron + patron[i]) mod(q)     ▷ Esta operación sirve con un abecedario normal como tabla ascii, para usarlo en Python el accesor se tiene que meter como parametro de ord()
12:        ValorHashVentanaTexto = (d * ValorHashVentanaTexto + texto[i]) mod(q)
13:    end for
       j = 0  ▷ definirla afuera para poder usarla dentro del for sin tener que redefinirla cada vez que empiece el for otra vez
14:    for i = 0 ≤ m − n do
15:        if ValorHashPatron == ValorHashVentanaTexto then ▷ Solo hacer fuerza Bruta cuando los valores hash concuerdan
16:            for j = 0 < n do
17:                if patron[j]! = texto[i + j] then     ▷ Si la fuerza bruta se incumple salga
18:                    break
19:                else
20:                    j+ = 1
21:                end if
22:            end for
23:            if j == n then
```

```
            listaIndices+ = [i]                ▷ Al llegar al final de la fuerza bruta regista el indice
        end if
    end if
    if i < m − n then
        ValorHashVentanaTexto = (d∗(ValorHashVentanaTexto−texto[i]∗h)+texto[i+
n])mod(q)
        if ValorHashVentanaTexto < 0 then    ▷ GeeksForGeeks recomienda en caso de
que den hashes negativos
            ValorHashVentanaTexto+ = q
        end if
    end if
  end for
  return listaIndices
end procedure
```

Análisis del Algoritmo de Karp-Rabin

Paso 1

Paso 2

Paso 3

Paso 4

Paso 5

El tiempo de este algoritmo es O(n+m) normalmente, pero en el peor de los casos termina siendo igual que fuerza bruta o O(nm) siendo n y m longitudes de texto y patron

Paso 6

Conclusión

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris...
A Complete Tutorial on Changing the Boot Order in BIOS (2022) *A Complete Tutorial on Changing the Boot Order in BIOS* (2022)

Referencias

A Complete Tutorial on Changing the Boot Order in BIOS. (2022, abril). Descargado de https://www.lifewire.com/change-the-boot-order-in-bios-2624528 ([Online; accessed 24. Apr. 2022])