Taller Arboles Binarios Optimos

Pablo Ariza - Santiago Chaustre

Abril 9 del 2018

1. Problema

En los arboles binarios ordenados los elementos son estrictamente ordenados con una relación de comparación $<$ y si estós son equilibrados los arboles tienden a ser equivalentes con un mismo conjunto de datos haciendo que todas las busquedas tengan una complejidad $O(\log n)$. Los arboles binarios ordenados optimos tienden a dejar los datos más buscados de un conjunto de datos lo más cerca de la raiz, esto para que los tiempos de respuesta sean más rapido en los casos «generales».

2. Formalizacion

2.1. Entradas

Un histograma $h(c) : \mathbb{E} \longrightarrow [0, 1]$ que representa la probabilidad (o frecuencia) de la búsqueda de un código c.

2.2. Salidas

Una secuencia $S = \{(n, t) | (n \in \mathbb{N}) \wedge (t \in \mathbb{R})\}$ donde n es el tamaño de las busquedas realizadas y t el tiempo que se demoro realizando las n busquedas.

3. Estructura óptima del problema

$$e[i, j] = \begin{cases} q_{i-1} & ; \quad j = i - 1 \\ \min\limits_{i \leq r \leq j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & ; \quad i \leq j \end{cases}$$

$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

4. Algoritmo recurrente

```
1: procedure BUILD_OPTBINTREE(P, Q)
2:     return BUILD_OPTBINTREE_REC(P, Q, 1, |P|)
3: end procedure
```

```
1: procedure BUILD_OPTBINTREE_REC(P, Q, i, j)
2:     if j = i − 1 then
3:         return Q[i − 1]
4:     else
5:         w ← DUMMY_WEIGHT(P, Q, i, j)
6:         e ← ∞
7:         for r ← i to j do
8:             v_l ← BUILD_OPTBINTREE_REC(P, Q, i, r − 1)
9:             v_r ← BUILD_OPTBINTREE_REC(P, Q, r + 1, j)
10:            v ← v_l + v_r + w
11:            if v < e then
12:                e ← v
13:            end if
14:        end for
15:        return e
16:    end if
17: end procedure
```

```
1: procedure DUMMY_WEIGHT(P, Q, i, j)
2:     w ← Q[i − 1]
3:     for l ← i to j do
4:         w ← w + P[l] + Q[l]
5:     end for
6:     return w
7: end procedure
```

5. Algoritmo recurrente «memoizado»

```
1: procedure BUILD_OPTBINTREE(P, Q)
2:     let W be a matrix [1..|P|] × [1..|P|]
3:     let M be a matrix [0..|P|] × [0..|P|]
4:     W ← 0
5:     M ← 0
6:     for i ← 1 to |P| do
7:         W[i, i] ← Q[i − 1] + P[i] + Q[i]
8:         M[i, i] ← Q[i − 1]
9:         for j ← i + 1 to |P| do
10:            W[i, j] ← W[i, j − 1] + P[j] + Q[j]
11:            M[i, j] ← ∞
12:        end for
13:    end for
14:    return BUILD_OPTBINTREE_REC(P, Q, 1, |P|, M, W)
15: end procedure
```

```
1: procedure BUILD_OPTBINTREE_REC(P, Q, i, j, M, W)
2:     if M[i, j] = ∞ then
3:         if j = i − 1 then
4:             M[i, j] ← Q[i − 1]
5:         else
6:             for r ← i to j do
7:                 v_l ← BUILD_OPTBINTREE_REC(P, Q, i, r − 1, M, W)
8:                 v_r ← BUILD_OPTBINTREE_REC(P, Q, r + 1, j, M, W)
9:                 v ← v_l + v_r + W[i, j]
10:                if v < M[i, j] then
11:                    M[i, j] ← v
12:                end if
13:            end for
14:        end if
15:    end if
16:    return M[i, j]
17: end procedure
```

La tabla de memoización tiene la siguiente forma:

$M[i,j]$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	$Q[0]$	$M[1,1]$	$M[1,2]$	$M[1,3]$	$M[1,4]$	$M[1,5]$
2	0	$Q[1]$	$M[2,2]$	$M[2,3]$	$M[2,4]$	$M[2,5]$
3	0	0	$Q[2]$	$M[3,3]$	$M[3,4]$	$M[3,5]$
4	0	0	0	$Q[3]$	$M[4,4]$	$M[4,5]$
5	0	0	0	0	$Q[4]$	$M[5,5]$

Y w puede ser pre-calculado:

$W[i,j]$	1	2	3	4	5
1	$Q[0]+P[1]+Q[1]$	$W[1,1]+P[2]+Q[2]$	$W[1,2]+P[3]+Q[3]$	$W[1,3]+P[4]+Q[4]$	$W[1,4]+P[5]+Q[5]$
2	0	$Q[1]+P[2]+Q[2]$	$W[2,2]+P[3]+Q[3]$	$W[2,3]+P[4]+Q[4]$	$W[2,4]+P[5]+Q[5]$
3	0	0	$Q[2]+P[3]+Q[3]$	$W[3,3]+P[4]+Q[4]$	$W[3,4]+P[5]+Q[5]$
4	0	0	0	$Q[3]+P[4]+Q[4]$	$W[4,4]+P[5]+Q[5]$
5	0	0	0	0	$Q[4]+P[5]+Q[5]$

6. Algoritmo «bottom-up» con la solución

```
1:  procedure BUILD_OPTBINTREE(P, Q)
2:      let W be a matrix [1..|P|] × [1..|P|]
3:      let M be a matrix [0..|P|] × [0..|P|]
4:      let R be a matrix [1..|P|] × [1..|P|]
5:      W ← 0
6:      M ← 0
7:      R ← 0
8:      for i ← 1 to |P| do
9:          W[i,i] ← Q[i−1] + P[i] + Q[i]
10:         M[i,i] ← Q[i−1]
11:         for j ← i+1 to |P| do
12:             W[i,j] ← W[i,j−1] + P[j] + Q[j]
13:             M[i,j] ← ∞
14:         end for
15:     end for
16:     for l ← 1 to |P| do
17:         for i ← 1 to |P| − l + 1 do
18:             j ← i + l − 1
19:             for r ← i to j do
20:                 v ← M[i,r−1] + M[r+1,j] + W[i,j]
21:                 if v < M[i,j] then
22:                     M[i,j] ← v
23:                     R[i,j] ← r
24:                 end if
25:             end for
26:         end for
27:     end for
28:     return R
29: end procedure
```

7. Invariante

La invariante del bottom up es recorrer la tabal e ir llenandola con el minimo de la suma de las probabilidades de exito y de desborde.

8. Resultado

9. Manual de compilacion

Leer el readme.md para compilar y ejecutar el codigo.