Programación dinámica: Árboles de búsqueda binaria óptimos

Grupo L2_8

Aythami Torrado Cabrera

Kevin Diaz Marrero

Daniel Fernandez Perez

Índice

- Introducción.
- Descripción del problema.
- Algoritmos (pseudocódigo y análisis).
- Evaluación experimental.
- Conclusiones.
- Bibliografía.

Introducción

Árbol binario: Un árbol binario no es más que una estructura de datos cuyos nodos pueden tener un hijo a la izquierda o a la derecha (no más de 2).

Árbol binario óptimo: Un árbol binario óptimo consiste en un árbol de búsqueda donde el coste medio de buscar un elemento se reduce al mínimo.

Descripción del problema.

Se tiene un conjunto de claves distintas: $w_1 < w_2 < \ldots < w_n$

– Se conoce la probabilidad p_i, $1 \leq i \leq n$,

–Se quiere construir un árbol binario de búsqueda para guardar las claves que minimice el número medio de comparaciones para encontrar una clave o para garantizar que no está.

–La profundidad de la raíz es 0.

–El número medio de comparaciones para encontrar una clave o no hacerlo es :

$$C = \sum_{i=1}^{n} p_i (d_i + 1) + \sum_{i=0}^{n} q_i d_i$$

Algoritmos (pseudocódigo y análisis). Programación dinámica.

```
FUNCTION
OPTIMAL_BINARY_SEARCH_TREE(keys, freq, n)
begin
   for i = 0 to n-1 do
      aux_matrix(i,i) <- freq(i)
   for L = 2 to n do
      for i = 0 to n do
      j <- i + L - 1
      aux_matrix(i,j) <- INFINITY
      for r = i to j do
         obst_freq <- 0
         if r > i then
            obst_freq + aux_matrix(i, r-1)
         elseif r < j then
            obst_freq + aux_matrix(r+1,j)
         else...
         else
            obst_freq + 0

            obst_freq + SUM(freq, i, j)
            if obst_freq < aux_matrix(i,j)
            then
               aux_matrix(i,j) <-
obst_freq
               return aux_matrix(0, n-1)
end
```

Algoritmos (pseudocódigo y análisis). Programación dinámica.

Nuestra función SUM es:

```
FUNCTION SUM(freq, i, j)
begin
   s <- 0
   for k = i to j then
         s + freq(k)
   return s
end
```

Complejidad: El tiempo de complejidad del algoritmo es $O(n^4)$.

Algoritmos (pseudocódigo y análisis). Programación recursiva.

En el caso del algoritmo recursivo. Tenemos que:

```
FUNCTION BOTTOMUP(keys, freq, n)
begin
  for i = 1 to SIZE_KEYS do
    limit <- i + SIZE_KEYS - n;
    for i = 0 to limit do
      j <- i + n - 1
      aux_matrix(i,j) <- INFINITY
      for  r = i to j
        temp < - SUM (i, j)
        if r > i then
          temporal  <- temporal + aux_matrix(i, r-1)
        if r < j
          temporal  <- temporal + aux_matrix(r+1, j)
        if temporal  < aux_matrix(i,j) then
          aux_matrix(i,j) <- temporal
end
```

Evaluación experimental.

Ejecución del programa para los siguientes nodos para la programación Dinámica.

Nodos	Iteraciones	Tiempo (~seg)
2	3	0.11
3	11	0.29
4	26	0.49
7	133	2.30
10	375	22.68
15	1225	139.74
50	42875	37589.90

Ejecución del programa para los siguientes nodos para el algoritmo recursivo.

Nodos	Iteraciones	Tiempo (~seg)
2	6	0.17
3	20	0.33
4	50	0.61
7	336	3.85
10	1210	19.58
15	5440	141.42
50	563550	39434.32

Conclusiones

– Efectividad

– Comparativa con la programación recursiva.

Bibliografía

http://www.geeksforgeeks.org/dynamic-programming-set-24-optimal-binary-search-tree/

https://gist.github.com/mike168m/55fdb7e962fc521e56499b5c0b7dbe95

https://alg12.wikischolars.columbia.edu/file/view/OPTIMALBST.pdf

http://www.eli.sdsu.edu/courses/fall95/cs660/notes/OBST/OBST.html

http://software.ucv.ro/~cmihaescu/ro/laboratoare/SDA/docs/arboriOptimali_en.pdf

http://www.radford.edu/~nokie/classes/360/dp-opt-bst.html