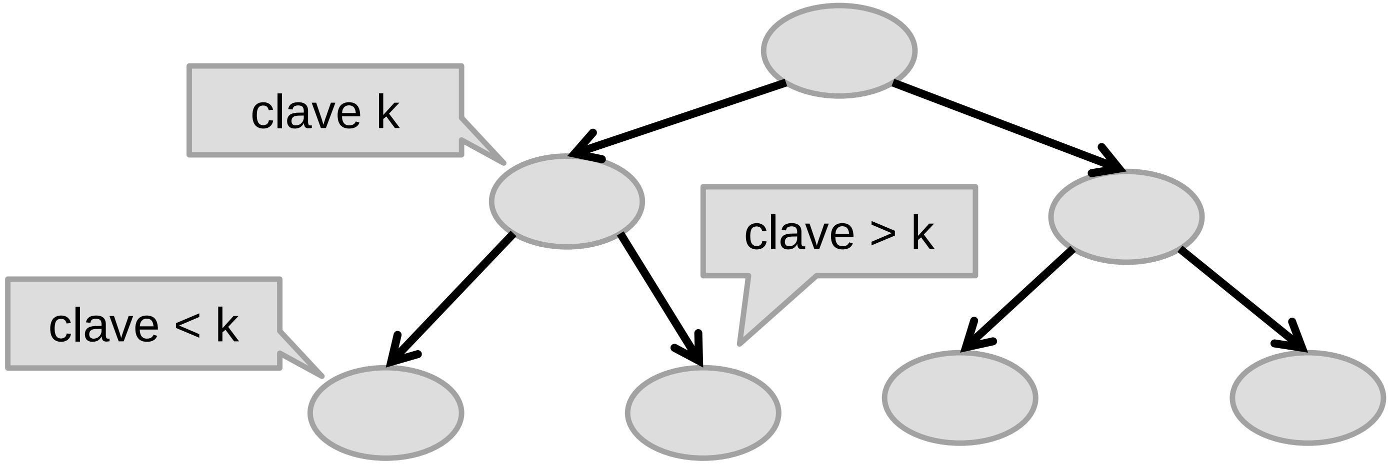
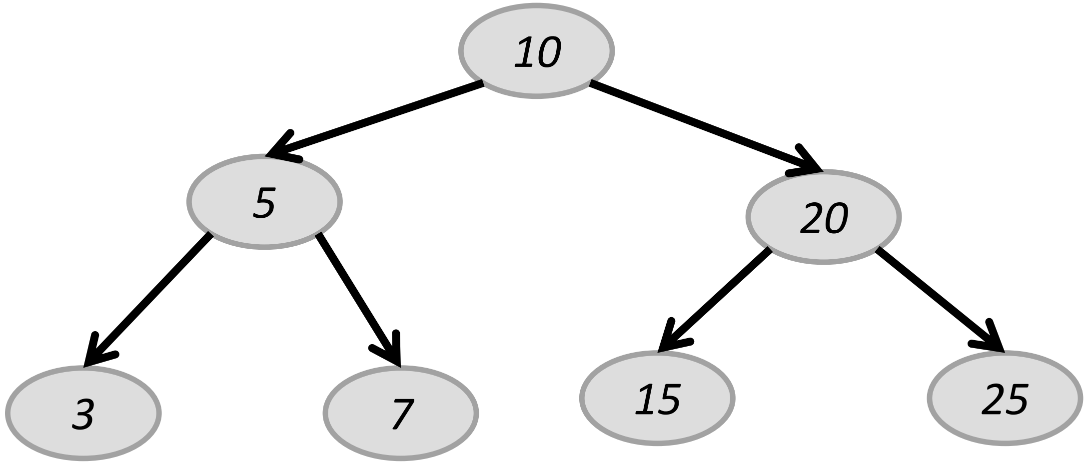
Árboles binarios de búsqueda óptimos, parte 1

Árboles binarios de búsqueda óptimos

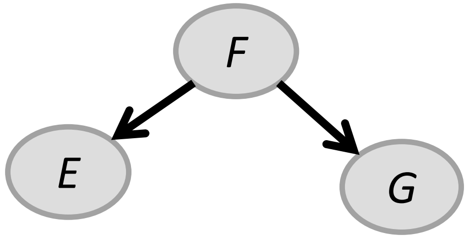
Hagamos memoria: ¿qué es un árbol binario de búsqueda?





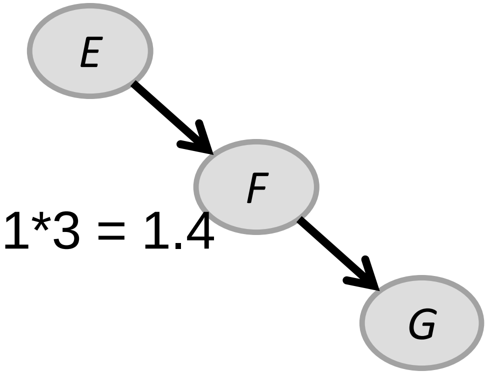
Árboles binarios de búsqueda óptimos

Supongamos que queremos tener un árbol binario de búsqueda con tres elementos E, F y G, y que la relación entre las claves correspondientes es $C_E < C_F < C_G$. Si adicionalmente sabemos que las probabilidades de búsqueda de cada elemento son $C_E = 0.7$, $C_F = 0.2$ y $C_G = 0.1$, ¿cuál es el tiempo de búsqueda promedio del siguiente ABB?



0.7*2 + 0.2*1 + 0.1*2 = 1.8

¿Cuál sería el ABB que minimiza ese tiempo?



0.7*1 + 0.2*2 + 0.1*3 = 1.4

Conclusión: un ABB balanceado no necesariamente es óptimo cuando existen probabilidades de búsqueda No uniformes

Árboles binarios de búsqueda óptimos

Entrada: Probabilidades de búsqueda $P = \{p_1, p_2, \ldots, p_N\}$ para N elementos. Por simplicidad vamos a suponer que dichas probabilidades aparecen ordenadas según las claves de los elementos ($c_1 < c_2 < \ldots < c_N$)

En teoría $\sum_{i=1}^{N} p_i = 1$, aunque en general se pueden considerar como valores reales No negativos y no nulos

Salida: ABB que minimice el tiempo promedio de búsqueda

En otras palabras, encontrar T que es el ABB que minimiza:

$$C(T) = \sum_{i=1N} p_i * (tiempo\ de\ búsqueda\ de\ i\ en\ T) \quad = \sum_{i=1}^{N} p_i * (nivel\ de\ i\ en\ T + 1)$$

Árboles binarios de búsqueda óptimos

Una solución por búsqueda exhaustiva sería ingresar los nodos en las N! formas posibles a un ABB y por cada uno calcular el tiempo promedio de búsqueda. En este caso la eficiencia sería O$((N+1)!)$

Antes de pensar en una solución mediante programación dinámica, primero pensemos: si ya supiéramos que en la solución óptima r ($1 \leq r \leq N$) debe ser la raíz, ¿qué podríamos decir de los subárboles izquierdo TL y derecho TR?

Respuesta: TL debe ser óptimo para los elementos 1,2,…,r-1 y TR debe ser óptimo para los elementos r+1,r+2,…,N

Árboles binarios de búsqueda óptimos

Como vimos antes: $C(T) = \sum_{i=1}^{N} p_i * (tiempo\ de\ búsqueda\ de\ i\ en\ T)$

Pero si ya conocemos la raíz *r* esta expresión se transforma en:

$$C(T) = p_r * 1 + \sum_{i=1}^{r-1} p_i * (t\ búsqueda\ de\ i\ en\ T) + \sum_{i=r+1}^{N} p_i * (t\ de\ búsqueda\ de\ i\ en\ T)$$

$$C(T) = p_r * 1 + \sum_{i=1}^{r-1} p_i * (1 + t\ búsqueda\ de\ i\ en\ T_L) + \sum_{i=r+1}^{N} p_i * (1 + t\ búsqueda\ de\ i\ en\ T_R)$$

$$C(T) = p_r * 1 + \sum_{i=1}^{r-1} p_i + \sum_{i=1}^{r-1} p_i * (t\ búsqueda\ de\ i\ en\ T_L) + \sum_{i=r+1}^{N} p_i + \sum_{i=r+1}^{N} p_i * (t\ búsqueda\ de\ i\ en\ T_R)$$

$$C(T) = \sum_{i=1}^{N} p_i + \sum_{i=1}^{r-1} p_i * (t\ búsqueda\ de\ i\ en\ T_L) + \sum_{i=r+1}^{N} p_i * (t\ búsqueda\ de\ i\ en\ T_R)$$

Finalmente, obtenemos que: $C(T) = \sum_{i=1}^{N} p_i + C(T_L) + C(T_R)$ [Ecuación 1]

Solución mediante programación dinámica

¿Cómo definimos que elementos constituyen un subproblema y cuál sería la forma de su solución óptima?

Notación: sea $C_{i,j}$ con $1 \leq i \leq j \leq N$ el tiempo promedio de búsqueda del ABB óptimo para los elementos *i* a *j*

Considerando esta notación junto con la Ecuación 1, podemos definir la siguiente relación de recurrencia:

$$C_{i,j} = \left\{\begin{matrix} p_i & si\ i = j \\ MIN_{i \leq r \leq j}\left(\sum_{h=i}^{j} p_h + C_{i,r-1} + C_{r+1,j}\right) & si\ i \neq j \end{matrix}\right\}$$

En otras palabras, dados los elementos *i* a *j*, se evalúan las *j-i+1* posibilidades para la raíz y se escoge la mejor alternativa.