Apunte: Función de costo T(n) y árbol de recursión

[Version: 6 de octubre de 2023]

Función de costo $\mathbf{T(n)}$

Dado un algoritmo podemos derivar la función de costo temporal $T(n)$ para el peor caso.

Composición secuencial. El costo temporal del programa `S1;S2` es la suma del costo temporal de `S1` y el costo temporal de `S2`:

$$T_{\texttt{S1;S2}}(n) = T_{\texttt{S1}}(n) + T_{\texttt{S1}}(n)$$

Asignación. El costo temporal del programa `x = Expr` es la suma de los costos de

(1) acceder a `x`,
(si `x` es una variable, el costo de acceder a `x` es constante)

(2) calcular `Expr`, y

(3) copiar el valor de `Expr`.
(si `x` es una referencia entonces no se contabiliza el costo de copiado).

Si `x` es una expresión más compleja como `v[i]` el costo es el costo de calcular esa expresión. En particular, el caso de `v[i]` de acceso a una posición del vector también es constante como indica su documentación.

Condicional. El costo temporal de `if (cond) S1 else S2` es la suma de los costos de

(1) evaluar `cond`, y

(2) el peor caso entre ejecutar `S1` y ejecutar `S2`.

Si `cond` siempre es `false` no se contabiliza `S1`. Si `cond` siempre es `true` no se contabiliza `S2`.

For. El costo temporal del programa `for(int i = k; i <= Expr; i++) {Sc}` es la sumatoria de los costos de cada iteración del ciclo. Se computa el costo de evaluar la condición una vez más que la cantidad de iteraciones, esto es para contabilizar la última evaluación de la condición que da falso.

$$T(n) = \text{costo de asignar “}\texttt{i = k}\text{”} + \text{costo de evaluar “}\texttt{i <= Expr}\text{”} + \sum_{i=k}^{Expr} (\text{costo de la iteración } \texttt{i})$$

El costo de cada iteración del ciclo es la suma de los costos de evaluar `i <= Expr`, ejecutar el cuerpo `Sc` en esa iteración y ejecutar `i++`. El costo de ejecutar `Sc` podría depender de `i`.

While. El costo temporal del programa `while(B) Sc` es la sumatoria de los costos de cada iteración del ciclo. El costo de cada iteración del ciclo es la suma del costo de evaluar `B` y el costo de ejecutar el cuerpo `S` para esa iteración. Al igual que en el caso del `for`, se computa un costo adicional de evaluar `B`.

$$T_{\texttt{while(B)Sc}}(n) = \text{costo de evaluar } \texttt{B} + \sum_{\text{por cada iter.}} (T_{\texttt{Sc}}(n) + \text{costo de evaluar } \texttt{B})$$

En este caso, para especificar los límites de la sumatoria se debe identificar cuál es la variable que se está utilizando para iterar. El costo `Sc` podría depender del número de iteración y esto debe estar reflejado en la sumatoria al escribir el costo $T_{\texttt{Sc}}(n)$.

Llamado a función.

- El costo temporal de llamar a una función `mi_func` es el costo temporal $T(m)$ del algoritmo que implementa `mi_func`, donde m es la medida de tamaño de entrada adecuada para los valores de los parámetros con los que se llamó a `mi_func`.
- Si no conocemos $T(m)$ pero sabemos que $T(m) \in \Theta(f(m))$, podemos hacer el abuso de notación de tomar el costo como $c \cdot f(m)$, donde c es una constante.
- Si, en cambio, sólo sabemos que $T(m) \in O(f(m))$, podemos también utilizar $c \cdot f(m)$ pero esto sólo será una cota superior del costo y servirá únicamente para calcular el órden O del algoritmo que estemos analizando.

Algoritmos recursivos. El costo temporal $T(n)$ de un algoritmo recursivo es la función partida:

$$T(n) = \begin{cases} \text{costo de ejecutar caso base} & \text{si n es caso base} \\ \text{costo de ejecutar caso recursivo} & \text{si n no es caso base} \end{cases}$$

En general, los algoritmos recursivos que analizaremos en la materia siguen el patrón Divide & Conquer, cuyo costo temporal tiene la siguiente forma:

$$T(n) = \begin{cases} c & \text{si } n = 0, n = 1, \\ aT(n/b) + f(n) & \text{si } n > 1 \end{cases}$$

(a subproblemas de tamaño n/b, $f(n)$: costo de conquer)

Esta es una ecuación de recurrencia, una forma de expresar $T(n)$ de manera recursiva. En general, para analizar el órden de complejidad de $T(n)$ queremos poder expresarla como una formula cerrada sin recursión. Para eso haremos uso del método del árbol de recursión.

Árbol de recursión de $\mathbf{T(n)}$

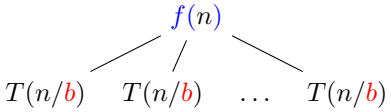
- En general, el árbol de recursión permite dar una fórmula cerrada no recursiva para $T(n)$, la cual podemos analizar más fácilmente para proponer otra función $g(n)$ candidata a cumplir a cumplir $T(n) \in O(g(n))$.
- El árbol de recursión es el desarrollo expandido de la recurrencia $T(n)$ escrito en forma de arbol, con la raíz arriba de todo y las hojas abajo de todo.
- Cada nodo del árbol indica un término que esta sumando una parte del costo al costo total. Es decir, nuestro objetivo final es calcular la suma de lo que indican todos los nodos del árbol.

Dibujar árbol de recursión. Para dibujar el árbol de una ecuación de recurrencia $T(n)$ que sigue el patrón Divide & Conquer (visto en el apartado anterior) se deben realizar los siguientes pasos:

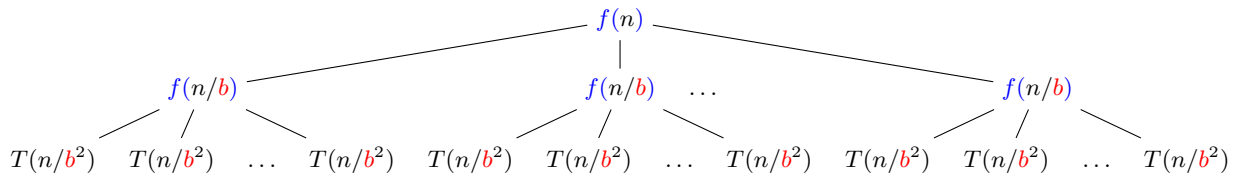
1) escribir $T(n)$ arriba de todo donde irá la raíz:

$$T(n)$$

2) reemplazar $T(n)$ por un arbol cuya raíz es $f(n)$, de la que se desprenden a nodos hijos cuyo valor es $T(n/b)$:



3) realizar el paso anterior para cada uno de los $T(n/b)$ surgidos en el paso anterior,



4) escribir puntos suspensivos verticales indicando que omitimos la escritura de los niveles intermedios,

5) escribir el nivel más bajo del árbol: una serie de nodos cuyo valor es la constante c, la cantidad de estos nodos aún no la conocemos

$$\vdots$$

$$c \qquad c \qquad c \qquad c \qquad c \qquad c \qquad \dots \qquad c$$

Deducir formula cerrada para $T(n)$. Queremos una formula cerrada no recursiva para $T(n)$ que podamos analizar más facilmente y nos permita proponer una función $g(n)$ candidata a cumplir $T(n) \in O(g(n))$. En general, la formula cerrada seguirá la siguiente forma:

$$T(n) = \sum_{i=0}^{\text{altura árbol}} \text{costo total del nivel } i \text{ del árbol}$$

donde el nivel $i = 0$ es la raíz, el nivel $i = 1$ es el que desprende directamente de la raíz y el nivel $i =$ 'altura árbol' es el nivel más bajo que contiene todas las "hojas" (nodos que no tienen "hijos").

- **La altura del árbol es** $\log_b n$. Se llega al último nivel cuando los nodos toman el valor $T(1)$, es decir cuando $n/b^i = 1$, es decir cuando $b^i = n$, donde i es el número de nivel. Esto se cumple cuando $i = \log_b n$.
- **Costo total del nivel i del árbol**. Para todos los niveles excepto el último (i.e. para $i \neq$ 'altura arbol') sabemos que:

  $$\text{Costo del nivel } i = f(n/b^i) \cdot (\text{cantidad de nodos del nivel } i)$$

  Además, como cada nivel multiplica por a la cantidad de nodos del nivel anterior, tenemos que (cantidad de nodos del nivel i) $= a^i$.
- **Costo total del último nivel del árbol**. En general, necesitamos tratar aparte el costo del último nivel, que se da cuando $i =$ 'altura arbol', es decir $i = \log_b n$. Para ese caso, como cada nodo del último nivel tiene costo constante c tenemos que:

  $$\text{Costo del nivel } \log_b n = c \cdot (\text{cantidad de nodos del nivel } i) = c \cdot a^{\log_b n} = c \cdot n^{\log_b a}$$

Teniendo esto en cuenta podemos reescribir el cálculo de $T(n)$:

$$T(n) = \left(\sum_{i=0}^{\log_b n - 1} \text{Costo del nivel } i \right) + \text{Costo del nivel } \log_b n$$

Para los propósitos de este análisis se puede suponer que n es potencia de b. Es decir que existe un entero k tal que $b^k = n$.