Pauta

Análisis Amortizado

Ejercicio 1

Observemos que la cantidad de cuadrados perfectos desde 1 hasta n es $\lfloor\sqrt{n}\rfloor$, a lo que llamaremos m.

El objetivo es calcular el costo amortizado por operación, que está dado por $C(n) = \frac{T(n)}{n}$, donde $T(n)$ es el tiempo total tras realizar n operaciones y n es el número total de operaciones.

El tiempo total $T(n)$ está dado por:

$$T(n) = n - m + \sum_{i=1}^{m} i^2$$

Desarrollando esta expresión:

$$\begin{aligned} T(n) &= n - m + \frac{m(m+1)(2m+1)}{6} \\ &= n - m + \frac{2m^3 + 3m^2 + m}{6} \\ &= \frac{6n - 6m + 2m^3 + 3m^2 + m}{6} \\ &= \frac{6n + 2m^3 + 3m^2 - 5m}{6} \end{aligned}$$

Ahora calculamos $C(n)$:

$$\begin{aligned} C(n) &= \frac{T(n)}{n} \\ &= \frac{6n + 2m^3 + 3m^2 - 5m}{6n} \\ &= 1 + \frac{m^3}{3n} + \frac{m^2}{2n} - \frac{5m}{6n} \end{aligned}$$

Reemplazando m por $\lfloor\sqrt{n}\rfloor$:

$$C(n) = 1 + \frac{\lfloor\sqrt{n}\rfloor^3}{3n} + \frac{\lfloor\sqrt{n}\rfloor^2}{2n} - \frac{5\lfloor\sqrt{n}\rfloor}{6n}$$

Dado que es complicado trabajar con $\lfloor\sqrt{n}\rfloor$, aproximamos $\lfloor\sqrt{n}\rfloor \approx \sqrt{n}$:

$$\begin{aligned} C(n) &= 1 + \frac{n^{3/2}}{3n} + \frac{n^{2/2}}{2n} - \frac{5n^{1/2}}{6n} \\ &= 1 + \frac{n^{1/2}}{3} + \frac{1}{2} - \frac{5}{6n^{1/2}} \\ &= 1 + \frac{1}{2} + n^{1/2} - \frac{5}{6n^{1/2}} \in \Theta(\sqrt{n}) \end{aligned}$$

Por lo tanto, el costo amortizado por operación $C(n)$ es del orden $\Theta(\sqrt{n})$.

Subarreglo con Suma Máxima

```
int maxSubArray(int A[], int izq, int der) {
    if (der - izq + 1 == 1)
        return A[izq];
    else {
        // Dividir
        int m = (izq + der) / 2;

        // Conquistar
        int s1 = maxSubArray(A, izq, m);
        int s2 = maxSubArray(A, m + 1, der);

        // Combinar
        int s = A[m];
        int M1 = A[m];
        for (int j = m - 1; j >= izq; j--) {
            s += A[j];
            if (s > M1) M1 = s;
        }

        s = A[m + 1];
        int M2 = A[m + 1];
        for (int j = m + 2; j <= der; j++) {
            s += A[j];
            if (s > M2) M2 = s;
        }

        return max({s1, s2, M1, M2, M1 + M2});
    }
}
```

El tiempo de ejecución es $O(n \log n)$.

Fibonacci

Asumiremos que la multiplicación de dos números tiene un tiempo constante. Aunque esto no siempre es cierto en todas las circunstancias, para este problema consideramos que la operación de multiplicar dos números es $O(1)$.

Siendo f_n el n-esimo numero de Fibonacci, tenemos:

$$\begin{pmatrix} f_{n-1} & f_n \\ f_n & f_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n$$

Como las multiplicacioes de matrices so siempre entre matrices 2×2, podemos considerar el tiempo de multiplicacion como constante. Realizar las n multiplicaciones nos tomaria tiempo $O(n)$.

Hay que notar que la multiplicacion de matrices es asociativa.

Para $n = 4$, por ejemplo, podemos considerar multiplicar la mitad de la derecha y luego la de la izquierda, esto reduce el numero de multiplicaciones a 3:

$$\begin{pmatrix}0 & 1\\1 & 1\end{pmatrix}^4 = \begin{pmatrix}0 & 1\\1 & 1\end{pmatrix}\begin{pmatrix}0 & 1\\1 & 1\end{pmatrix}\begin{pmatrix}0 & 1\\1 & 1\end{pmatrix}\begin{pmatrix}0 & 1\\1 & 1\end{pmatrix} = \left(\begin{pmatrix}0 & 1\\1 & 1\end{pmatrix}\begin{pmatrix}0 & 1\\1 & 1\end{pmatrix}\right)\left(\begin{pmatrix}0 & 1\\1 & 1\end{pmatrix}\begin{pmatrix}0 & 1\\1 & 1\end{pmatrix}\right)$$

Notar ahora que ambas mitades son iguales, por lo que solo necesito multiplicar una mitad y multiplicar el resultado por si mismo, terminando con un total de 2 multiplicaciones.

Generalizando, cuando $n = 2^k, k \in \mathbb{N}$ (n es potencia de 2), podemos defiir recursivamete la matriz para n:

$$M(n) = \begin{cases} M, & n = 1 \\ (M(n/2))^2, & n > 1 \end{cases}$$

De esta forma se realizan $log(n)$ multiplicaciones de matrices.

Para el caso general, es decir, cuando a no es necesariamente una potencia de 2, en algun punto puede que $n/2$ sea impar. En este caso se realiza division entera (i.e: funcion piso) y luego se multiplica por el resto:

$$M(n) = \begin{cases} M, & \text{n} = 1 \\ (M(n/2))^2, & \text{n} > 1 \text{ y n par} \\ M \times (M(\lfloor n/2 \rfloor))^2, & \text{n} > 1 \text{ y n impar} \end{cases}$$

Asi, tenemos un algoritmo que calcula el n-esimo numero de Fibonacci en tiempo $O(log(n))$

Desafío ultra difícil

Click aqui