

# Algoritmos y Estructuras de Datos - Pauta Control 2

Profesor: Patricio Poblete

27 de Junio del 2009

## Pregunta 2

(a) La idea de linear probing es resolver las colisiones buscando el casillero libre más cercano hacia la derecha (de manera circular). Asumimos dada la función de hashing  $h$ , el arreglo `tabla` y una variable  $M$  que indica el tamaño de la tabla. También asumimos por simplicidad que siempre hay un casillero libre en la tabla de hash (esto se puede verificar fácilmente).

```
public void insertar(int x)
{
    int i=h(x); //calculamos la funcion de Hash
    while(tabla[i] != 0) //buscamos el primero espacio vacio
        i=(i+1) % M;
    tabla[i]=x;
}
```

(b) Ahora la idea es, en caso de colisión, dejar el elemento entrante en la posición que le asigna la función de hash, y desplazar todo el bloque de colisión una casilla a la derecha (siempre de manera circular). Asumimos lo mismo que en la parte (a).

```
public void insertar(int x)
{
    int i=h(x);
    int aux=x; //recuerda el elemento que sera sobrescrito
    while(tabla[i] != 0) //vamos desplazando los elementos a la derecha
    {
        int aux2=tabla[i];
        tabla[i]=aux;
        aux=aux2;
        i=(i+1) % M;
    }
    tabla[i]=aux;
}
```

(c) El efecto de una inserción sobre el costo total de búsqueda es exactamente el mismo en los dos casos. Si llamamos  $d$  al tamaño del bloque de colisión (si  $d$  es 0 lo interpretamos como que no hay colisión), en el caso (a) el costo total de búsqueda se ve incrementado sólo por el costo asociado al nuevo elemento (ya que el de los demás no varía), el cual será  $d + 1$ . En el caso (b), el costo se ve incrementado debido al nuevo elemento y a los que pertenecen al bloque de colisión. El nuevo elemento aporta al costo una unidad, al igual que cada elemento en el bloque de colisión, por lo tanto el incremento en el costo total de búsqueda también es  $d + 1$ . Lo anterior nos dice que si partimos con la tabla vacía e insertamos  $n$  elementos utilizando en un caso (a) para resolver colisiones y (b) en el otro, tendremos el mismo costo total de búsqueda, lo cual implica que tenemos el mismo costo esperado de búsqueda exitosa (asumiendo distribución uniforme sobre las  $n$  llaves).

Otra forma de ver que el costo esperado de búsqueda exitosa es el mismo, es notar que en los 2 casos formamos un nuevo bloque de colisión de tamaño  $d + 1$ , en la misma posición de la tabla, lo cual nos dice que después de  $n$  inserciones, la forma en que están distribuidas las llaves en la tabla es exactamente la misma en

los 2 casos (bajo renombres de llaves). Como se asume distribución uniforme el costo esperado de búsqueda exitosa debe ser el mismo.

### Pregunta 3

(a) Queremos ver de que orden es la altura del árbol de partición más desbalanceado (que es el peor caso). El árbol más desbalanceado deja a un lado  $\frac{1}{3}$  de los elementos y al otro  $\frac{2}{3}$ . En este último subárbol sucede lo mismo, deja  $\frac{1}{3}$  a un lado y  $\frac{2}{3}$  al otro, así sucesivamente hasta llegar a las hojas. Para ver el orden de la altura en el peor caso podemos escribir la recurrencia

$$h(n) = 1 + h\left(\frac{2}{3}n\right)$$

donde  $h(1) = 0$ . Desenrollando la ecuación tenemos que  $h(n) = k$ , con  $k$  tal que  $\left(\frac{2}{3}\right)^k n = 1 \Leftrightarrow \left(\frac{3}{2}\right)^k = n$ , es decir,  $h(n) = \log_{\frac{3}{2}} n = O(\log n)$ . Otra forma más directa de llegar a que  $h(n)$  es de orden logarítmico es usar el teorema maestro.

Como en escoger el pivote y particionar tenemos costo  $O(n)$ , el costo total en el peor caso es  $O(n \log n)$ .

(b) En QuickSelect, después de escoger el pivote, solo buscamos en el lado que nos conviene. El peor caso, al igual que antes, es cuando el pivote deja en cada lado  $\frac{1}{3}$  y  $\frac{2}{3}$  de los elementos. Como escoger el pivote y particionar es  $O(n)$  podemos escribir la siguiente recurrencia.

$$T(n) = cn + T\left(\frac{2}{3}n\right)$$

desenrollando ( $T(1) = 0$ )

$$\begin{aligned} T(n) &= cn + c\left(\frac{2}{3}\right)n + T\left(\left(\frac{2}{3}\right)^2 n\right) \\ &= cn + c\left(\frac{2}{3}\right)n + c\left(\frac{2}{3}\right)^2 n + T\left(\left(\frac{2}{3}\right)^3 n\right) \\ &= cn\left(1 + \left(\frac{2}{3}\right) + \left(\frac{2}{3}\right)^2 + \dots\right) + T(1) \\ &< cn \sum_{i=0}^{\infty} \left(\frac{2}{3}\right)^i \\ &= 3cn \end{aligned}$$

es decir,  $T(n) = O(n)$ . Otra forma más directa es usar el teorema maestro.