CC4102 - Diseño y Análisis de Algoritmos
Auxiliar 3

Prof. Gonzalo Navarro; Aux. Mauricio Quezada

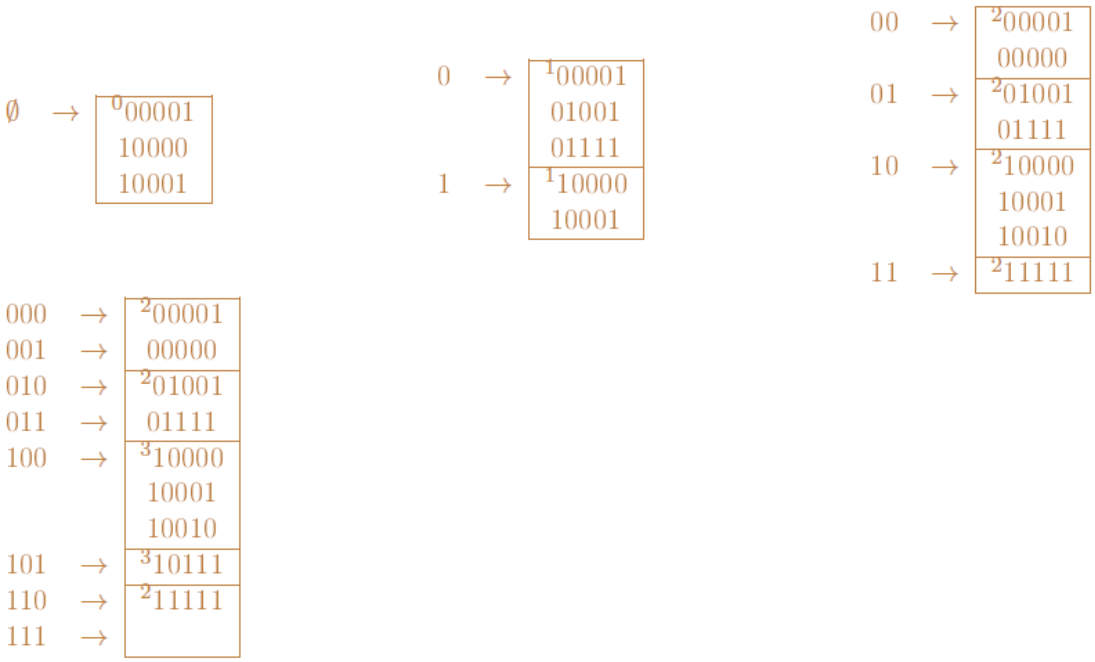
16 de Noviembre, 2011

1 Extendible Hashing

1. Dado el tamaño de página $B = 3$, muestre cómo se construye una tabla de hash según el esquema Extendible Hashing con la secuencia

$$00001, 10000, 10001, 01001, 01111, 00000, 10010, 11111, 10111$$

1.1 Solucion



2 Análisis amortizado

1. Considere la siguiente estructura: inicialmente, se tiene un arreglo de tamaño n_0. Al llenarse completamente e insertar un nuevo elemento, se crea un arreglo del doble de tamaño y se copian todos los elementos anteriores en el nuevo arreglo, de forma de tener espacio para el elemento a insertar. Muestre mediante el método de la función potencial que el costo amortizado por inserción en esta estructura es $\mathcal{O}(1)$.

2. (Propuesto) Haga el mismo análisis utilizando el análisis agregado y la contabilidad de costos.

2.1 Solución

1. Si comenzamos con un arreglo vacío, y agregamos elementos n veces, tomará $O(n)$, incluso duplicando su tamaño cuando esté lleno. Para ver esto, necesitamos determinar la complejidad amortizada de agregar un elemento al arreglo.

Definimos una función potencial Φ que mide el tiempo "precargado" para un estado dado de la estrutura de datos. La función potencial ahorra tiempo para usarlo en futuras operaciones.

Entonces, el costo amortizado de una sola operación que cambia la estructura desde el estado h al estado h' equivale al costo real más el cambio de potencial, $\Phi(h') - \Phi(h)$. Considere una secuencia de n operaciones, cuyos costos reales son $c_1, c_2, \ldots, c_n$, y la estructura de datos cuyo estado inicial es h_0, y los estados resultantes de las operaciones son $h_1, \ldots, h_n$. El costo amortizado total es entonces $c_1 + c_2 + \ldots + c_n + (\Phi(h_1) - \Phi(h_0)) + (\Phi(h_2) - \Phi(h1)) + \ldots + (\Phi(h_n) - \Phi(h_n - 1)) = \sum_{i=1}^{n} c_i + \Phi(h_n) - \Phi(h_0)$. Por lo tanto, el costo amortizado sobreestima el costo real por la diferencia de potencial sobre toda la secuencia de operaciones. Si nos las arreglamos para que la diferencia de potencial sea ≥ 0, el costo total amortizado será siempre una cota superior por sobre el tiempo real.

Definimos la función potencial de forma que se "guarde tiempo" cada vez que el factor de carga del arreglo sea distinto de 0.5. En ese caso, tendremos suficiente ahorro para hacer el redimensionamiento. Si denotamos por n la cantidad de elementos en el arreglo, y m el número de espacios en el arreglo, la función potencial será $\Phi(h) = 2(n - m/2)$ (siempre se cumplirá que $n \geq m/2$, ¿por qué?).

Ahora, considere todos los casos que puedan ocurrir con todas las posibles operaciones. En este caso, la única operación posible es agregar un elemento. Si añadimos un elemento al arreglo de n elementos y m espacios tendremos $n + 1$ elementos. Considere los siguientes dos casos:

1. Si $n < m$, el potencial aumenta en 2, por lo que el costo amortizado es $1 + 2 = 3$
2. Si $n = m$, el arreglo se redimensiona, por lo que el costo real es $m + 1$; pero el potencial va de m a 2, por lo que el costo amortizado es $1 + m + 2 - m = 3$

En cada caso, el costo amortizado es $O(1)$. Si comenzamos con el arreglo vacío, su potencial inicial será 0. Como vemos que el potencial nunca puede disminuir el costo amortizado será una cota superior sobre el costo real.

3 Búsqueda Binaria dinámica

Sabemos que la Búsqueda Binaria en un arreglo ordenado funciona en tiempo logarítmico, pero *insertar* un nuevo elemento toma tiempo lineal en el largo del arreglo. Trataremos de mejorar este tiempo diseñando una nueva estructura.

Queremos diseñar una estructura que soporte las operaciones `Buscar` e `Insertar` en un conjunto de n elementos. Sea $k = \lceil \lg(n+1) \rceil$ y $n_{k-1} n_{k-2} \ldots n_0$ la representación binaria de n en k bits. Considere k arreglos ordenados $A_0, A_1, \ldots, A_{k-1}$ donde para cada $0 \leq i < k$, el tamaño de A_i es 2^i.

Cada arreglo o está vacío o completamente lleno, según si $n_i = 0$ o $n_i = 1$, respectivamente. Note que la cantidad de elementos en los k arreglos es $\sum_{i=0}^{k-1} n_i 2^i = n$.

1. Describa cómo buscar un elemento x en esta estructura y de su rendimiento en el peor caso.

2. Describa cómo insertar un elemento x en esta estructura. Analice su rendimiento en términos del peor caso y del costo amortizado.

3. (Propuesto) Describa cómo eliminar un elemento del arreglo y su rendimiento en el peor caso y costo amortizado.

3.1 Solución

1. La búsqueda se hace revisando cada uno de los arreglos ordenados.

   - Usando búsqueda binaria toma $O(\lg m)$ si m es el tamaño del arreglo
   - En el peor caso, x no está y todos los arreglos $A_0, \ldots, A_{k-1}$ están llenos, $k = \lg(n+1)$:

$$\begin{aligned} T(n) &= O(\lg 2^{k-1} + \lg 2^{k-2} + \ldots + \lg 2^1 + \lg 2^0) \\ &= O((k-1) + (k-2) + \ldots + 1 + 0) \\ &= O(k(k-1)/2) \\ &= O(\lg(n+1) \cdot \lg(n+1) - 1)/2) \\ &= O(\lg^2 n) \end{aligned}$$

   - El costo en el peor caso es $O(\lg^2 n)$

2. Creamos un nuevo arreglo de tamaño 1 que contenga el nuevo elemento a insertar. Si A_0 (de tamaño 1) está vacío, entonces reemplazamos A_0 con el nuevo arreglo; de otra forma, hacemos *merge* de amgos arreglos en un nuevo arreglo de tamaño 2. Si A_1 está vacío, usamos el nuevo arreglo creado, si no, hacemos *merge* entre A_1 y este nuevo arreglo, y así sucesivamente.

   - En el peor caso:
     - Hacer el merge entre 2 listas de tamaño m toma $2m$. Si todos los arreglos están llenos excepto el último, el tiempo en insertar un elemento es

$$\begin{aligned} T(n) &= O(2(2^0 + 2^1 + \ldots + 2^{k-2})) \\ &= O(2(2^{k-1} - 1)) \\ &= O(2^k - 2) \\ &= O(n) \end{aligned}$$

   - El costo amortizado mediante el método agregado:
     - Calculamos el costo total de n inserciones, partiendo con la estructura vacía.
     - Sea r la posición del 0 más a la derecha en la representación binaria $n_{k-1}n_{k-2}\cdots n_0$ de n, de forma que $n_j = 1$ para $j = 0, 1, \ldots, r-1$.
     - El costo de una inserción cuando n elementos ya están insertos es $\sum_{j=0}^{r-1} 2 \cdot 2^j = O(2r)$
     - Sin embargo, $r = 0$ la mitad del tiempo, $r = 1$ un cuarto de las veces, etc. Por lo que hay a lo más $\lceil n/2r \rceil$ inserciones por cada valor de r. El costo total de las n operaciones está acotado por

$$\sum_{r=0}^{\lceil \lg(n+1) \rceil} \left(\left\lceil \frac{n}{2^r} \right\rceil \right) = O(n \lg n)$$

  - Por lo que el costo amortizado por operación de inserción es $O(\lg n)$

- Otra forma de verlo es usando contabilidad de costos:
  - Cobramos k por inserción: \$1 paga la inserción en sí, y guardamos $(k-1)$ en el elemento para futuros merges.
  - Cada vez que hay un merge, el elemento se mueve hacia un arreglo de índice superior, y esto puede pasar a lo más $k-1$ veces, por lo que los $(k-1)$ alcanzan para pagar todas las veces que ocurrirá un merge. Como $k = \Theta(\lg n)$, tenemos un costo amortizado de $\Theta(\lg n)$ por inserción.