Distribución Pseudo-aleatoria de Datos

Año 2026

Introducción

En esta asignatura hemos tratado de resolver el siguiente problema: dada una relación cómo distribuimos sus elementos, en una estructura de almacenamiento, de forma tal de poder resolver eficientemente la localización de un elemento en la estructura. Recordemos que planteamos como muy importante el resolver la localización porque ella se usa en todas las rutinas que operarán sobre la estructura: *Pertenencia*, *Alta*, *Baja* y *Evocación Asociativa* (con asociante X y asociado Y, o sea, en el cual se aporta la componente X y se obtiene como respuesta la componente Y).

Hemos visto algunos tipos simples de estructuras de almacenamiento: listas desordenadas y ordenadas (secuenciales y vinculadas), árboles, listas de dos niveles, etc. Hasta ahora lo mejor que conseguimos, en el caso general, fue aplicar localizaciones que se resolvían examinando sólo partes del conjunto y no la totalidad, utilizando la información brindada por el orden.

Un nuevo tipo de estructura surge al hacer uso de una función total, que se define de la siguiente manera:

$$h : X \longmapsto \{0\} \cup \mathbb{I}_{M-1}$$

Este tipo de funciones suelen llamarse *funciones de pseudo-aleatorización* o *funciones de hashing* [1].

Si tenemos un conjunto de N elementos (N elementos a instalar) y la función devuelve un valor de un conjunto de M elementos, podemos ver que no será fácil obtener una función inyectiva porque existen M^N funciones posibles y de ellas sólo $\frac{M!}{(M-N)!}$ darán valores distintos para cada argumento.

En algunos casos es posible encontrar una función que sea también inyectiva y en ese caso la función se denomina *perfecta*. Si además es sobreyectiva se denomina *mínima perfecta*.

Como la función en general no es inyectiva, puede ocurrir que para dos elementos de X distintos, x_1 y x_2, $h(x_1)$ sea igual a $h(x_2)$. Cuando esto suceda diremos que x_1 y x_2 son *sinónimos* para la función h; aunque desde otro punto de vista podría decirse que dichos elementos son *homónimos*, porque por h valen lo mismo pero están asociados a elementos distintos.

Entonces, esta función se debe fabricar de forma tal que se produzcan pocos conflictos para conjuntos X muy irregulares.

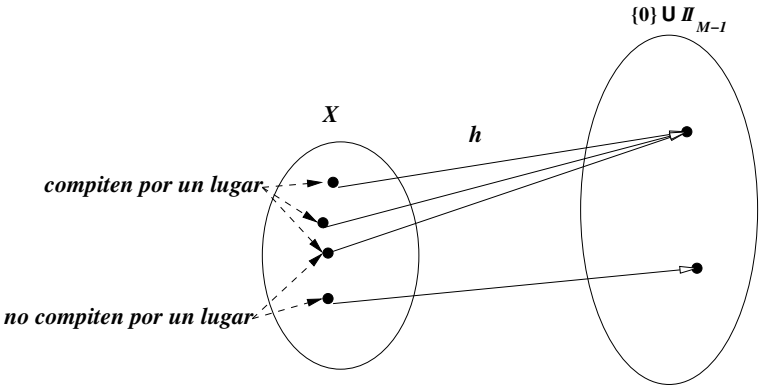
La estructura que analizaremos es una *distribución pseudo-aleatoria de datos*, porque hace uso de este tipo de funciones de pseudo-aleatorización. La idea es:

> *Si tenemos $h : X \longmapsto \{0\} \cup \mathbb{I}_{M-1}$, entonces reservamos espacio secuencial de M celdas y un elemento x del conjunto se almacenará en el lugar indicado por $h(x)$.*

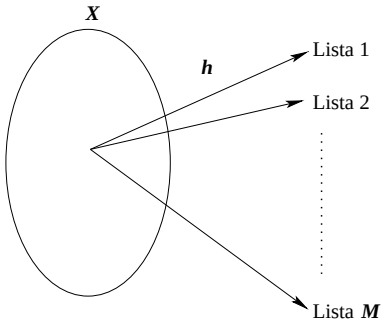
[1] Debido a ello, este tipo de estructuras comúnmente recibe el nombre de *Hashing* y las funciones se llaman así porque hacen un proceso (picadillo) sobre el argumento para obtener un valor numérico.

El problema que se nos puede presentar ahora es querer instalar un elemento x en $h(x)$ y que ese lugar ya esté ocupado por otro elemento. En ese caso se produce lo que llamamos *rebalse* (*overflow*).

Para minimizar los conflictos por el espacio debemos mantener a M mayor que la cardinalidad de X; porque si permito más valores a la función la posibilidad de conflictos baja. Además, si para un lugar hay más conflictos la probabilidad de conflicto en los otros lugares baja.



Pero aún así la posibilidad de rebalse de una celda sigue existiendo. Para solucionarlo se arma una lista con todos aquellos elementos que tienen igual valor para h. Por lo tanto, al aplicar la función al elemento de X obtengo un número natural en el intervalo $[0, \ldots, M-1]$ que me dice en qué lista debo buscar ese elemento y entonces el número de listas [2] es M .



Lo que veremos ahora son las representaciones que podemos usar para estas M listas.

Distintas clases de tratamientos de rebalse

Existen dos representaciones simples posibles para dichas listas, y cada una de ellas da origen a un esquema de *tratamiento de rebalse*:

1. Listas vinculadas y
2. Listas de alojamiento secuencial

La primera representación para las listas da origen al *rebalse separado*, en él las listas son independientes, por ello los elementos que tienen distinto valor por la función h no compiten entre sí por la utilización del espacio (el espacio es separado para cada lista).

[2] A la Lista 1 corresponden los elementos con valor de función h igual a 0, y así sucesivamente, a la Lista i corresponden los elementos con valor de función $i-1$.

La segunda representación, junto con la condición de que no reservo espacio para cada una de las listas en forma separada, origina el *rebalse abierto*; en él los elementos con distinto valor de función pueden competir por la utilización del espacio (el espacio es abierto para todas las listas). Por ello, dichas listas no son independientes.

Entonces, la solución completa será:

Pseudo-aleatorización + tratamiento de rebalse.

A cada elemento de la lista se lo denomina *balde* y éste será capaz de almacenar uno o varios elementos. A cada posición en un balde, capaz de almacenar un elemento o nupla, se la llama *ranura*; y nombraremos con r a la cantidad de ranuras por balde.

Por simplicidad para el análisis trabajaremos con una ranura por balde ($r = 1$), aunque hay ocasiones en que es conveniente que r sea mayor que 1 y esto, en general, suele ser impuesto por el problema [3].

En un rebalse separado se necesitan tener M accesos a listas, y además una cantidad de baldes suficientes para almacenar los N elementos del conjunto. En un tratamiento de rebalse abierto se necesitan M baldes, en los cuales poder almacenar los N elementos.

Definimos el *factor de carga* como la razón entre N y M (o sea, $\frac{N}{M}$) y lo denotamos con ρ (en algunos libros aparece como α). En caso de que la cantidad de ranuras por balde sea mayor que 1 ($r > 1$) se obtiene como: $\rho = \frac{N}{M \cdot r}$.

En un rebalse separado ρ puede ser mayor que 1, en cambio en un rebalse abierto debe mantenerse siempre menor 1 (teóricamente puede ser cercano o igual a 1, pero en esos casos es sumamente ineficiente).

El esfuerzo medio de evocación en estas estructuras se obtiene como una función de ρ y además si ρ crece el esfuerzo también lo hace; por lo tanto, la función del esfuerzo debe ser monótona creciente estricta como función de ρ.

En un rebalse separado existen M listas vinculadas. Entonces la pertenencia básicamente será:

Pertenencia (**in** x, **out** *éxito*)
    $i \leftarrow h(x)$
    pertenencia-en-lista-vinculada (i, x, *éxito*)

En este caso la longitud media de la lista es aproximadamente $\frac{N}{M}$ (ρ).

En un rebalse abierto, en el caso general, existen M listas secuenciales [4]. Entonces la pertenencia será:

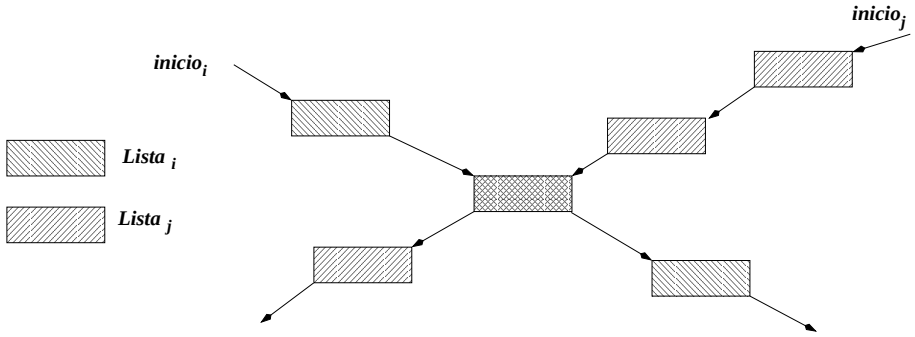
Pertenencia (**in** x, **out** *éxito*)
    $i \leftarrow h(x)$
    pertenencia-en-lista-secuencial (i, x, *éxito*)

[3] Si la estructura de almacenamiento se aloja en memoria principal no hay diferencia en considerar una o más ranuras por balde; pero, si se almacena en disco, hay situaciones en que la estructura se comporta mejor si tiene varias ranuras por balde.

[4] En este caso las listas de alojamiento secuencial tienen la terminación controlada por contenido; o sea, que existe una marca que indica el fin de la lista.

Análisis de las distintas técnicas de tratamiento de rebalse abierto

Vamos ahora a analizar más en detalle el tratamiento de rebalse abierto. Habíamos dicho que en él existían distintas listas de alojamiento secuencial para las cuales los espacios se entrecruzaban, por no tener reservado espacio individual para cada una de las listas. Entonces tenemos:



En un rebalse abierto la posición inicial de una lista (*inicio*) se obtiene aplicando la función h al argumento, entonces la Lista i del ejemplo empezaría en el i-ésimo balde de la estructura, y la Lista j en el j-ésimo balde.

En un rebalse abierto se obtiene de manera analítica cuál es la siguiente celda a examinar, y existen distintas formas de hacerlo. Veremos cuatro de las más importantes, y dependiendo de cuál de ellas se use será el nombre que recibe el tratamiento particular de rebalse abierto:

1. Se suma 1 a la posición anterior, por consiguiente el siguiente elemento de la lista es el siguiente balde de la estructura (también podría usarse -1).
2. Se suma un valor que depende de qué número de elemento dentro de la lista quiero alcanzar; si quiero alcanzar el segundo elemento suma 1, si quiero el tercero suma 2, ..., si quiero el k-ésimo suma k-1.
3. Se suma una cantidad que depende del valor devuelto por otra función pseudo-aleatoria $f : X \longmapsto \mathbb{I}_{M-1}$, que es el paso de avance para ese elemento.
4. Se obtiene de una de dos formas posibles:
   - sumando, a la posición de inicio, una cantidad que depende del valor devuelto por otra función pseudo-aleatoria $f : X \times \mathbb{I}_{M-1} \longmapsto \mathbb{I}_{M-1}$, al aplicarla al elemento y al número de intento correspondiente;
   - o de acuerdo al valor devuelto por una función $f : X \times \mathbb{I}_M \longmapsto \{0\} \cup \mathbb{I}_{M-1}$. O sea, al aplicarla al elemento y al número de intento, f da la posición siguiente en la lista.

Los métodos reciben los siguientes nombres de acuerdo a la función sucesora utilizada:

Función de avance utilizada	Nombre del método de rebalse abierto
1	lineal
2	cuadrático
3	paso realeatorizado
4	realeatorizado total

Pueden existir otros métodos de tratamiento de rebalse abierto, y además combinaciones de estos métodos.

Si una lista se mantuviera ordenada y fuera alojada secuencialmente se podría aplicar búsqueda binaria pero, en éste caso, como cada lista puede estar contaminada con elementos de otra lista y además son cortas, no funcionaría bien. Por consiguiente, no tiene sentido mantenerlas ordenadas.

Todos estos métodos deben tener en cuenta que todas las listas deben poder recircular en el espacio disponible, porque existe un único espacio para todas ellas. Por lo tanto, en aquellos métodos que se realizan sumas para obtener la siguiente posición se debe aplicar, al resultado de dicha suma, la función **mod** M; ya que ésta devuelve el resto de dividir por M el número al cuál se aplica, o sea un valor entre 0 y $M - 1$.

Además, luego de M intentos se debe haber pasado por los M baldes. Esto es fácil asegurarlo para el rebalse abierto lineal (con avance 1 o -1); pero en el caso del cuadrático, y del de paso realeatorizado, tiene mucho que ver el valor de M. En particular, en el rebalse abierto de paso realeatorizado cada posible valor de avance debe ser coprimo con el tamaño de la tabla (M); es decir que el mcd entre el avance y M debe ser 1, y la forma más sencilla de lograrlo es tomando a M como un número primo [5]. Se puede usar para el método lineal un avance constante distinto de 1 o -1, pero en ese caso se deberá tener en cuenta la misma consideración que en el caso de paso realeatorizado.

El segundo método se denomina cuadrático porque para llegar a obtener el i-ésimo elemento, dentro de la lista correspondiente, he debido sumar a la posición inicial (dada por la aplicación de la función h) los $i - 1$ primeros números naturales lo cual es:

$$\sum_{j=1}^{i-1} j = \frac{i \cdot (i-1)}{2} = \frac{i^2 - i}{2}$$

Análisis de por qué surgen los distintos métodos

Para cada uno de los métodos veamos la secuencia de los baldes visitados que se obtendría para 3 elementos x, x' y x'', para los cuales $h(x) = i$, $h(x') = i + 1$ y $h(x'') = i + 2$.

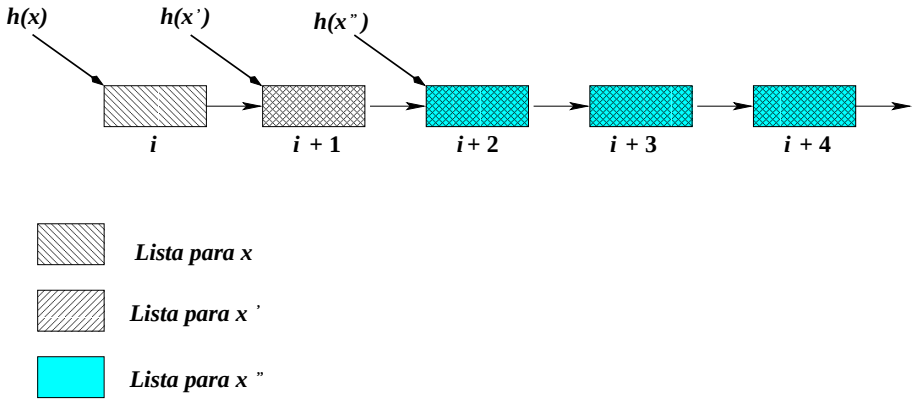
1. Método de rebalse abierto lineal:

   secuencia para x : $i, i + 1, i + 2, i + 3, \ldots$

   secuencia para x' : $i + 1, i + 2, i + 3, \ldots$

   secuencia para x'' : $i + 2, i + 3, i + 4, \ldots$

[5] Si M es un número primo cualquier número entre 1 y $M - 1$ será coprimo con él.



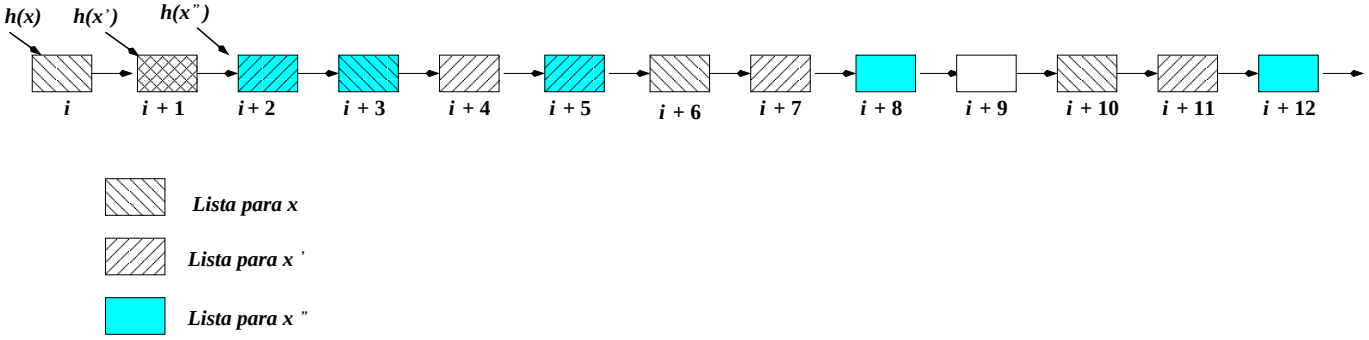
Vemos que ambas listas claramente coinciden en casi la totalidad de sus elementos, es decir hay muchas *colisiones* entre listas distintas. A este problema se lo denomina *agrupamiento primario* (*primary clustering*). Entonces, surge el siguiente método para tratar de minimizarlo:

2. Método de rebalse abierto cuadrático:

   secuencia para x : $i, i+1, i+3, i+6, i+10, \ldots$

   secuencia para x' : $i+1, i+2, i+4, i+7, i+11, \ldots$

   secuencia para x'': $i+2, i+3, i+5, i+8, i+12, \ldots$



En este caso se ve que mejoramos bastante respecto del caso anterior, ya que elementos con valor de h distinto empiezan a tener menos espacios comunes. Pero, también es claro que dados dos elementos cuyo valor de función h sea el mismo deben recorrer la misma secuencia y, por lo tanto, competir por los mismos espacios. A este problema se lo denomina *agrupamiento secundario* (*secondary clustering*). Lo que debemos tratar de hacer es minimizar las colisiones, de forma tal de que las listas se hagan más cortas [6].

Entonces, surge la idea de que dos elementos x y x', con igual valor de función $h(x) = h(x') = i$, sigan distintas secuencias:

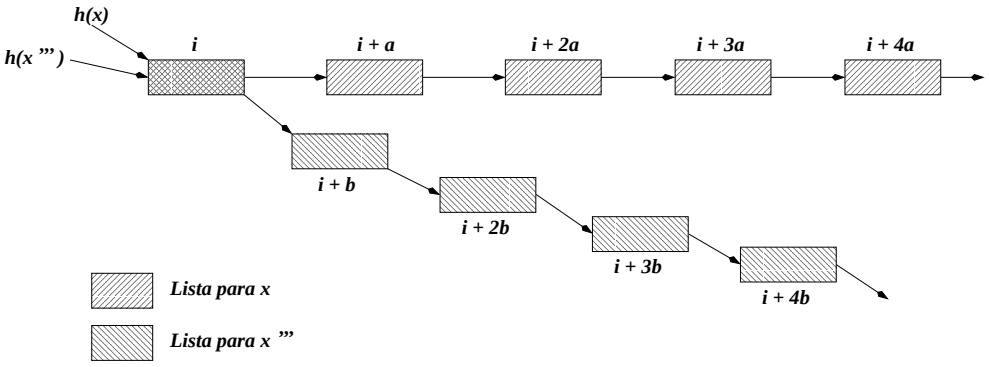
[6] Al tener listas más cortas se mejoran los distintos esfuerzos a considerar sobre la estructura.

3. Método de rebalse abierto de paso realeatorizado:

   Sea f la función que da el avance para los elementos, pero en este caso $f : X \longmapsto \mathbb{I}_{M-1}$, y supongamos que $f(x) = a$ y $f(x''') = b$, y que $a \neq b$.

   secuencia para x : i, $i + a$, $i + 2a$, $i + 3a$, ...

   secuencia para x''' : i, $i + b$, $i + 2b$, $i + 3b$, ...



   Es evidente que para dos elementos cuyo valor de función h coincidía, pero dónde la función f no, en la secuencia tienden a disminuir las colisiones.

   El inconveniente que se presenta aquí es que se debe encontrar otra función f, y que esto en algunos casos no es muy sencillo. Como la función f debe devolver un valor para el avance, queda en evidencia que él mismo no puede ser 0 ni M, porque en ambos casos no avanzaría.

   Este método nuevamente presenta un problema, que aunque no es muy probable, es posible. Éste ya no necesariamente se presenta en elementos cuyo valor de h coincidía, pero que sí mantienen una relación. Por ejemplo, si $h(x) = i$ y $h(x''') = i + a$, y $f(x) = f(x''') = a$; es claro que aquí se mantienen los supuestos que se hacían para este caso (porque estamos diciendo de antemano que $h(x) \neq h(x')$); pero es inevitable que las listas que recorran ambos elementos tengan muchas posiciones comunes. Otro caso podría suceder si, por ejemplo, $h(x) = i$ y $h(x''') = i + a$, y $f(x) = b$ y $f(x''') = a$, pero $b = 2a$. Por lo tanto, nuevamente se trata de mejorar la situación y entonces aparece el siguiente método:

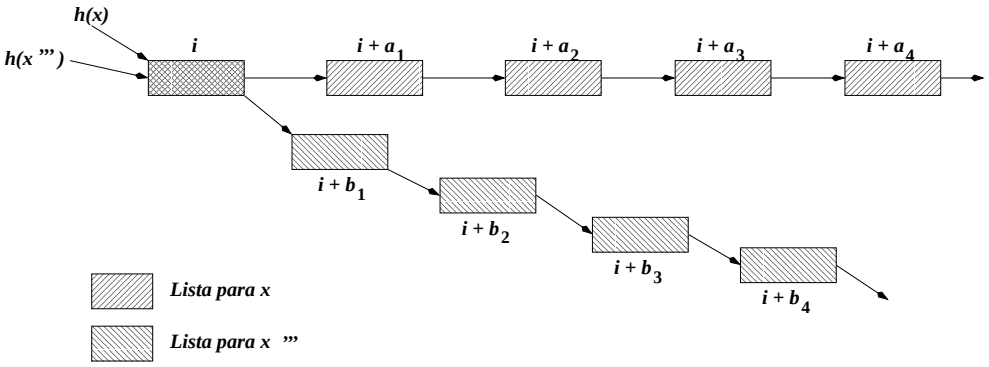
4. Método de rebalse abierto realeatorizado total:

   Surge así el método de tratamiento de rebalse realeatorizado total, que tiene en cuenta ya no sólo cuál es el elemento al que se aplica la función, sino además tiene que ver con el número de intento que se está realizando.

   De acuerdo a esta técnica en cada intento se obtiene un número que pertenece al rango de valores $[1, \ldots, M - 1]$ y que en cada paso se debe asegurar que sea distinto; ya que sino no será posible asegurar que luego de M intentos se visiten todas las posiciones. Es fácil ver que, en este caso, no es necesario pedir condiciones respecto de M por la manera en que se calcula la nueva posición a consultar. Así, se evidencia que mediante f se obtiene una secuencia de todas las posiciones en el rango $[0, \ldots, M - 1]$ en un determinado orden, o sea una permutación posible de los valores entre 0 y $M - 1$.

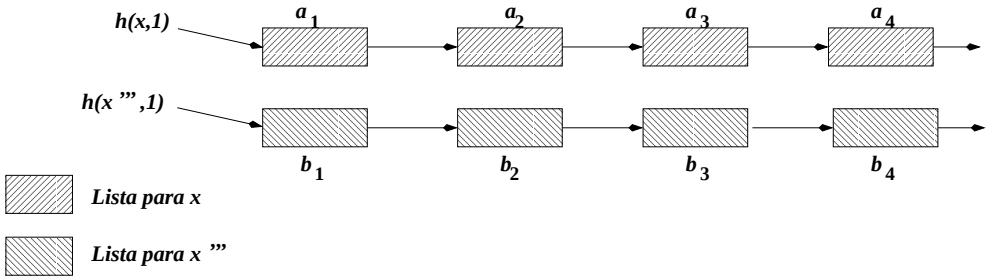
Trabajando con desplazamientos aleatorios desde una posición:

Supongamos $h(x) = h(x''') = i$, y los valores de $f(x, j)$ denotados por a_j, y los valores para $f(x''', j)$ denotados por b_j:



Trabajando con posiciones aleatorias:

Supongamos $h(x, j) = a_j$, y los valores para $h(x''', j) = b_j$, tendríamos:



La diferencia más importante que se plantea es: si dos elementos colisionan en una posición es muy probable que en el intento siguiente no lo hagan, porque no arrastran un avance fijo.

Pertenencia, Localizaciones, Altas y Bajas

Haremos ahora algunas observaciones sobre el comportamiento de estas estructuras al realizar operaciones de pertenencia, localizaciones, altas y bajas.

- Rebalse separado:

  Como en un rebalse separado se mantienen M listas vinculadas, realizar una de estas operaciones en la estructura es realizar dicha operación sobre la lista vinculada correspondiente.

  Por lo tanto, y dado que como ya hemos visto en general no sirve de nada mantener a las listas vinculadas ordenadas [7], el hacer un alta (o una baja) en la estructura es simplemente hacer un alta (o una baja) en la lista vinculada desordenada que corresponda. Responder a la pertenencia de un elemento (o una localización) se traduce en responder dicha pertenencia (o una localización) sobre la lista vinculada desordenada correspondiente.

[7] Salvo que se quiera bajar el esfuerzo medio de fracaso.

- Rebalse abierto:

  Consideremos primero la localización, dado que esta operación se utiliza en las otras operaciones. Para cualquiera de los distintos tipos de rebalse abierto la localización realiza una búsqueda en una lista desordenada "secuencial" [8]. Por lo tanto, esta rutina realiza una iteración que en cada paso debe verificar no haber llegado al final de la lista y sólo si es así se pregunta si el elemento alcanzado es igual al buscado.

  Entonces, los puntos de parada de dicha rutina serán: o haber llegado al fin de la lista (en cuyo caso la búsqueda fracasó), o haber encontrado el elemento (en cuyo caso tuvo éxito). Lo que resta preguntarse es cómo se determina si se llegó al final de la lista o no, dado que no conocemos cuántos elementos hay en dicha lista (porque una lista está contaminada con elementos de otras). Para ello, debemos pensar en qué situación estamos seguros que la lista terminó y esto será cuando se haya alcanzado una celda vacía [9].

  Analicemos más en detalle qué celdas consideraremos como vacías. Si en la estructura no han habido bajas, celda vacía será aquella que nunca se usó pero, si han habido bajas, *¿una celda cuyo elemento ha sido dado de baja la podremos considerar como que indica un fin de lista?* La respuesta es no; ya que en caso de considerarla así no podremos recuperar los elementos cuya localización pasó por esa posición, encontrándola ocupada, cuando se incorporaron a la estructura.

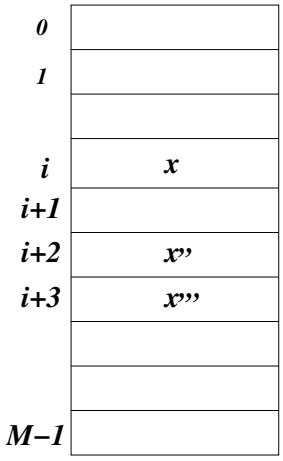
  Veamos un ejemplo que evidencia claramente la situación planteada sobre un tratamiento de rebalse abierto lineal. Sean x, x', x'' y x''' elementos del conjunto X y sus correspondientes valores de función: $h(x) = h(x') = i$, $h(x'') = h(x''') = i + 1$. Al realizar las altas, en el orden dado, la estructura resultante sería:

0	
1	
i	***x***
i+1	***x'***
i+2	***x''***
i+3	***x'''***
M−1	

  Si ahora damos de baja el elemento x' de la estructura y dejamos la celda en la misma condición que la que se tenía antes de que lo diéramos de alta, la estructura quedaría:

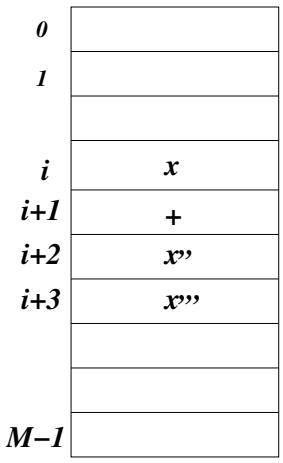
[8] Dado que se obtiene analíticamente la siguiente celda a examinar.

[9] De no ser así, el elemento buscado se hubiera colocado en dicha posición. Dicha celda tendrá una marca que indique que está vacía.



Entonces, al buscar ahora a x'' miraríamos primero en la posición $i+1$ y como está vacía diríamos que fracasamos, cuando en realidad x'' se encuentra en la posición $i+2$; porque cuando se lo dio de alta, él encontró la posición $i+1$ ocupada. Por lo tanto, las celdas que fueron ocupadas alguna vez no pueden quedar en igual condición que aquellas que nunca se usaron.

Si por ejemplo marcamos las celdas dadas de baja con el símbolo “+”, la estructura anterior quedaría:



Así una celda de la estructura se puede encontrar en una de tres situaciones: *“virgen”* u *“ocupada”* o *“libre”*. Y la localización sólo fracasará al alcanzar una celda “virgen”.

Si por ejemplo marcamos las celdas vírgenes o nunca usadas con el símbolo “*”, la estructura anterior quedaría:

0	*
1	*
	*
i	***x***
i+1	**+**
i+2	***x'***
i+3	***x''***
	*
	*
M−1	*

Ahora podemos observar que, para cualquiera de los distintos tipos de rebalse abierto, un elemento se debería dar de alta en la ***primera celda desocupada***.

En comienzo podemos pensar que dicha posición desocupada corresponde a la posición en donde se detuvo la localización (realizada para verificar que dicho elemento no se encontraba ya en la estructura), que sería una celda "virgen"; pero si mientras se desarrollaba la localización se pasó por una celda "libre" podemos dar de alta en esa posición. Así, lograremos mejorar el costo de búsqueda del elemento (respecto de darlo de alta en una celda "virgen") y también el costo de las búsquedas que fracasan; y además no afectaría a futuras localizaciones exitosas de otros elementos.

Por lo tanto, se debe recordar al hacer la localización cuál fue la primera celda "libre" que se encontró, si es que hay una, en el recorrido realizado. Si la búsqueda fracasa la posición que se devuelva en el parámetro será la posición de dicha celda "libre", en caso de que exista, o en otro caso la de la celda "virgen" en donde se detuvo la búsqueda.

Así un alta realizará una localización y si ésta última fracasa el elemento se dará de alta en la posición devuelta en el parámetro.

Una baja realizará la localización y si es exitosa colocará la marca de "libre" en la posición devuelta como parámetro.

Reconocimientos

El presente apunte se realizó tomando como base notas de las clases del **Ing. Hugo Ryckeboer** en la Universidad Nacional de San Luis.