Estructuras de Datos y Algoritmos

Año 2022

Estructuras de Datos Aleatorias: Skip Lists

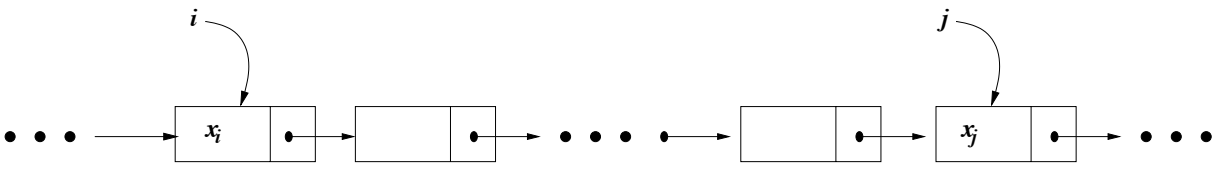
Introducción

En esta asignatura tratamos 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. Resolver la *Localización* es muy importante porque se usa en todas las rutinas que operarán sobre la estructura: *Pertenencia*, *Alta*, *Baja* y *Evocación Asociativa* (con asociante X y asociado Y; es decir, 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). De éstas, la más eficiente hasta ahora fue la lista de alojamiento secuencial ordenada en la que podíamos aplicar un tipo de búsqueda llamada búsqueda binaria (por bisección o trisección). La búsqueda permitía resolver la localización examinando sólo partes del conjunto y no la totalidad, utilizando la información brindada por el orden.

Mantener dicha relación en una lista secuencial ordenada nos posibilitaba tener un esfuerzo medio y máximo de localización (exitosa y no exitosa) de $O(\log N)$ celdas consultadas, pero teníamos un esfuerzo máximo de alta (o baja) de $O(N)$ corrimientos de celdas para realizar la modificación estructural. El esfuerzo de localización es muy bueno frente al que tenemos en otras estructuras, pero los esfuerzos máximos de alta y baja son malos. Por otra parte, la lista vinculada tiene muy buenos esfuerzos de alta y baja, pero la localización es $O(N)$. Por lo tanto, podemos encontrar una nueva estructura que guarde los beneficios de la localización de $O(\log N)$ celdas consultadas de la lista secuencial ordenada, pero que logre mejorar los esfuerzos máximos de alta y baja acercándose a los de la lista vinculada.

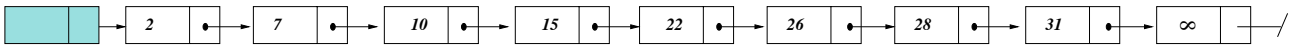
Claramente es imposible realizar una búsqueda binaria, ya sea por bisección o por trisección, sobre una lista vinculada ordenada. Esto es porque no existe manera de calcular desde las direcciones de dos elementos i y j, con valores x_i y x_j para X, la dirección de un elemento con un valor de X intermedio entre x_i y x_j. La razón de ello es que las celdas que contienen elementos de la lista pueden provenir arbitrariamente desde cualquier parte de la memoria.



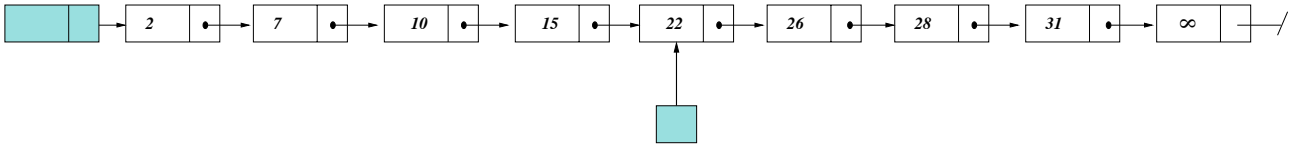
De hecho, ésa fue la razón que nos llevó a acotar el uso de dicho tipo de listas; a pesar de que las modificaciones estructurales sobre ellas eran muy eficientes (de orden $O(1)$).

Si tenemos, por ejemplo, la siguiente lista vinculada ordenada[1]; el único tipo de examinación posible para la localización es secuencial y, por lo tanto, el peor caso de localización exitosa sería de N celdas consultadas (el peor caso de localización que fracasa sería de $N + 1$ celdas consultadas).

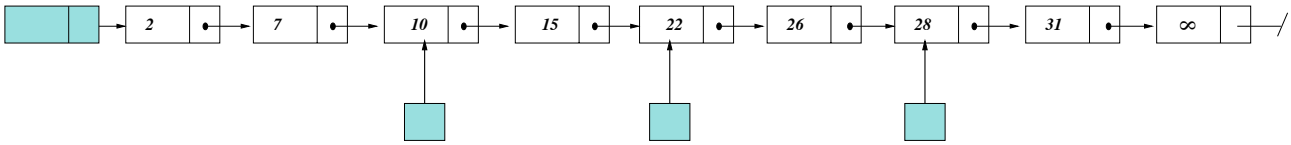
[1] La lista la armamos con una celda de cabecera y con marca de fin de lista (∞).



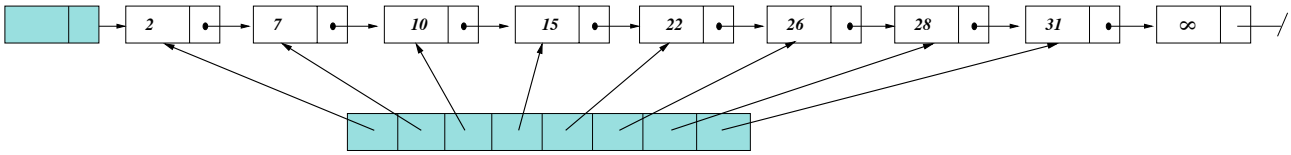
Con el fin de mejorar los esfuerzos medios y máximos de localización, podríamos pensar en mantener un apuntador a la posición media de la lista. Eso nos permitiría que la búsqueda se inicie comparando el elemento buscado contra el que se encuentra en esa posición media y luego, si el elemento buscado no es ése, continuar la búsqueda en la parte inicial de la lista o en la final, dependiendo de si el x buscado es menor o mayor que el elemento de la posición media. Así reduciríamos el esfuerzo máximo aproximadamente a la mitad ($\approx N/2$).



Repitiendo este razonamiento podríamos intentar mantener apuntadores adicionales en las posiciones medias de la parte inicial y de la final, lo que nos permitiría reducir el esfuerzo máximo aproximadamente a $N/4$.

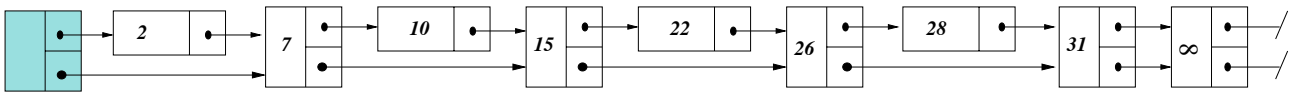


Si seguimos aplicando esta técnica podríamos llegar a mantener N apuntadores adicionales, uno a cada posición de la lista. Si los mantenemos ordenados por la posición podríamos alojarlos secuencialmente, para poder ubicar fácilmente el apuntador correspondiente a la posición que nos interese.

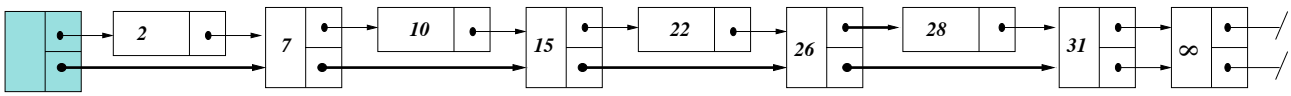


Esto se asemeja a una *Lista Invertida*. Por lo tanto no tendría sentido, ya que volveríamos a un esfuerzo máximo de alta y baja de orden $O(N)$ (corrimientos de punteros), usando más espacio que el necesario para una Lista Invertida común.

Supongamos, sin embargo, que a dicha lista vinculada la estructuramos de manera tal que el segundo, el cuarto, el sexto, ... elemento, se vinculen por un segundo puntero. Entonces, se vinculan todos los elementos en posiciones pares de la lista mediante este nuevo puntero (o lo que es lo mismo, aquellos elementos que se encuentran en posiciones múltiplos de 2 en la lista original).



Al localizar, por ejemplo, el elemento 28 haríamos el recorrido que está resaltado en la siguiente figura; siendo éste el elemento cuyo costo en cantidad de celdas consultadas sería el máximo:



En esta estructura podríamos bajar los esfuerzos de localización casi a la mitad respecto de la lista vinculada ordenada; ya que aprovechando el orden y usando este nuevo puntero podemos dar saltos a través de la lista de a 2 celdas, mientras se pueda, y luego a lo sumo mirar un elemento más. En una lista vinculada estructurada de esta manera el peor caso de localización exitosa sería $\left\lceil \frac{N}{2} \right\rceil + 1$ celdas consultadas.

Por consiguiente, vemos que sólo agregando un puntero más en algunas celdas logramos bajar a la mitad los esfuerzos de localización.

En el caso general tendríamos los elementos vinculados a través de 2 listas. Una lista L_2 que contiene sólo algunos elementos del conjunto y la otra L_1 que vincula a todos los elementos. La lista L_2 se usa para dar saltos más grandes sobre el conjunto, mientras se pueda. La lista L_1 finalmente se revisa cuando no se ha podido seguir avanzando sobre la otra lista, revisando los celdas que quedaron al medio de la penúltima y el última celda revisada en la lista L_2; es decir, las celdas que contienen valores de X mayores que el valor de la penúltima celda visitada de L_2 y menores que el de su última celda.

Si consideramos la mejor manera de vincular los elementos usando 2 listas, deberíamos analizar cuál sería la manera de armarlas para obtener los menores esfuerzos de localización posibles. Sean $|L_1|$ y $|L_2|$ los largos de las listas L_1 y L_2 respectivamente. Si la cantidad de celdas de L_1 que existen entre dos celdas de L_2 es pareja, se espera que hayan $|L_1|/|L_2|$ celdas de L_1 entre dos de L_2. En el peor caso de localización visitaríamos en cada nivel la mayor cantidad de celdas posibles. Entonces, se debería recorrer la lista L_2 completa (mirar $|L_2|$ celdas) y la cantidad de celdas de L_1 que habría entre dos de L_2 (mirar $|L_1|/|L_2|$ celdas más). Así llegaríamos aproximadamente a: $\approx |L_2| + \frac{|L_1|}{|L_2|}$. Si analizamos cuánto debería valer $|L_2|$ para minimizar el esfuerzo máximo, podemos ver que si consideramos que $|L_2|$ sea chico, el primer término se achica mientras que el segundo se hace mayor y a la inversa, si agrandamos $|L_2|$ el primer término se hace más grande mientras que el segundo se achica. Por lo tanto, existe un compromiso entre lo que sucede en ambos términos en función del valor de $|L_2|$. Así, el mínimo esfuerzo se obtendrá cuando ambos términos se igualen[2]:

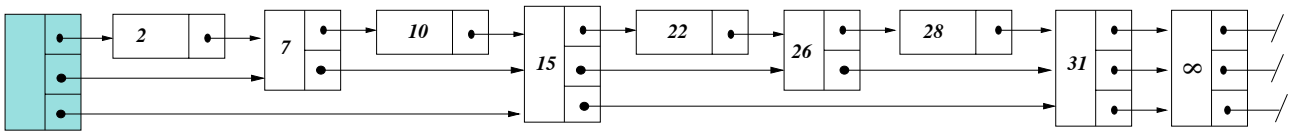
$$|L_2| = \frac{|L_1|}{|L_2|}$$

vemos que esto se consigue cuando: $|L_2|^2 = |L_1|$, pero como en $|L_1|$ deben estar todos los elementos: $|L_1| = N$, así $|L_2| = \sqrt{N}$ y entonces el esfuerzo máximo de localización sería de $2\sqrt{N}$. Por lo tanto, lo mejor que podríamos hacer es que haya cada $\sqrt{N}$ elementos de L_1 un elemento de L_2.

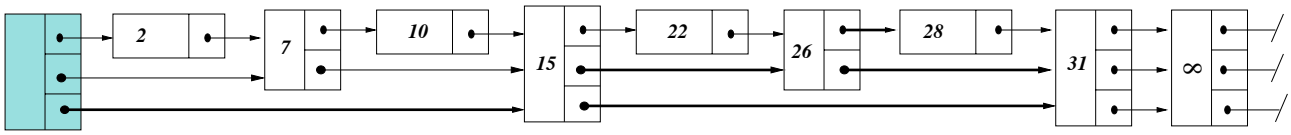
Siguiendo con este razonamiento, podríamos vincular los elementos por una tercera lista. Si consideramos que ahora vinculamos de acuerdo a la tercer lista los elementos que se encuentran en posiciones múltiplos

[2] Pueden diferir sólo en factores constantes.

de 4, en la lista vinculada ordenada original, usando el nuevo puntero de esta nueva lista. Obtendríamos así la siguiente estructura, sobre el ejemplo:



Si en dicha lista tratamos nuevamente de localizar el elemento 28 haríamos el recorrido que se ha resaltado en la siguiente figura:



Se puede mostrar que nuevamente logramos bajar a casi la mitad los esfuerzos de localización ya que, por ejemplo, el peor caso de localización exitosa en esta nueva estructura sería $\left\lceil \frac{N}{4} \right\rceil + 2$ celdas consultadas.

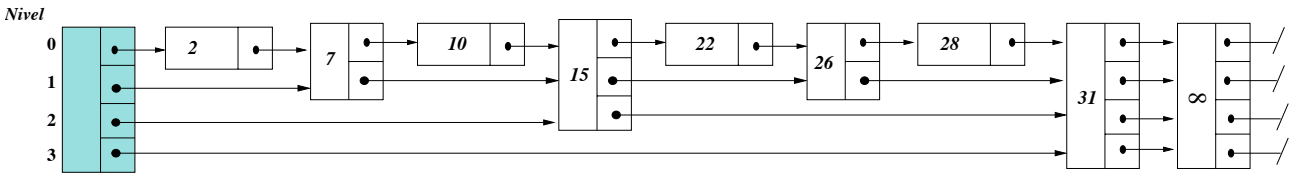
Si ahora analizamos qué es lo mejor que podríamos hacer manteniendo 3 listas vinculadas, una lista L_3 que contiene sólo algunos pocos elementos del conjunto (para dar saltos más grandes), una lista L_2 que mantiene, además de los elementos de L_3, algunos elementos más (para seguir dando saltos un poco más pequeños) y la otra L_1 que vincula a todos los elementos. En este caso la lista L_3 se usa para dar saltos más grandes sobre el conjunto, mientras se pueda, la L_2 se usa para dar algunos saltos más, no tan grandes, entre los elementos que no pudieron descartarse desde las consultas a elementos de L_3 y finalmente la lista L_1 para revisar los últimos elementos no descartados.

Para analizar cuál es la mejor manera de vincular los elementos usando las 3 listas, deberíamos analizar cúantos elementos se deberían manetner en cada una de ellas. Nuevamente, sean $|L_1|$, $|L_2|$ y $|L_3|$ los largos de las listas L_1, L_2 y L_3 respectivamente. Si la cantidad de celdas de L_1 que existen entre dos celdas de L_2 es pareja y la cantidad de celdas de L_2 que existen entre dos celdas de L_3 también es pareja, se espera que hayan $|L_1|/|L_2|$ celdas de L_1 entre dos de L_2 y $|L_2|/|L_3|$ celdas de L_2 entre dos de L_3. En el peor caso de localización visitaríamos en cada nivel la mayor cantidad de celdas posibles. Entonces, se debería recorrer la lista L_3 completa (mirar $|L_3|$ celdas), al bajar a la siguiente lista se revisan $|L_2|/|L_3|$ celdas de L_2 y finalmente la cantidad de celdas de L_1 que habría entre dos de L_2 (mirar $|L_1|/|L_2|$ celdas más). Así llegaríamos a: $\approx |L_3| + \frac{|L_2|}{L_3} + \frac{|L_1|}{|L_2|}$. Ahora debemos analizar cuánto debería ser $|L_3|$ y $|L_2|$ de manera tal de minimizar el esfuerzo máximo de localización. Si consideramos que $|L_3|$ sea chico, el primer término se achica mientras que el segundo se hace mayor y a la inversa si agrandamos $|L_3|$ el primer término se hace más grande mientras que el segundo se achica, lo mismo ocurriría con $|L_2|$ y $|L_1|$. Por lo tanto, existe un compromiso entre lo que sucede en los tres términos en función del valor de $|L_3|$. Así, el mínimo esfuerzo se obtendrá cuando todos los términos se igualen:

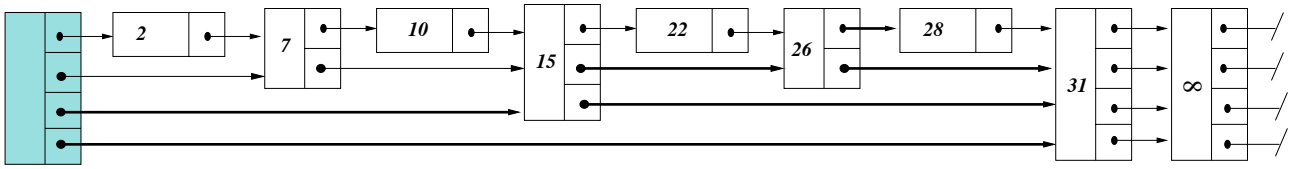
$$|L_3| = \frac{|L_2|}{|L_3|} = \frac{|L_1|}{|L_2|}$$

y esto se consigue cuando: $|L_3| = \sqrt[3]{N}$ y ambas razones entre largos $\frac{|L_2|}{|L_3|} = \frac{|L_1|}{|L_2|} = \sqrt[3]{N}$ y entonces [3] el esfuerzo máximo de localización sería de $3\sqrt[3]{N}$. Por lo tanto, lo mejor que podríamos hacer es que haya cada $\sqrt[3]{N}$ elementos de L_1 un elemento de L_2 y cada $\sqrt[3]{N}$ celdas de L_2 una de L_3 y así si $|L_3| = \sqrt[3]{N}$, $|L_2| = (\sqrt[3]{N})^2$ y $|L_1| = N$ (es decir, $|L_1| = (\sqrt[3]{N})^3 = N$).

Siguiendo con el ejemplo que estamos utilizando y con este razonamiento podríamos hacer uso de un nuevo puntero que vincule aquellos elementos que se encuentran en posiciones múltiplos de 8 en la lista original, obteniendo así la siguiente estructura:



Al localizar, nuevamente, el elemento 28 haríamos el recorrido que está resaltado en la siguiente figura:



Para esta cantidad de elementos no parece ayudarnos mucho en la localización el mantener este nuevo puntero. Esto es porque, mediante él, no logramos más que saltar al final de la lista. Con mayor cantidad de elementos sería más claro el beneficio obtenido.

Sin embargo, en general, nuevamente logramos bajar los esfuerzos de localización a la mitad, respecto de la última estructura del ejemplo analizada, ya que el peor caso de localización exitosa sería de $\left\lceil \frac{N}{8} \right\rceil + 3$ celdas consultadas.

Con esta idea logramos mejorar los esfuerzos de localización, pero a costa de aumentar la cantidad de punteros; es decir, de aumentar el espacio utilizado para la representación de la estructura.

Llevando la idea general al límite, podríamos pensar en tener N niveles de listas vinculadas y así mejorar aún más los costos de búsqueda. Sin embargo, en ese caso el esfuerzo máximo de localización sería de $N\sqrt[N]{N} = N^{(1+\frac{1}{N})} > N$, o sea que no logramos mejorar, sino que terminamos empeorando los costos respecto a una lista vinculada común. Esto significa que debemos hallar una situación itermedia en la cantidad de listas, de modo tal que no empeore los costos de la lista original; vemos que, si mantenemos $\log N$ listas, el esfuerzo máximo en ese caso sería de $\log N\, \sqrt[\log N]{N} = 2 \log N$, ya que $2^{\log N} = N$.

En general, si tenemos una lista con N elementos, tendremos a lo sumo $\log N$ punteros por celda. Ahora bien, en ese caso el esfuerzo mínimo se alcanza cuando el largo de la última lista y las razones

[3] Al multiplicar los tres términos, si son iguales, debe dar como resultado los N elementos.

entre los largos de las listas de niveles consecutivos son, salvo factores constantes, iguales. Llamaremos a esa razón entre largos r. Por ello, al hacer el producto de cuántas celdas tiene la última lista y las razones de los largos de las listas de niveles consecutivos debe ser N, si hay $\log N$ términos en el producto tendríamos que $r^{\log N} = N$ y entonces cada una de las razones de los largos de las listas de niveles consecutivos es de $r = 2$. Es decir, si L_1 debe tener N elementos, L_2 debe tener la mitad ($|L_2| = \frac{N}{2}$ porque $\frac{|L_1|}{|L_2|} = \frac{N}{|L_2|} = 2$), L_3 la cuarta parte ($|L_3| = \frac{N}{4} = \frac{N}{2^2}$), así siguiendo $|L_i| = \frac{N}{2^{(i-1)}}$ y finalmente $L_{\log N} = \frac{N}{2^{(\log N - 1)}} = 2$. Por lo tanto, si tenemos $\log N$ listas y el esfuerzo máximo es de $2 \log N$, en la lista de nivel mayor debemos revisar 2 celdas, en el siguiente nivel otras 2 celdas y así siguiendo hasta que en el último nivel deberíamos revisar 2 celdas más.

Por lo tanto, en la mayoría de las celdas no se necesitarían tantos punteros, porque la mitad de los elementos necesitan sólo un único campo de puntero, y de los restantes elementos sólo la mitad necesitaría un puntero adicional, la cuarta parte 1 puntero más (3 en total) y así siguiendo. Por lo tanto, hay $\frac{N}{2}$ celdas con sólo 1 puntero, $\frac{N}{4}$ celdas con 2 punteros, $\frac{N}{8}$ con 3 punteros; y en general $\frac{N}{2^i}$ con i punteros.

Para analizar cuántos punteros estamos usando en total para almacenar N elementos, podemos plantear la siguiente fórmula:

$$N^o \textit{de punteros} = \sum_{i=1}^{\log N} i \cdot \frac{N}{2^i} = N \cdot \sum_{i=1}^{\log N} \frac{i}{2^i}$$

para resolverla necesitamos ver en particular cómo resolver $\sum_{i=1}^{\log N} \frac{i}{2^i}$. Para ello, llamemos S a dicha subfórmula, o sea $S = \sum_{i=1}^{\log N} \frac{i}{2^i}$. En realidad no sabemos cuánto vale S, pero sí sabemos cómo resolver una serie geométrica con razón $\rho < 1$; es decir, sabemos que: $\sum_{i=0}^{k} \rho^i = \frac{1 - \rho^k}{1 - \rho}$.

Podemos así tratar de usar lo que conocemos para deducir el resultado de la sumatoria que nos interesa; si tomamos $\rho = \frac{1}{2}$ entonces podemos escribir a S como:

$$S = \sum_{i=1}^{\log N} \frac{i}{2^i} = \sum_{i=1}^{\log N} i \cdot \rho^i$$

por lo tanto, podemos restar a S la serie geométrica conocida[4]:

Si tomamos $k = \log N$ obtenemos:

[4]Nuevamente utilizamos el método de la sustracción para resolver una sumatoria desconocida en base a una que no lo es.

$$\left.\begin{array}{r} \rho + 2\rho^2 + 3\rho^3 + 4\rho^4 + \ldots + k\rho^k \\ - \\ 1 + \rho + \ \rho^2 + \ \rho^3 + \ \rho^4 + \ldots + \rho^k \end{array}\right\} \begin{array}{l} \longrightarrow \text{ Suma de partida} \\ \\ \longrightarrow \text{ Serie geométrica de razón } \rho \end{array}$$

$$-1 + 0 + \rho^2 + 2\rho^3 + 3\rho^4 + \ldots + (k-1)\rho^k \quad \longrightarrow \text{ Resultado de hacer la diferencia}$$

pero:

$$\rho + 2\rho^2 + 3\rho^3 + 4\rho^4 + \ldots + k\rho^k - \sum_{i=0}^{k} \rho^i = -1 + \rho \underbrace{(\rho + 2\rho^2 + 3\rho^3 + 4\rho^4 + \ldots + (k-1)\rho^{k-1})}_{S - k\rho^k}$$

entonces, podemos plantear la siguiente igualdad: $S - \sum_{i=0}^{\log N} \rho^i = S - \left(\frac{1-\rho^k}{1-\rho}\right) = -1 + \rho(S - k\rho^k)$
y de aquí podemos obtener que:

$$S = -1 + \rho S - k\rho^{k+1} + \left(\frac{1-\rho^k}{1-\rho}\right)$$

y agrupando las S obtenemos que:

$$S - \rho S = -1 - k\rho^{k+1} + \left(\frac{1-\rho^k}{1-\rho}\right)$$

y sacando factor común a S obtenemos:

$$S(1-\rho) = -1 - k\rho^{k+1} + \left(\frac{1-\rho^k}{1-\rho}\right)$$

luego, tomando común denominador $(1-\rho)$ en la expresión de la derecha se obtiene:

$$S(1-\rho) = \frac{-1(1-\rho) - k\rho^{k+1}(1-\rho) + 1 - \rho^k}{(1-\rho)} = \frac{-1 + \rho - k\rho^{k+1} + k\rho^{k+2} + 1 - \rho^k}{(1-\rho)}$$

despejando ahora S obtenemos:

$$S = \frac{-1 + \rho - k\rho^{k+1} + k\rho^{k+2} + 1 - \rho^k}{(1-\rho)^2}$$

reemplazando a ρ por $\frac{1}{2}$, a k por $\log N$ y simplificando, obtenemos:

$$
\begin{aligned}
S &= \frac{\frac{1}{2} - \log N \left(\frac{1}{2}\right)^{\log N+1} + \log N \left(\frac{1}{2}\right)^{\log N+2} - \left(\frac{1}{2}\right)^{\log N}}{\left(1-\frac{1}{2}\right)^2} \\
&= \frac{\frac{1}{2} + \frac{1}{N}\left(-\frac{\log N}{2} + \frac{\log N}{4} - 1\right)}{\left(\frac{1}{2}\right)^2} \\
&= \frac{\frac{1}{2} + \frac{1}{N}\left(-\frac{\log N}{4} - 1\right)}{\frac{1}{4}} = 2 - \frac{\log N + 4}{N}
\end{aligned}
$$

y entonces podemos finalmente decir que S es:

$$\sum_{i=1}^{\log N} i \cdot \rho^i = 2 - \frac{\log N + 4}{N}$$

Ahora, volviendo al desarrollo que quedó pendiente, teníamos:

$$N^o \textit{de punteros} = \sum_{i=1}^{\log N} i \cdot \frac{N}{2^i} = N \cdot S = N\left(2 - \frac{\log N + 4}{N}\right) = 2N - \log N - 4$$

$$N^o \textit{de punteros} = 2N - \log N - 4 \approx 2N$$

Por lo tanto, la cantidad de apuntadores necesarios para esta estructura coincide con la necesaria para almacenar los N elementos en un árbol binario de búsqueda. Por otra parte, el espacio utilizado es menor que el necesario para almacenar los N elementos en un árbol AVL, ya que éste debe almacenar en cada celda, además de los dos apuntadores, el factor de balance del nodo.

Llamemos a los punteros de la lista original los punteros de *nivel 0*, y en general a los punteros que saltan hacia adelante 2^i elementos los punteros de *nivel i*. Una celda de cabecera tendría un campo para X (no usado), y tendría los punteros iniciales de las listas de todos los niveles.

De acuerdo a esta representación, los punteros de alto nivel serían usados durante una localización para saltar rápidamente a través de grandes segmentos de la lista; sólo cuando tuviéranos un puntero a un elemento con valor de X mayor que el x buscado, seguiríamos con un puntero de menor nivel. El algoritmo de localización sería a grandes rasgos el siguiente:

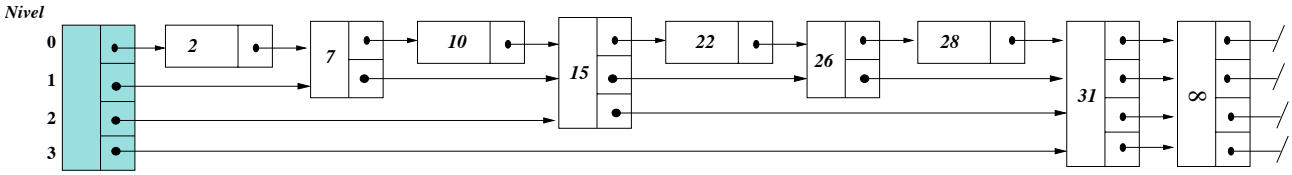
> *Comenzando con los punteros del mayor nivel, seguirlos hasta que se encuentre un elemento con valor de X mayor o igual que el x buscado. Si en dicha celda el valor de X encontrado es igual al x buscado, paramos y la búsqueda tuvo* ***éxito****; si en otro caso es mayor que el x buscado, volvemos hacia atrás una celda (desandamos un puntero) y continuamos siguiendo los punteros del siguiente nivel más bajo. Repetimos estos pasos*

*hasta alcanzar el nivel 0 y si un puntero de nivel 0 guía a un elemento con valor igual al buscado se tuvo éxito, si en cambio encuentra un valor de X mayor que el x buscado, paramos y la búsqueda **fracasó**.*

El volverse hacia atrás no requiere verdaderamente invertir la dirección del puntero, sino simplemente mantener la traza de la celda del último puntero seguido. Para que este algoritmo sea completamente correcto, se coloca una marca de fin de lista con un valor de X que exceda cualquier valor real y posible para X, al que denotamos con "∞" ($+\infty$).

Si la lista fuera perfectamente organizada, como se muestra en la siguiente figura, cualquier elemento puede ser encontrado (o descubierto como no presente) en $O(\log N)$ pasos. Esto es porque existen $\log N$ niveles y podemos seguir sólo dos punteros por nivel antes de bajar a un nivel menor. De hecho, el modelo de acceso sería como el de la búsqueda binaria.

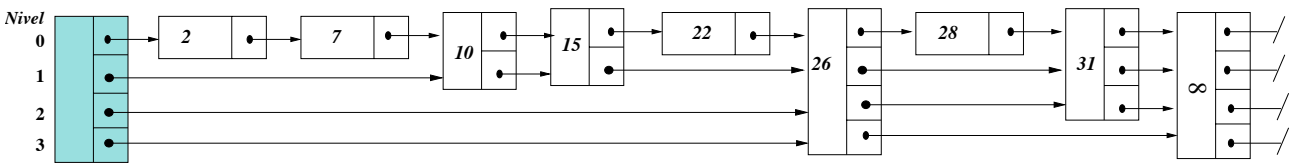
Sea una celda con $i+1$ punteros, conteniendo punteros de niveles $0, 1, \ldots, i$, una celda de *nivel i* en la estructura.



Una lista con la estructura como la de la figura tendría un buen comportamiento para la localización, pero sería bastante impráctica si el conjunto fuese dinámico, ya que insertar un único item en la lista podría forzar a que se reestructure completamente la lista. Por ejemplo, si insertamos un elemento con valor de X igual a 1 en la lista de la figura anterior, causaría que cada celda que ya estaba en la lista cambie de nivel. De esta manera, se perdería la gran ventaja de las listas vinculadas sobre las listas secuenciales respecto del bajo costo de la modificación estructural; pero esto es por intentar mantener siempre "perfecta" la lista (o sea manteniendo realmente los elementos en posiciones múltiplos de i vinculados por punteros de *nivel* $i+1$).

Lo mismo ocurriría frente a la eliminación de un elemento en la lista dado que, por ejemplo, si eliminamos el primer elemento de la lista se debería reestructurar completamente la lista, para mantenerla tal como la habíamos planteado.

Sin embargo, si permitimos relajar esta condición, el comportamiento *esperado* de la estructura sería a grandes rasgos el mismo. En lugar de tener alternación perfecta de niveles (como en la figura anterior), mantendremos el mismo modelo general, con las celdas de varios niveles presentes en similares proporciones que antes, pero distribuídas aleatoriamente a través de la lista, como en la siguiente figura.



El proceso general que sigue la localización sería el mismo que el enunciado anteriormente, aunque flexibilicemos la alternación perfecta de niveles en la estructura.

Las celdas de niveles más altos son relativamente poco frecuentes, y por lo tanto cadenas de niveles más altos nos permiten realizar una localización que salta rápidamente hacia abajo en la lista. Si ahora no requerimos que se mantenga una estructura perfecta, y sólo necesitamos asegurar que las celdas de los distintos niveles existan en la proporción correcta y que sean distribuidas uniformemente a través de la lista, el problema de la inserción es mucho más simple.

Analicemos entonces cómo sería la inserción. Para insertar un nuevo elemento, primero encontramos su posición correcta en la lista, y *generamos aleatoriamente su nivel*, sujeto a la condición que para cada i el nivel sería al menos el doble de probable de que fuera $i+1$ o mayor.[5]

En general, se plantea que la generación aleatoria de los niveles para las celdas respete la siguiente distribución de probabilidades:

Nivel	Probabilidad
0	$\frac{1}{2}$
1	$\frac{1}{4}$
2	$\frac{1}{8}$
$\vdots$	$\vdots$
i	$\left(\frac{1}{2}\right)^{i+1}$

Es posible, aunque poco probable, que una larga secuencia de inserciones se produjeran en el nivel 0, en cuyo caso la estructura no sería parecida a la mostrada en las figuras anteriores. En ese caso la estructura más bien parecería una lista vinculada ordenada común, con sus pobres características frente a la localización de elementos. Pero, esta situación es realmente poco probable.

Por ejemplo, la probabilidad de que una secuencia de 20 inserciones ocurrieran todas a nivel 0 es sólo $\left(\frac{1}{2}\right)^{20}$, o menos que uno en un millón, dado que la probabilidad de tener nivel 0 es de $\frac{1}{2}$ (porque ya vimos que la mitad de los elementos eran de nivel 0) y asumiendo independencia de nivel en sucesivas inserciones se obtiene por probabilidad conjunta de eventos independientes el valor de $\left(\frac{1}{2}\right)^{20} < \frac{1}{1000000}$.

Más aún, estas probabilidades no dependen de ninguna manera de los valores de X de los elementos; no existen "malas" secuencias de elementos para este algoritmo sino sólo "malas" secuencias de salidas desde el generador de números aleatorios. Si los mismos elementos fuesen insertados en una

[5] La mayoría de los lenguajes de programación tienen un generador de números aleatorios que pueden hacer de esto un cálculo sencillo.

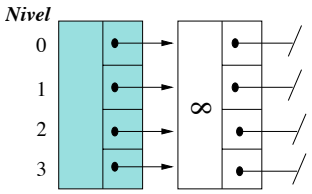
nueva estructura, utilizando una nueva secuencia de salidas desde el generador de números aleatorios, la probabilidad de que las 20 inserciones ocurran todas a nivel 0 sería también de $\left(\frac{1}{2}\right)^{20}$.

Para dar más detalles sobre los algoritmos para estas *skip lists*, necesitamos fijar el nivel máximo para las celdas, porque no deberían existir celdas teniendo un nivel tan alto que la lista completa sea saltada de una sola vez.

El nivel máximo útil es cerca de $\log N - 1$, dado que sólo un nodo se espera que tenga un nivel mayor que éste [6]. Por conveniencia, tomaríamos el *máximo nivel* como $\lfloor \log N \rfloor - 1$, donde N es la cantidad máxima de elementos anticipada para la estructura de datos. Por ejemplo, si $N = 10000$, entonces el máximo nivel será 12, correspondiendo a una lista en la cual las celdas tienen 13 niveles posibles.

La *skip list* en sí misma se puede implementar como una estructura de registro de dos campos: *Cabecera*, la cual sería una celda ficticia para comenzar las listas, y *Nivel*, el cual sería un entero que daría el nivel más grande de cualquier celda presente actualmente en la lista. Una celda tendría, además de los campos para almacenar X e Y, una tabla de apuntadores a las siguientes celdas de acuerdo a las listas en las que dicha celda participa. El tamaño de esta tabla dependería del nivel de la celda, pero dicho número no necesitaría ser almacenado en la celda. Inicialmente el *Nivel* de una *skip list* sería 0 y todos los apuntadores apuntarían a la celda ficticia final cuyo valor almacenado en el campo X sería, como ya habíamos expresado, el valor considerado "∞"para cualquier valor posible de X.

La siguiente figura muestra la situación inicial de una *skip list* con *máximo nivel* = 3:



Para insertar un elemento en una *skip list* necesitaríamos primero tener una rutina que genere los números aleatorios apropiados del nivel. En general, se puede contar con una función *Random()* disponible en la mayoría de los lenguajes de programación, la cual devuelve un número aleatorio k en el rango $0 \leq k < 1$ con distribución uniforme. Entonces deberíamos hacer, por ejemplo, lo siguiente para obtener un nivel aleatorio n entre 0 y *Máximo nivel* con la distribución deseada:

```
n ← 0
Mientras Random() < 1/2 y n < Máximo Nivel hacer n ← n + 1
Retornar(n)
```

de esta manera, se puede generar un nivel aleatorio n en el rango $0, \ldots,$*Máximo nivel* con probabilidades que decrecen exponencialmente; porque con la misma probabilidad de $\frac{1}{2}$ se devolvería un nivel 0 que uno de todos los mayores a 0 (o sea, de nivel 1 hasta *Máximo nivel*)[7]. A su vez con $\frac{1}{4}$ de probabilidad

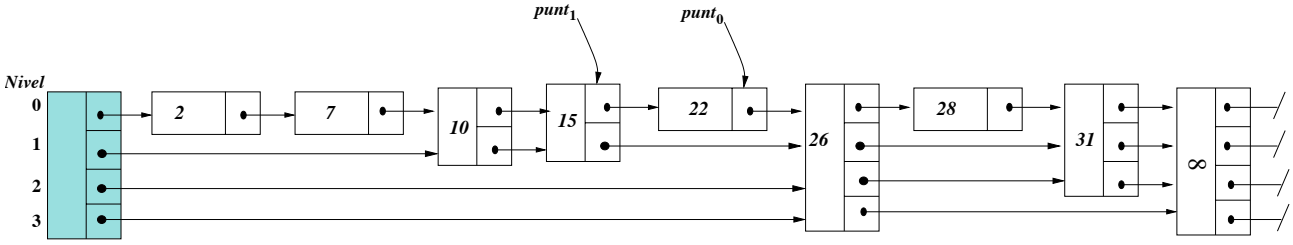
[6] El nivel máximo útil sería de $\log N$ si empezamos a enumerar los niveles desde 1 en lugar que desde 0.

[7] Si la función *Random()* devuelve con igual probabilidad cualquier valor k, tal que $0 \leq k < 1$; y como la mitad de los valores en ese rango son mayores o iguales que 0 y menores que $\frac{1}{2}$ (e igualmente la mitad de los valores en ese rango son

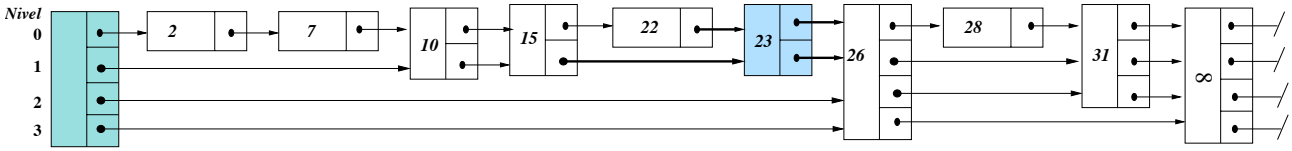
se devolvería un nivel 1 y así siguiendo. Es decir, en cada paso que se aumenta en uno el valor de n, que se devolverá como nivel aleatorio de la celda, dicho incremento se realiza con probabilidad $\frac{1}{2}$.

Para insertar un nuevo elemento con un nuevo valor para X, comenzamos usando el algoritmo de localización para encontrar su posición apropiada. Durante el proceso de búsqueda se mantiene una cantidad de *Máximo nivel* +1 apuntadores auxiliares; $punt_i$ mantendrá un puntero a la celda de más a la derecha de nivel i o superior que es anterior a la posición de la inserción. Cuando la posición ha sido ubicada, se crea una celda de un nivel generado aleatoriamente n y se ubica en las listas vinculándola en todas aquellas listas de hasta el nivel n.

Por ejemplo si deseamos insertar un nuevo elemento con valor 23, la localización fracasaría y nos dejaría la siguiente información importante sobre los punteros:



si el nivel generado aleatoriamente fuese de $n = 1$, crearíamos una nueva celda con dos apuntadores a la que deberíamos incorporar, de acuerdo al orden, en las listas de niveles 0 y 1; produciendo luego de la modificación estructural la siguiente estructura, donde se han remarcado los apuntadores que se modifican o agregan y la nueva celda incorporada.

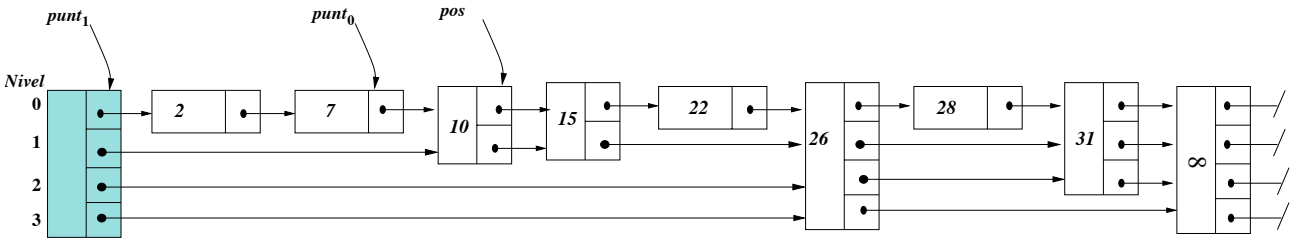


La eliminación en una *skip list* es bastante similar a la inserción, primero se debe localizar al elemento manteniendo, como ya habíamos mencionado *Máximo nivel* +1 apuntadores, donde $punt_i$ mantendrá un puntero a la celda de más a la derecha de nivel i o superior que es anterior a la posición de la celda que contiene el x buscado. Pero, para poder utilizar la misma rutina de *Localización* para todas las otras rutinas, es necesario que la localización exitosa llegue indefectiblemente hasta el nivel 0 y no se detenga en niveles mayores para así poder registrar, en los punteros auxiliares, las direcciones de todas las celdas que son factibles de modificación luego de la baja.

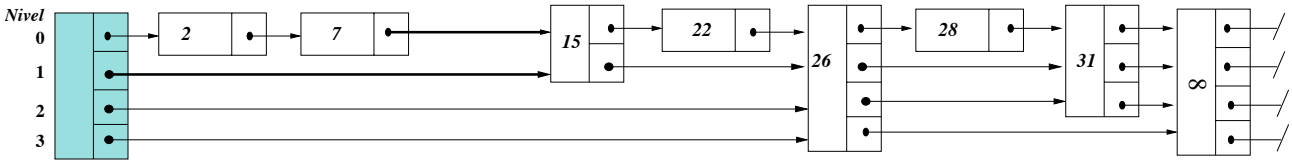
Por lo tanto, luego de que la localización tenga éxito y conociendo las direcciones de las celdas anteriores a la celda que se quiere dar de baja, en las listas de cada uno de los niveles, se procede a actualizar sólo los apuntadores de los niveles hasta el nivel de la celda a dar de baja.

mayores o iguales que $\frac{1}{2}$ y menores que 1); por lo tanto la probabilidad de que el valor sea menor que $\frac{1}{2}$ es realmente $\frac{1}{2}$ (igual que la probabilidad de que sea mayor o igual que $\frac{1}{2}$).

Por ejemplo, si se desea eliminar el elemento 10 de la lista anterior, la localización exitosa nos brindaría la siguiente información útil [8]:



y luego de la modificación estructural la estructura quedaría como:



Los apuntadores que aparecen remarcados son los únicos que se modifican para reflejar la baja de la celda de nivel 1 conteniendo el valor 10. Por ser la celda de nivel 1 sólo se modifican los apuntadores hasta dicho nivel, en este caso para reflejar la baja en las listas de nivel 0 y de nivel 1.

Lo que resta por analizar son los esfuerzos medios para las operaciones de *Localización* [9], *Altas* y *Bajas*.

Vamos a hacer un análisis simplificado para mostrar que los esfuerzos medios para dichas operaciones son de orden $O(\log N)$, siendo N la cantidad máxima esperada de elementos a almacenar en la estructura.

Lo único que es más dificil de establecer es que el tiempo esperado para la localización de un elemento en una *skip list* es logarítmico; que las restantes operaciones toman también tiempo esperado de orden logarítmico se puede deducir desde este hecho. Esto es debido a que la modificación estructural, en el peor de los casos, es también logarítmica por tener que modificar del orden de $O(\log N)$ punteros (por ser *Máximo nivel* $\approx \lfloor \log N \rfloor - 1$).

Una localización comienza en el puntero de *Nivel* (que es el máximo nivel corriente)[10] en la cabecera y procede en general de acuerdo a pasos de dos clases: siguiendo punteros dentro de un nivel y bajando dentro de un nodo desde un nivel al próximo nivel más bajo. Cuando se sigue un puntero, su nivel es siempre igual al nivel del nodo al cual éste apunta. Analizaremos la longitud esperada de tal paso de búsqueda, contando o los seguimientos de punteros o las bajadas de nivel como un paso de costo 1.

Para analizar la longitud de paso esperada para alcanzar un elemento de nivel 0 en la lista, trazamos el paso hacia atrás. En general, preguntamos: suponiendo que estamos trazando un paso hacia atrás desde

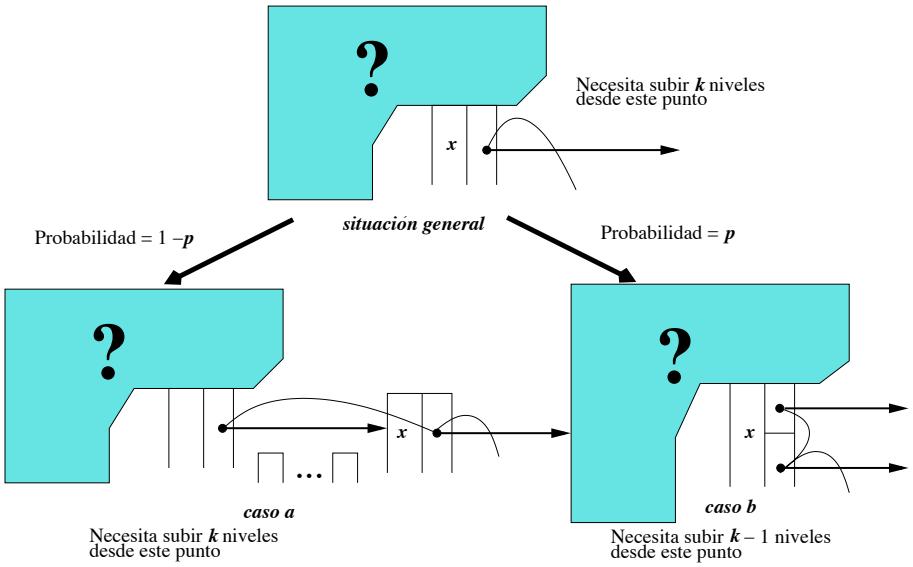
[8] Luego de la localización también tendrían direcciones los punteros auxiliares de 2^o y 3^o nivel, pero como la celda a eliminar es de nivel 1 no es necesario utilizarlos.

[9] No vale la pena obtener el esfuerzo máximo para *Evocación Asociativa* dado que será el mismo que el de la *Localización*.

[10] En la celda de cabecera teníamos un campo *Nivel* que mantenía el máximo nivel alcanzado en la *skip list* y era menor o igual que *Máximo nivel*.

un puntero de nivel i en una celda P; ¿cuánto debería esperar el paso para continuar antes de que crezca k niveles? Si nos enfocamos sobre una celda P al final del paso, existen dos posibilidades, las cuales se muestran en la figura.

1. P es una celda de nivel i y el paso precede hacia atrás un puntero a una celda de nivel al menos i, desde la cual todavía pueden crecer k niveles, o sino
2. el paso crece un nivel en el nodo P y debe aún crecer $k-1$ niveles más a medida que se proceda hacia atrás desde P.



Cada uno de los casos ***a*** y ***b*** tiene probabilidad $\frac{1}{2}$, ya que en nuestro caso consideramos que $p = \frac{1}{2}$ y por lo tanto $1 - p = \frac{1}{2}$ también. Por lo tanto, si $C(k)$ es la longitud esperada de un paso que crezca k niveles sobre su trayectoria hacia atrás, obtenemos la recurrencia:

$$\begin{aligned} C(k) &= \frac{1}{2} \cdot (((\text{costo de ir atrás un puntero en nivel } i) + C(k)) + \\ &\quad \frac{1}{2}(((\text{costo de ir hacia arriba desde el nivel } i \text{ al } i+1) + C(k-1)) \\ &= \frac{1}{2}((1 + C(k)) + \frac{1}{2}((1 + C(k-1)) \end{aligned}$$

y por lo tanto,

$$C(k) = 2 + C(k-1),$$

lo cual guía a la solución $C(k) = 2k$ ya que $C(0)$, la longitud de paso necesaria para aumentar 0 niveles, es 0. Este análisis es realmente un poco pesimista, ya que asume que el paso no alcanza la celda

cabecera en el curso de aumentar k niveles; una vez que se encuentra la celda cabecera el paso sólo puede aumentar, no proceder más a la izquierda (es decir hacia las celdas con valores de X menores).

Ahora aplicamos este análisis para determinar la longitud de un paso desde una celda de nivel 0 todo el camino hacia atrás hasta el apuntador de nivel *Nivel* en la celda de cabecera. Para obtener desde una celda de nivel 0 hacia atrás una celda de nivel $\log N - 1$ se espera que tome $2(\log N - 1)$ pasos. Si el paso no termina en la celda cabecera, necesitamos seguir un paso a través de los apuntadores de niveles $\log N - 1$ y más altos hacia atrás a la celda cabecera, pero el número esperado de pasos hacia la izquierda en este caso no es mayor que el número de celdas de nivel $\log N - 1$ o mayores esperadas en la lista completa. Dado que la probabilidad que una celda tenga nivel i o más alto es $\frac{1}{2^i}$, entre N elementos el número esperado de celdas de nivel $\log N - 1$ o mayor es:

$$N \cdot \left(\frac{1}{2}\right)^{\log N - 1} = N \cdot \frac{1}{N} \cdot 2 = 2$$

si $Nivel > \log N - 1$ el paso debe también crecer $Nivel - (\log N - 1)$ pasos, pero el valor esperado de *Nivel* es a lo más $\log N + 1$, así el crecimiento esperado desde el nivel $\log N - 1$ es a lo sumo 2. Por lo tanto, la longitud de paso total esperada es:

$$2(\log N - 1) + 2 + 2 = 2 \log N + 2 \in O(\log N)$$

y así las operaciones de *Localización*, *Alta* y *Baja*, las cuales toman tiempo lineal en la longitud de paso atravesado, tiene tiempo esperado de $O(\log N)$.

□

En el otro sentido, no existe garantía sobre el comportamiento de peor caso de los algoritmos de las *skip lists*, excepto que no es mucho peor que el peor caso de los correspondientes algoritmos sobre listas vinculadas ordenadas. Pero, el peor caso es muy poco probable, y todavía más importante está fuera de control de quien provee los datos a ser almacenados. El peor caso depende sólo del comportamiento del generador de números aleatorios dentro de los algoritmos de *skip list*; los algoritmos serían inmunes aún frente al ataque de un adversario malicioso quien conozca bien el código del programa y provea requerimientos de secuencias de inserciones y eliminaciones especialmente elegidas en un intento por forzar que ocurra el comportamiento del peor caso.

Claramente, por ser *aleatoria*, esta estructura de datos es distinta al resto de las estructuras de datos vistas. En las restantes estructuras de datos cada vez que se intente generar una estructura a partir de una secuencia dada, la estructura resultante tendrá la misma "estructura interna", pero la *skip list* generada en un momento dado con una determinada secuencia de entrada puede ser muy diferente a la generada con la misma secuencia en otro momento.

Aunque las *skip lists* son relativamente recientes, la evidencia experimental sugiere que su comportamiento es competitivo con respecto a las estructuras de datos más sofisticadas de árboles balanceados y mucho más fáciles de programar.

Un planteo más general de las *skip lists* se puede hacer considerando que en el nivel 1 queremos saltar t elementos, y entonces una de cada t celdas debe tener 2 vínculos, en el nivel 2 saltamos t^2 elementos, y así una de cada t^2 celdas tendrán 3 vínculos, en conclusión queremos que una de cada t^i celdas tengan al menos $i + 1$ vínculos. En ese caso para la *skip list* serían necesarios en promedio

$\left(\frac{t}{t-1}\right) N$ vínculos, porque tendríamos N vínculos en el nivel 0, $\frac{N}{t}$ vínculos en el nivel 1, $\frac{N}{t^2}$ vínculos en el nivel 2 y así siguiendo, para un total en la lista completa de:

$$N\left(1+\frac{1}{t}+\frac{1}{t^2}+\frac{1}{t^3}+\ldots\right)=\frac{N}{\left(1-\frac{1}{t}\right)}=\left(\frac{t}{t-1}\right)\cdot N$$

Una localización en una *skip list* con parámetro t requiere, en promedio, aproximadamente $\frac{t\log_t N}{2}=\left(\frac{t}{2}\right)\cdot\log_t N$ celdas consultadas.

Por lo tanto, en este caso hay que tener en cuenta el compromiso espacio-tiempo, tomando un valor de t apropiado: *si aumentamos t aumenta el esfuerzo de localización y disminuye el espacio utilizado para vínculos*. Por ejemplo, con $t=2$ una *skip list* necesita en promedio aproximadamente $\log N$ celdas consultadas en una localización y usa $2N$ vínculos.

Bibliografía de Referencia

1. William Pugh: *Skip Lists: A Probabilistic Alternative to Balanced Trees*. Communications of the ACM, 33, 1990, pág. 668-676.
2. Harry Lewis y Larry Denenberg: *Data Structures and Their Algorithms*, Capítulo 6. Harper Collins Publishers, 1991.
3. Mark Weiss: *Data Structures and Algorithms Analysis in C*, Capítulo 10. The Benjamin/Cummings Publishing Company, 1993.
4. Robert Sedgewick: *Algorithms in C: Parts 1- 4*, Capítulo 13. Addison-Wesley, 1998.
5. Erik Demaine: *Lecture 12: Skip Lists*, disponible en: `http://videolectures.net/mit6046jf05_demaine_lec12/`.