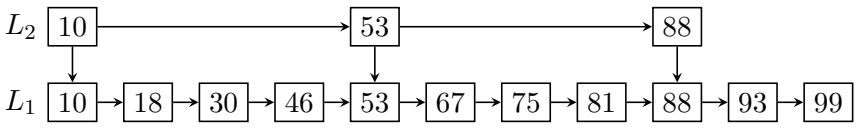
Data Structures and Algorithms
Lecture Outline
April 21, 2016

Skip Lists

Skip lists (due to Bill Pugh in 1990) is a randomized data structure that can be thought of as a generalization of sorted linked lists. They have most of the desirable properties of balanced binary search trees and the simplicity of linked lists. Skip lists support the standard operations for managing a dynamic data set – INSERT, DELETE, and SEARCH.

In a sorted linked list with n elements, searching an element takes $\Theta(n)$ time. What if we had two-level sorted linked lists structure as shown in the figure below? The bottom list L_1 contains all the elements and another list L_2 contains only a subset of elements of L_1.



Note that to minimize the worst case search time, the nodes in L_2 should be evenly distributed, i.e., the number of nodes in L_1 that are "skipped" between any two consecutive nodes in L_2 should be the same. The search time in such a 2-level structure is given by

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

To minimize the search time, the two terms in the above expression must be equal. Thus solving

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

we get $|L_2| = \sqrt{n}$ and hence the total search time is at most $2\sqrt{n}$. Generalizing it a step further, consider a 3-level structure – list L_1 containing all the elements, list L_2 containing elements that are evenly distributed over elements in L_1, and list L_3 containing elements that are evenly distributed over elements in L_2. The search time in a 3-level structure is given by

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

The above expression is minimized when the three terms are equal. If we solve the following two equations:

$$|L_3| = \frac{|L_2|}{|L_3|} \qquad \text{and} \qquad \frac{|L_2|}{|L_3|} = \frac{n}{|L_2|},$$

we get that $|L_3| = \sqrt[3]{n}$ and hence the search time is at most $3\sqrt[3]{n}$. Generalizing this to a k-level structure, we get the total search time to be at most $k\sqrt[k]{n}$. Setting $k = \lg n$, we get the search time to be at most

$$\lg n(n^{\frac{1}{\lg n}}) = \lg n(2^{\lg n})^{\frac{1}{\lg n}} = 2\lg n$$

Thus in a skip list with $\lg n$ levels, the bottom list L_1 contains all of the n elements, L_2 contains $n/2$ elements (every other element of L_1), L_3 contains $n/4$ elements, and so on. Note that this structure is like a perfectly balanced binary search tree. The insertion or deletion of a node could disrupt this structure, however just as in AVL trees, we do not need a perfectly balanced structure – a little imbalance still allows for fast search time. In skip lists the balancing is done using randomization – for each element in L_i we toss a fair coin to decide whether the element should also belong to L_{i+1} or not. In expectation, we would expect half the elements to also be part of L_{i+1}. Note that the randomization in constructing this data structure does not arise because of any assumption on the distribution of the keys. The randomization only depends on a random number generator. Thus an adversary cannot pick a "bad" input for this data structure, i.e., a sequence of keys that will result in poor performance of this data structure.

The three operations SEARCH, INSERT, and DELETE can be implemented as follows. Let ℓ be the largest index of a linked list. We add a sentinel nodes $-\infty$ to each list.

```
SEARCH(x):
v_ℓ ← element with key −∞ in L_ℓ.
for i ← ℓ down to 1 do
   Follow the down-link from v_i to v_{i−1}.
   Follow the right-links starting from v_{i−1} until the key of an element is larger than x.
   v_{i−1} ← largest element in L_{i−1} that is at most x.
return v_1
```

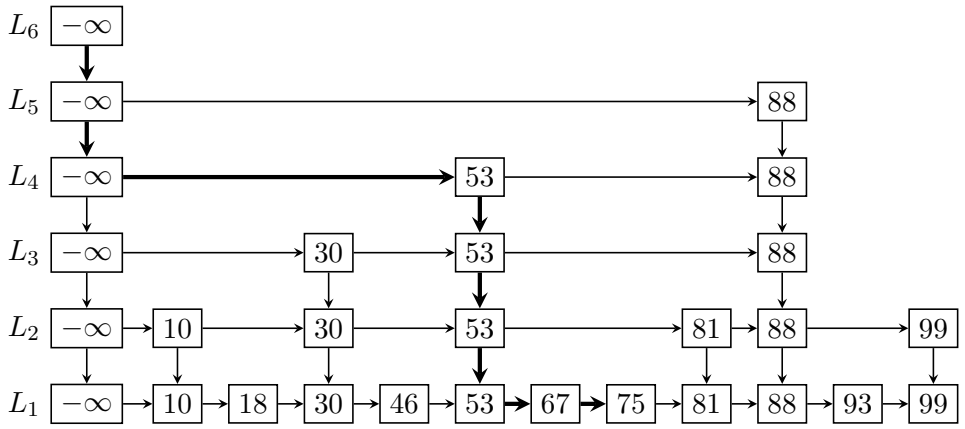
The pseudo-code for the INSERT operation is given below.

```
INSERT(x):
if SEARCH(x) = x then
   return
(v_ℓ, v_{ℓ−1}, ..., v_1) ← elements in L_ℓ, L_{ℓ−1}, ..., L_1 with key values at most x
Flip a fair coin until we get tails. Let f be the number of coin flips.
for i ← 1 to min(ℓ, f) do
   Insert x in L_i
if f > ℓ then
   Create lists L_{ℓ+1}, L_{ℓ+2}, ..., L_f.
   Add elements {−∞, x} to L_ℓ, L_{ℓ+1}, ..., L_f.
   Create L_{f+1} containing −∞.
ℓ ← max(ℓ, f + 1)
```

The pseudo-code for the DELETE operation is as follows.

```
DELETE(x):
if SEARCH(x) ≠ x then
  return
(v_ℓ, v_{ℓ−1}, ..., v_1) ← elements in L_ℓ, L_{ℓ−1}, ..., L_1 with key values at most x
for i ← 1 to ℓ do
  if v_i = x then
      Delete x in L_i
while ℓ ≥ 2 do
  if L_ℓ contains only the element −∞ then
      ℓ ← ℓ − 1
```

An example of a skip list on the set $\{30, 81, 10, 46, 53, 75, 99, 88, 93, 67, 18\}$ is as shown in the figure below. The search path for element with key 75 is highlighted.



Analysis

In the analysis below we will show that some events happen with a high probability (w.h.p). This means that the probability with which the event occurs is at least $1 - \frac{1}{n^c}$, for some constant $c \geq 1$.

Let $x_1, x_2, \ldots, x_n$ be the elements inserted into the skip list. Let H_i denote the number of lists that element x_i belongs to. Note that H_i is exactly equal to the total number of flips of a fair coin until the first "success". This is a geometric random variable with parameter $1/2$ and hence $\mathbf{E}[H_i] = 2$. Let $H = \max\{H_1, H_2, \ldots, H_n\}$.

Theorem. The space requirement for a skip list with n elements is $O(n)$ in expectation.

Proof. Let S be the random variable denoting the total amount of space required by a skip list with n elements. Taking into account the space overhead of the sentinel keys

$(-\infty)$, we have

$$\begin{aligned} S &= O\left(\sum_{i=1}^{n} H_i\right) \\ \mathbf{E}[S] &= O\left(\sum_{i=1}^{n} \mathbf{E}[H_i]\right) \\ &= O(n) \end{aligned}$$

Lemma 1 $\Pr[H \geq h] \leq \frac{n}{2^{h-1}}$

Proof We have

$$\begin{aligned} \Pr[H \geq h] &= \Pr[\bigcup_{i=1}^{n} H_i \geq h] \\ &\leq \sum_{i=1}^{n} \Pr[H_i \geq h] \qquad \text{(using the union-bound)} \\ &= \frac{n}{2^{h-1}} \end{aligned}$$

Lemma 2 The expected number of levels in a skip list with n elements is $\lceil \lg n \rceil + 3$.

Proof The number of levels in a skiplist of n elements is $H + 1$. The second term is due to the list with the highest index that contains only the key $-\infty$. Since H is a non-negative random variable, we have

$$\begin{aligned} \mathbf{E}[H] &= \sum_{h=1}^{\infty} \Pr[H \geq h] \\ &= \sum_{h=1}^{\lceil \lg n \rceil} \Pr[H \geq h] + \sum_{h \geq \lceil \lg n \rceil + 1} \Pr[H \geq h] \\ &= \sum_{h=1}^{\lceil \lg n \rceil} 1 + \sum_{h \geq \lceil \lg n \rceil + 1} \frac{n}{2^{h-1}} \\ &\leq \lceil \lg n \rceil + \sum_{h \geq \lg n + 1} \frac{n}{2^{h-1}} \\ &= \lceil \lg n \rceil + \frac{n}{2^{\lg n}} \sum_{i=0}^{\infty} \frac{1}{2^i} \\ &= \lceil \lg n \rceil + 2 \end{aligned}$$

Thus the expected number of levels is $\mathbf{E}[H] + 1 = \lceil \lg n \rceil + 3$.

Lemma 3 With a high probability, a skip list with n elements has $O(\log n)$ levels.

Proof. We will show that w.h.p, the number of levels in a skip list with n elements is at most $2\lg n + 1$. This is equivalent to showing that w.h.p, $H \leq 2\lg n$. From Lemma 1, we have

$$\begin{aligned}\Pr[H \geq 2\lg n + 1] &\leq \frac{n}{2^{2\lg n+1-1}} \\ &= \frac{1}{n}\end{aligned}$$

Lemma 4 For any r such that $0 < r \leq n$,

$$\sum_{i=0}^{r} \binom{n}{i} = \left(\frac{ne}{r}\right)^r$$

Proof. Rewriting the left hand side in a more conveninent form we have

$$\begin{aligned}\sum_{i=0}^{r} \binom{n}{i} &= \left(\frac{n}{r}\right)^r \sum_{i=0}^{r} \binom{n}{i} \left(\frac{r}{n}\right)^r \\ &\leq \left(\frac{n}{r}\right)^r \sum_{i=0}^{r} \binom{n}{i} \left(\frac{r}{n}\right)^i \\ &\leq \left(\frac{n}{r}\right)^r \sum_{i=0}^{n} \binom{n}{i} \left(\frac{r}{n}\right)^i \\ &= \left(\frac{n}{r}\right)^r \left(1 + \frac{r}{n}\right)^n && \text{(using the Binomial Theorem)} \\ &\leq \left(\frac{n}{r}\right)^r e^{(\frac{r}{n})n} && \text{(using} 1 + x \leq e^x, \text{for all } x) \\ &= \left(\frac{ne}{r}\right)^r\end{aligned}$$

Theorem. With a high probability, in a skip list of n elements, every search takes $O(\log n)$ time.

Proof. When searching an element with key x, we begin at the first element in the top list and then move down to the next lower indexed list or move right along the same list until we reach the element v_1 (the element with the largest key in L_1 that is at most x). We want to bound the total number of steps ("down" steps plus "right" steps) with a high probability. Instead of counting the total number of down steps plus the total number of right steps, we will start at the element v_1 and trace the search path in reverse, i.e., either go "left" or go "up". Note that we go left at key x_j in list L_i because x_j was not promoted to the next higher list L_{i+1}, i.e., the ith flip of the coin must have resulted in tails when inserting x_j in the skip list. From Lemma 3, we know that w.h.p., the number of up moves are at most $2\lg n + 1$. Thus we want to calculate the total number of coin flips that will result in $2\lg n + 1$ heads w.h.p.. Consider $20\lg n$ flips of a fair coin. Let X be the number

of heads. We will bound the probability of the bad event, i.e., $\Pr[X \leq 2\lg n]$ as follows.

$$\begin{aligned}
\Pr[X \leq 2\lg n] &= \sum_{i=0}^{2\lg n} \Pr[X = i] \\
&= \sum_{i=0}^{2\lg n} \binom{20\lg n}{i} \left(\frac{1}{2}\right)^{i} \left(\frac{1}{2}\right)^{20\lg n - i} \\
&= \left(\frac{1}{2}\right)^{20\lg n} \sum_{i=0}^{2\lg n} \binom{20\lg n}{i} \\
&\leq \left(\frac{1}{2}\right)^{20\lg n} \left(\frac{e \cdot 20\lg n}{2\lg n}\right)^{2\lg n} && \text{(using Lemma 4)} \\
&= \left(\frac{1}{2}\right)^{20\lg n} (10e)^{2\lg n} \\
&= 2^{\lg(10e)\cdot 2\lg n} \cdot 2^{-20\lg n} \\
&= 2^{2\lg n(\lg(10e)-10)} \\
&\leq \frac{1}{2^{2\lg n}} \\
&= \frac{1}{n^2}
\end{aligned}$$