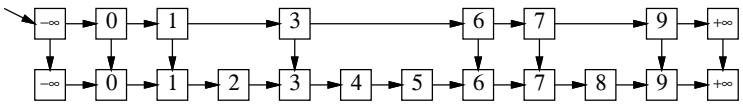
> *For example, creating shortcuts by sprinkling a few diversely connected individuals throughout a large organization could dramatically speed up information flow between departments. On the other hand, because only a few random shortcuts are necessary to make the world small, subtle changes to networks have alarming consequences for the rapid spread of computer viruses, pernicious rumors, and infectious diseases.*
>
> — Ivars Peterson, *Science News*, August 22, 1998.

A Skip Lists

This lecture is about a probabilistic data structure called *skip lists*, first discovered by Bill Pugh in the late 1980's.[1] Skip lists have many of the desirable properties of balanced binary search trees, but their structure is completely different.

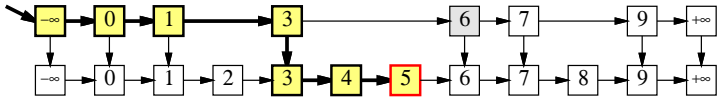
A.1 Shortcuts

At a high level, a skip list is just a sorted, singly linked list with some shortcuts. To do a search in a normal singly-linked list of length n, we obviously need to look at n items in the worst case. To speed up this process, we can make a second-level list that contains roughly half the items from the original list. Specifically, for each item in the original list, we duplicate it with probability $1/2$. We then string together all the duplicates into a second sorted linked list, and add a pointer from each duplicate back to its original. Just to be safe, we also add sentinel nodes at the beginning and end of both lists.



A linked list with some randomly-chosen shortcuts.

Now we can find a value x in this augmented structure using a two-stage algorithm. First, we scan for x in the shortcut list, starting at the $-\infty$ sentinel node. If we find x, we're done. Otherwise, we reach some value bigger than x and we know that x is not in the shortcut list. Let w be the largest item less than x in the shortcut list. In the second phase, we scan for x in the original list, starting from w. Again, if we reach a value bigger than x, we know that x is not in the data structure.



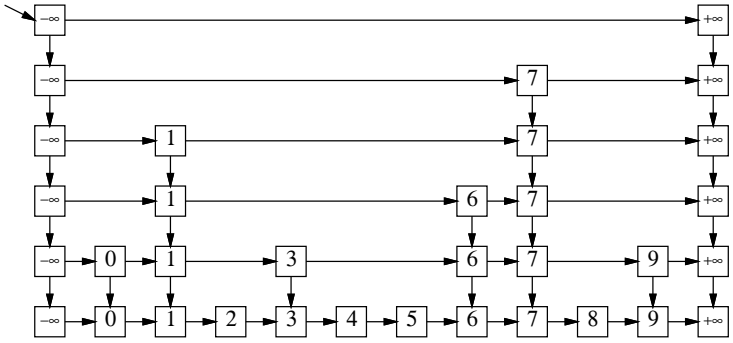
Searching for 5 in a list with shortcuts.

Since each node appears in the shortcut list with probability $1/2$, the expected number of nodes examined in the first phase is at most $n/2$. Only one of the nodes examined in the second phase has a duplicate. The probability that any node is followed by k nodes without duplicates is 2^{-k}, so the expected number of nodes examined in the second phase is at most $1 + \sum_{k \geq 0} 2^{-k} = 2$. Thus, by adding these random shortcuts, we've reduced the cost of a search from n to $n/2 + 2$, roughly a factor of two in savings.

[1] William Pugh. Skip lists: A probabilistic alternative to balanced trees. *Commun. ACM* 33(6):668–676, 1990.

A.2 Skip lists

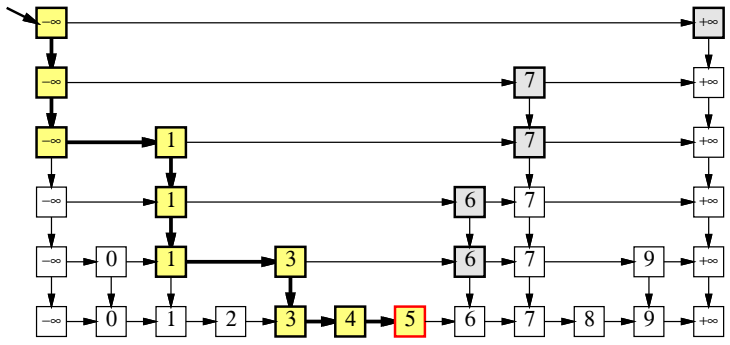
Now there's an obvious improvement—add shortcuts to the shortcuts, and repeat recursively. That's exactly how skip lists are constructed. For each node in the original list, we flip a coin over and over until we get tails. For each heads, we make a duplicate of the node. The duplicates are stacked up in levels, and the nodes on each level are strung together into sorted linked lists. Each node v stores a search key ($\text{key}(v)$), a pointer to its next lower copy ($\text{down}(v)$), and a pointer to the next node in its level ($\text{right}(v)$).



A skip list is a linked list with recursive random shortcuts.

The search algorithm for skip lists is very simple. Starting at the leftmost node L in the highest level, we scan through each level as far as we can without passing the target value x, and then proceed down to the next level. The search ends when we either reach a node with search key x or fail to find x on the lowest level.

```
SkipListFind(x, L):
  v ← L
  while (v ≠ Null and key(v) ≠ x)
      if key(right(v)) > x
          v ← down(v)
      else
          v ← right(v)
  return v
```



Searching for 5 in a skip list.

Intuitively, Since each level of the skip lists has about half the number of nodes as the previous level, the total number of levels should be about $O(\log n)$. Similarly, each time we add another level of random shortcuts to the skip list, we cut the search time in half except for a constant overhead. So after $O(\log n)$ levels, we should have a search time of $O(\log n)$. Let's formalize each of these two intuitive observations.

A.3 Number of Levels

The actual values of the search keys don't affect the skip list analysis, so let's assume the keys are the integers 1 through n. Let $L(x)$ be the number of levels of the skip list that contain some search key x, not counting the bottom level. Each new copy of x is created with probability $1/2$ from the previous level, essentially by flipping a coin. We can compute the expected value of $L(x)$ recursively—with probability $1/2$, we flip tails and $L(x) = 0$; and with probability $1/2$, we flip heads, increase $L(x)$ by one, and recurse:

$$E[L(x)] = \frac{1}{2} \cdot 0 + \frac{1}{2}\big(1 + E[L(x)]\big)$$

Solving this equation gives us $E[L(x)] = 1$.

In order to analyze the expected cost of a search, however, we need a bound on the number of levels $L = \max_x L(x)$. Unfortunately, we can't compute the average of a maximum the way we would compute the average of a sum. Instead, we will derive a stronger result, showing that the depth is $O(\log n)$ *with high probability*. 'High probability' is a technical term that means the probability is at least $1 - 1/n^c$ for some constant $c \geq 1$.

In order for a search key x to appear on the kth level, we must have flipped k heads in a row, so $\Pr[L(x) \geq k] = 2^{-k}$. In particular,

$$\Pr[L(x) \geq 2\lg n] = \frac{1}{n^2}.$$

(There's nothing special about the number 2 here.) The skip list has at least $2\lg n$ levels if and only if $L(x) \geq 2\lg n$ for at least one of the n search keys.

$$\Pr[L \geq 2\lg n] = \Pr\left[(L(1) \geq 2\lg n) \ \vee \ (L(2) \geq 2\lg n) \ \vee \cdots \vee \ (L(n) \geq 2\lg n)\right]$$

Since $\Pr[A \vee B] \leq Pr[A] + \Pr[B]$ for any random events A and B, we can simplify this as follows:

$$\Pr[L \geq 2\lg n] \leq \sum_{x=1}^{n} \Pr[L(x) \geq 2\lg n] = \sum_{x=1}^{n} \frac{1}{n^2} = \frac{1}{n}.$$

So with high probability, a skip list has $O(\log n)$ levels.

A.4 Logarithmic Search Time

It's a little easier to analyze the cost of a search if we imagine running the algorithm backwards. SkipListFind takes the output from SkipListFind as input and traces back through the data structure to the upper left corner. Skip lists don't really have up and left pointers, but we'll pretend that they do so we don't have to write '$v \leftarrow$ down(v)' or '$v \leftarrow$ right(v)'.[2]

```
SkipListFind(v):
  while (v ≠ L)
      if up(v) exists
          v ← up(v)
      else
          v ← left(v)
```

[2] Leonardo da Vinci used to write everything this way, but not because he wanted to keep his discoveries secret. He just had really bad arthritis in his right hand!

Now for *every* node v in the skip list, $\text{up}(v)$ exists with probability $1/2$. So for purposes of analysis, SkipFind is equivalent to the following algorithm:

```
FlipWalk(v):
  while (v ≠ L)
      if CoinFlip = Heads
      v ← up(v)
  else
      v ← left(v)
```

Obviously, the expected number of heads is exactly the same as the expected number of tails. Thus, the expected running time of this algorithm is twice the expected number of upward jumps. Since we already know that the number of upward jumps is $O(\log n)$ with high probability, we can conclude that the expected search time is $O(\log n)$.