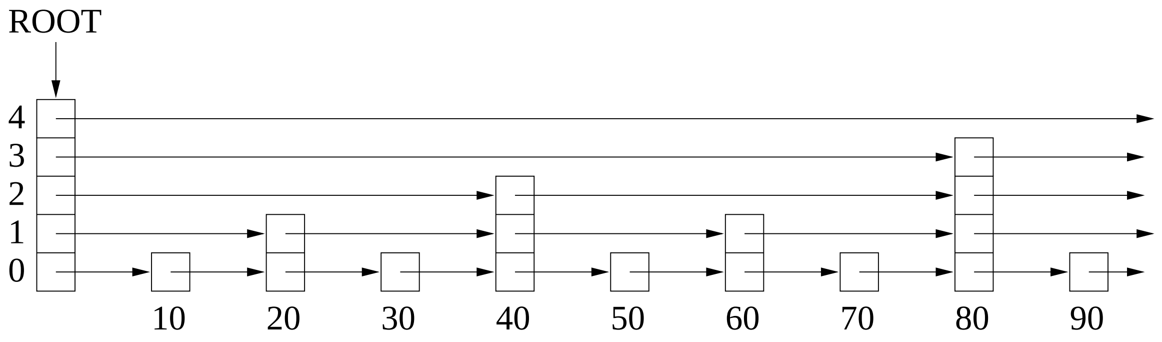
# SKIP LISTS

Richard Bartels, Jonathan Buss

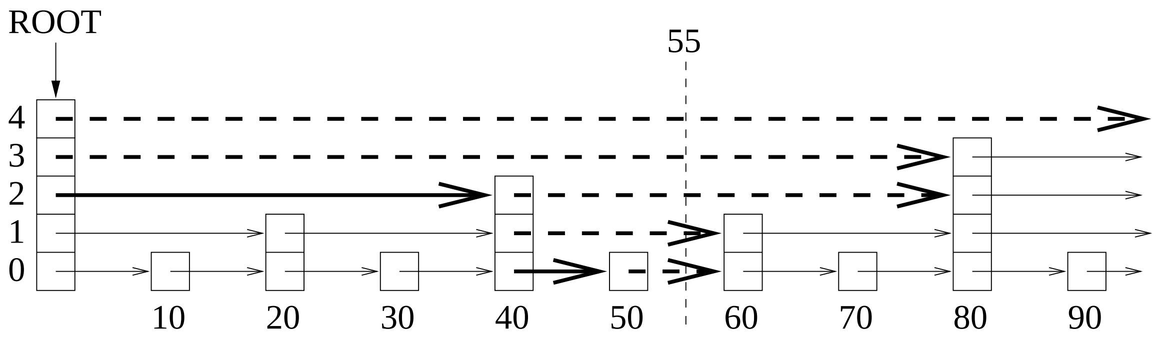
University of Waterloo
Faculty of Mathematics
Department of Computer Science

# Skip Lists

An ordinary linked list does not support binary search. But we can add pointers from position 0 to position $n/2$, from positions 0 and $n/2$ to $n/4$ and $3n/4$, and so on down until pointers reach from position $2i$ to $2i+1$ (for $0 \leq i \leq n/2$).



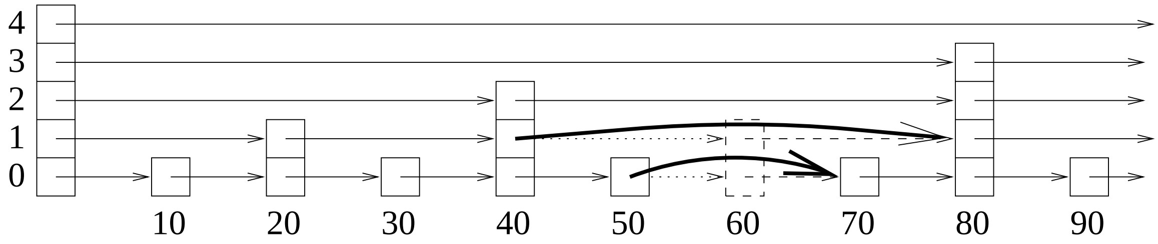

In this structure, one can perform binary search. E.g., a search for 55 follows the solid arrows and examines but does not follow the dashed arrows.




The search time is $\lfloor \log n \rfloor + 1$, just as in an array.

# Deletion

To delete a node at the end of a search, jump its predecessor's pointers over it.

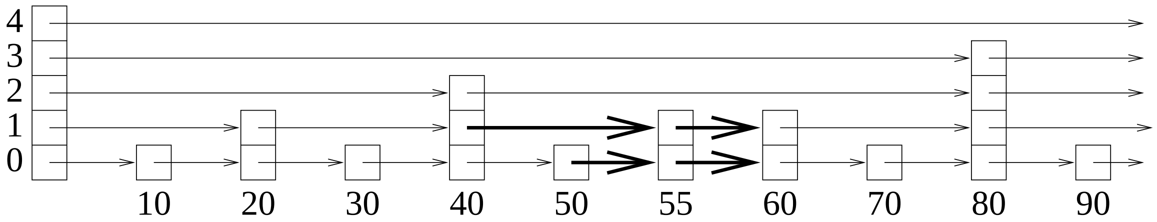



After a deletion, the list no longer has its perfect structure, but a search can still work—follow pointers at one level until you see past the search target, then drop down to the next level.

Search can now take more than $\log n$ steps. In the extreme case, deleting all of the multi-pointer nodes creates a simple linked list, of length $n/2$.

# Insertion

Insertion adds a new node into the pointer chains.




But how many pointers should the new node have?

**The Insertion Rule**:
Choose the height of an inserted node at random, with height $h$ occurring with probability $2^{-(h+1)}$.

Choose a height for each node independently of other nodes in the list.

Height $h$ is twice as likely as height $h+1$, for all $h$.

The expected number of nodes of height $h$ is the same as in the perfect list.

# Properties of Skip Lists

Expected cost of an operation:

The dictionary operations *LookUp*, *Insert* and *Delete* use $O(\log n)$ expected time in a skip list.

The probability depends only on the random choices made during insertions, and *not* on the elements in the list or on the order of operations.

Worst-case costs:

Operations can require $\Theta(n)$ time, but the probability is exponentially small.

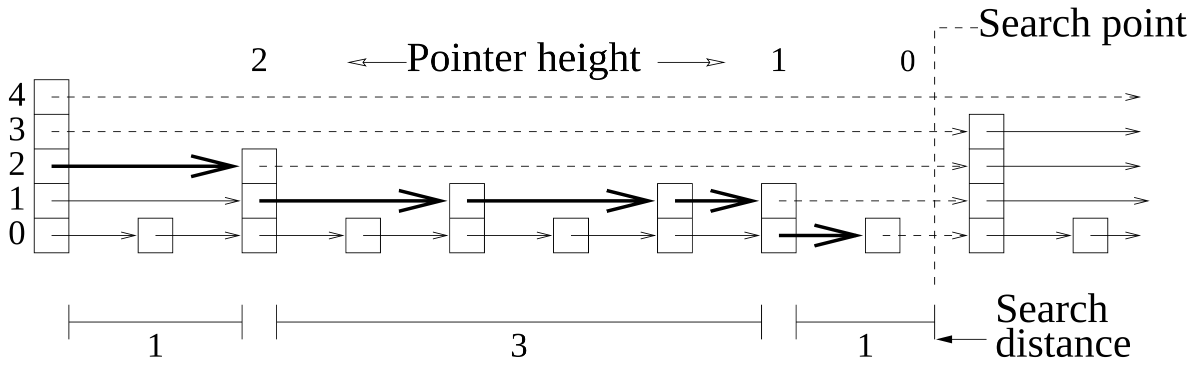
In practice:

Experimental evidence suggests that skip lists are competitive with AVL trees.

Skip lists are also easier to implement than balanced trees.

# Expected Behaviour

- Let $r$ = root height = maximum node height.
- Search: examine $r$ pointers that pass target $v$.
- Insert/Delete: change $r$ pointers.
- Remaining time: number of pointers in search path.




Average path length $= \sum_{i=0}^{r}$ (average length at level $i$).

Problem: Conditioning.

The probabilities at level $i$ are conditioned by being at level $i$—no node of height $i+1$ or more lies between the current point and the search target.

Trick to make the calculation easier:

Look at the path backwards, from its end to its beginning.

## Moving Backwards, Level by Level

The path backward has two kinds of steps:

(1) moving up one level in a node, and
(2) following a pointer backward.

Case (1) occurs if the next level exists at the current node, otherwise case (2).

At level $i$ of node $P$,

$$\Pr(\text{moving up}) = \Pr(P \text{ has a higher level}) = 1/2,$$

$$\text{and hence } \Pr(\text{following a pointer back}) = 1/2.$$

These probabilities depend only on $P$—not on the search.

Let $C(k)$ be the expected number of steps to rise $k$ levels from the current level. The two cases give

$$C(k) = \tfrac{1}{2}\left(1 + C(k-1)\right) + \tfrac{1}{2}\left(1 + C(k)\right).$$

The first term represents an upward move and the second represents a backward move.

Combining terms yields $C(k) = 2 + C(k-1)$.

Since $C(0) = 0$, we get $C(k) = 2k$.

# Going All the Way Back

To return to the root, the backward path must rise to level $r$ and then follow pointers back to the root. The expected number of pointers in the path at level $r$ is at most the expected number of nodes in the whole list that rise to height $r$, which is $2^{-r}n$.

Therefore, the expected search cost is at most

$$C(r) + 2^{-r}n = 2r + 2^{-r}n.$$

If $r \geq \log n - \log\log n$, then the second term is

$$\begin{aligned} 2^{-r}n &\leq 2^{-\log n + \log\log n}\, n \\ &= \frac{\log n}{n} n \\ &= \log n. \end{aligned}$$

If also $r = O(\log n)$, the expected search cost is $O(\log n)$.

# Remarks

Skip lists, unlike binary search trees, do not have any inherently bad inputs or sequences of operations. Skip lists depend only on the probabilities chosen by the algorithm.

To defeat a skip list, an adversary must either look into the data structure itself or know in advance what the sequence of random numbers will be.

The probability 1/2 of increasing the height of a node is not sacrosanct. Any ratio $0 < p < 1$ will work. The best ratio depends on the details of the particular implementation.

Skip lists are relatively new, their utility having been discovered in 1987 (by Bill Pugh) and published in 1990. In contrast, AVL trees were published in 1962, and 2-3 trees in 1970. Skip lists will probably be used much more in the future than they are now.