# Extended Algorithms Courses COMP3821/9801

Aleks Ignjatović

School of Computer Science and Engineering
University of New South Wales

Introduction to Randomized Algorithms: Skip Lists

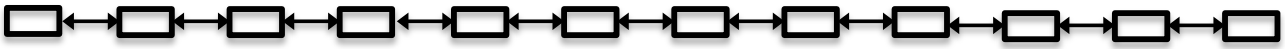
# Skip Lists

**Problem:**

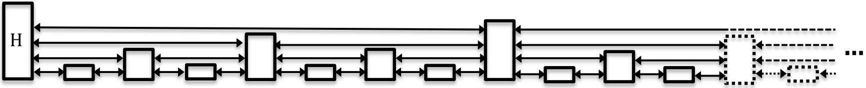
- A recent data structure - introduced in 1989 by William Pugh.
- Yes, I know, it is recent for people of my age, not at all recent for you ...
- A randomised data structure with benefits of balanced trees (e.g., AVL or Red - Black trees):
  - $O(\log n)$ **expected** time for INSERT and SEARCH;
  - $O(1)$ time for MIN, MAX, SUCC, PRED;
  - Can be enhanced so that finding the $k^{th}$ element in the list also runs in $O(\log n)$ time.
- Much easier to code than balanced trees and in practice also tend to be faster and use less space.
- Skip Lists have replaced balanced trees in many applications.

# Skip Lists

- Consider a doubly linked list:



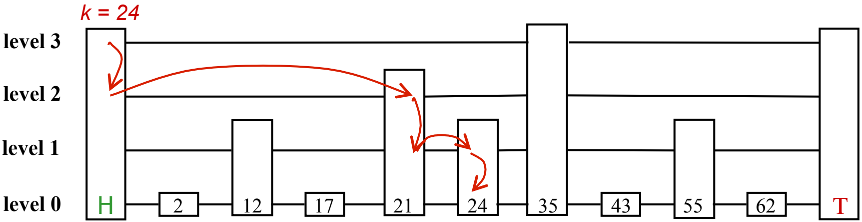
- Min, Max, Succ, Pred run in time $O(1)$.
- However, Search, Insert, Delete run in time $O(n)$.
- The culprit is searching.
- Can we modify doubly linked links to make search $O(\log n)$ expected time?
- **Idea:** make shortcuts on several levels:




This is something like the express elevators in skyscrapers which do not stop on every floor.

# Skip Lists

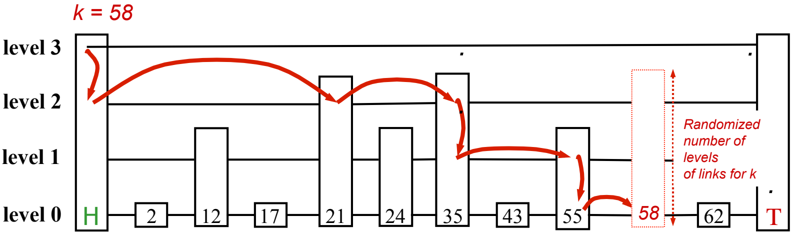
- Searching for $k$:




- Start from the head H and go as far right as you can, without exceeding $k$, using the highest possible level of links;
- then drop one level down and repeat the procedure using lower level links.
- How can we ensure such a search procedure runs in time $O(\log n)$?
- Can we link every other link on the second level, every fourth link on the third level, every eight on the fourth level and so on...

# Skip Lists

- Problem: insertions and deletions will destroy such a structure...
- We need dynamically self balancing structure.
- This is where randomisation comes into play, ensuring that in the long run the structure remains (essentially) balanced.




- **To Insert** $k$: first search to find the right place. Then toss a coin until you get a head, and count the number of tails $t$ that you got.
- Insert $k$ and link it at levels $0 - t$ from the bottom up (deciding up to what level elevators stop at that floor).

# Skip Lists

- **Deleting** an element is just like in a standard doubly linked list, but taking care of all pointers affected.
- How fast can we search for an element?
- The probability of getting $i$ consecutive tails when flipping a coin $i$ times is $1/2^i$.
- Thus, an element has links on levels $0 - i$ (and possibly also on higher levels) with probability $1/2^i$.
- If $n$ elements belong to a set, each with a probability $p$, then the expected size of that set is $n\,p$.
- Thus, an $n$ element Skip List has on average $n/2^i$ elements with links on level $i$.
- Since an element has links only on levels $0 - i$ with probability $2^{i+1}$, the total **expected** number of links per element is

$$O\left(\sum_{i=0}^{\infty} \frac{i+1}{2^{i+1}}\right) = O\left(\sum_{i=1}^{\infty} \frac{i}{2^i}\right) = 2$$

# Skip Lists

- Let $\#(i)$ denote the number of elements on level $i$.
- Since the expected number of elements having a link at level $i$ is $E_i = E(\#(i)) = n/2^i$, by the Markov inequality the probability of having at least one element at level $i$ satisfies

$$P(\#(i) \geq 1) \leq \frac{E_i}{1} = \frac{n}{2^i}$$

- Thus, the probability to have an element on level $2 \log n$ is smaller than $n/2^{2 \log n} = n/n^2 = 1/n$.
- The probability to have an element on level $k \log n$ is smaller than $n/2^{k \log n} = n/n^k = 1/n^{k-1}$.
- Thus, the probability that the highest level is $k \log n$ is also smaller than $1/n^{k-1}$.
- What is the expected value $E$ of $k$ such that $k$ is the least integer so that the number of levels is $\leq k \log n$?

# Skip Lists

$$E < \sum_{k=1}^{\infty} \frac{k}{n^{k-1}} = \left(\frac{n}{n-1}\right)^2$$

- Thus, the expected number of levels is barely larger than $\log n$.
- If an element has a link at a level $i$ then with probability $1/2$ it also has a link at level $i+1$.
- Thus, the expected number of elements between any two consecutive elements with a link on level $i+1$ which have links only up to level $i$ is equal to

$$\frac{0}{2} + \frac{1}{2^2} + \frac{2}{2^3} + \frac{3}{2^4} + \ldots = 1$$

So once on level $i$, on average we will have to inspect two elements on that level before going to a lower level.

# Skip Lists

- Thus, on average, there will be fewer than $2 \log n$ levels to go down, with visiting on average only two elements per each level.
- Consequently, on average, the search will be in time $O(\log n)$.
- For an element with links on levels $0 - i$ we have to store $2(i+1)$ pointers to other elements and the expected number of elements with highest link on level $i$ is $O(n/2^{i+1})$. Thus, total expected space for all pointers does not exceed

$$O\left(\sum_{i=0}^{\infty} 2(i+1)\frac{n}{2^{i+1}}\right) = O\left(2n\sum_{i=0}^{\infty}\frac{i+1}{2^{i+1}}\right) = O(4n) = O(n)$$

- Unless we must ensure that the worst case performance of search is $O(\log n)$, Skip Lists are a better option than BST.

# An improvement of Skip Lists

**Homework:**

- Note that accessing the $k^{th}$ largest element is still $O(n)$.
- Add something to the structure so that accessing the $k^{th}$ largest element is also $O(\log n)$ expected time.