COL 702 Lecture 20 Oct 14

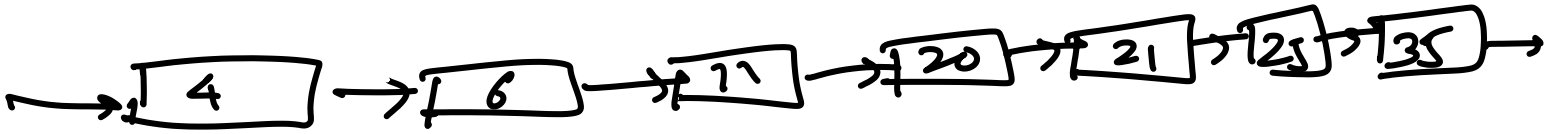
Maintaining balanced binary search trees to support dynamic dictionary operations like search, insert, delete is often quite cumbersome. Rotations / double rotations / color changes in Red Black trees / node splitting and joining in B trees, are not easy to remember. Challenge is to maintain $O(\log n)$-time for each operation

As opposed to trees, linked lists are relatively easier to maintain

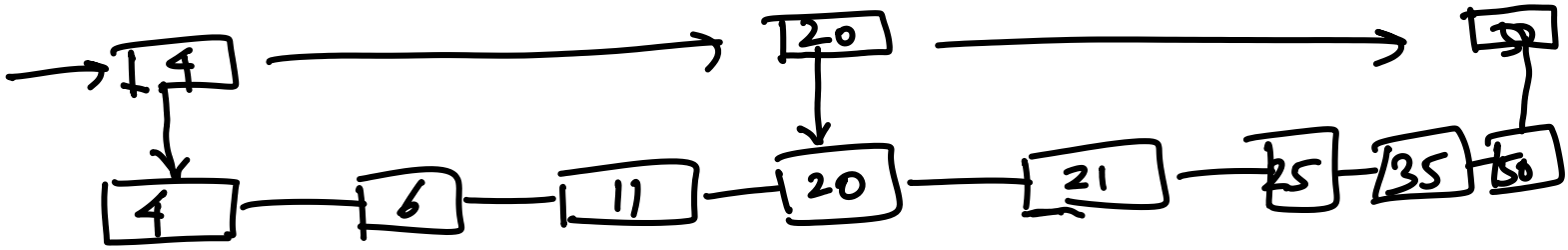
but suffer from the disadvantage of much higher time for operation, namely linear.

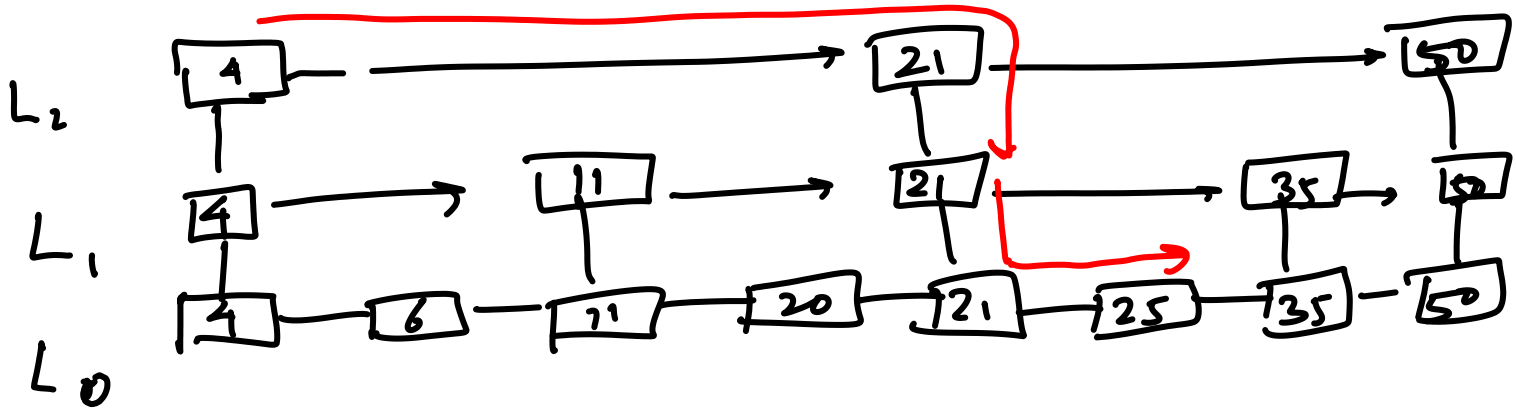
Can we combine the advantages of lists and trees?

We will maintain a sorted list to support fast searches



22





This supports $\log n$ search-time but requires a very strong invariant i.e. every alternate (every k^{th} level) must be "promoted" to the level above

How "randomized" invariant

Every element is promoted to the next level with probability $\frac{1}{2}$ (independently)

What is the "expected" gap between two promoted elements?

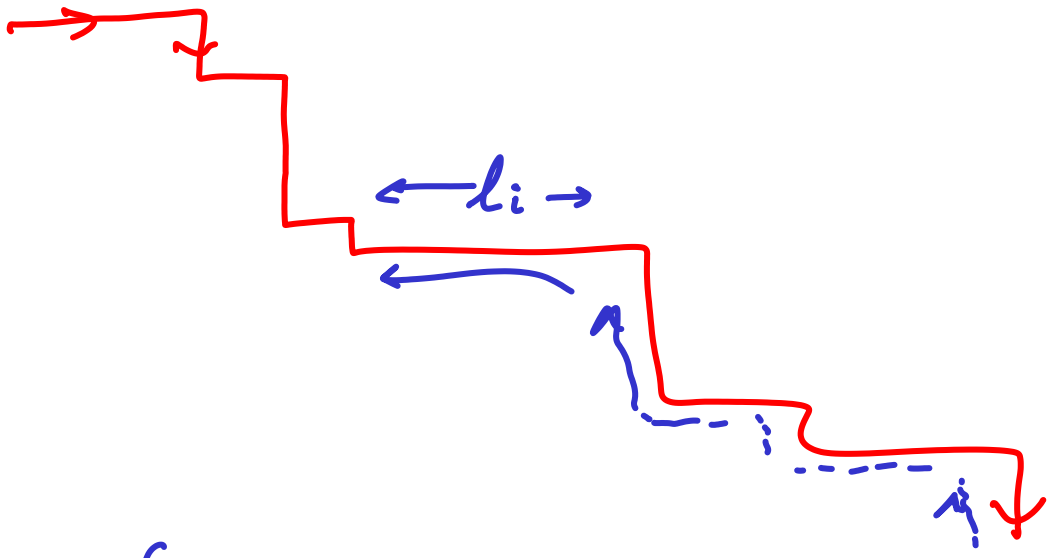
Recall that the gap determines the maximum # of traversals within a level

The expected gap is 2

What is the expected height: $O(\log n)$

What is the expected length of the search path?

Look at a search path in a backward manner



$$E[L] = E[l_1 + l_2 + \cdots l_k]$$

$$= \leq E[l_i] = 2 \cdot k$$

K can be bound by $\log n$ (by choice)

Then the Expected # of elements in level $k = \left(\frac{1}{2}\right)^k \cdot n = O(1)$ for $k = \log n$

So search time is $2k +$ Expected # element in top level

Skip Lists : a randomized data structure for dynamic dictionary

by William Pugh in 1988

Insertion : Search for the position in L_0 and promote by coin-tossing

Expected # of promotions = 2

Deletion : Reverse the process

Time for insert/delete = search time + # copies

Can be applied to "Concatenable Queues"

Split and Union of lists/sets

All elements of $L_2 \geq L_1$

H.W what is the expected height of the skip lists if we want to bound the total # elements in the topmost list by say 10.