Introduction to Algorithms March 18, 2004
Massachusetts Institute of Technology 6.046J/18.410J
Professors Erik Demaine and Shafi Goldwasser Handout 14

Lecture Notes on Skip Lists

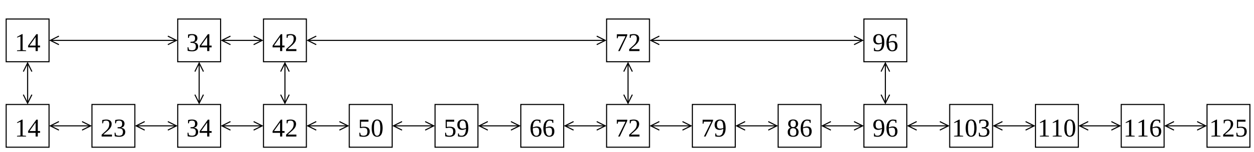
Lecture 12 — March 18, 2004

Erik Demaine

- Balanced tree structures we know at this point: B-trees, red-black trees, treaps.
- Could you implement them right now? Probably, with time... but without looking up any details in a book?
- Skip lists are a simple randomized structure you'll never forget.

Starting from scratch

- Initial goal: *just searches* — ignore updates (Insert/Delete) for now
- Simplest data structure: linked list
- Sorted linked list: $\Theta(n)$ time
- 2 sorted linked lists:
  - Each element can appear in 1 or both lists
  - How to speed up search?
  - **Idea:** Express and local subway lines
  - **Example:** $\boxed{14}$, 23, $\boxed{34}$, $\boxed{42}$, 50, 59, 66, $\boxed{72}$, 79, 86, $\boxed{96}$, 103, 110, 116, 125 (What is this sequence?)
  - $\boxed{\text{Boxed}}$ values are "express" stops; others are normal stops
  - Can quickly jump from express stop to next express stop, or from any stop to next normal stop
  - Represented as two linked lists, one for express stops and one for all stops:



  - Every element is in linked list 2 (LL2); some elements also in linked list 1 (LL1)
  - Link equal elements between the two levels
  - To search, first search in LL1 until about to go too far, then go down and search in LL2

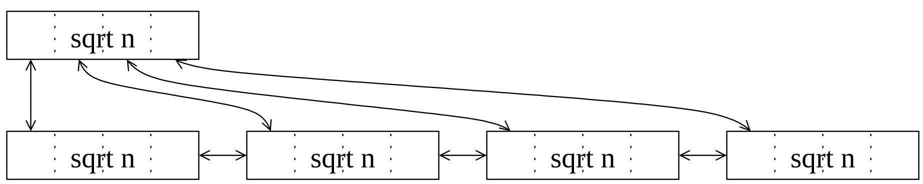
– Cost:

$$\text{len}(\text{LL1}) + \frac{\text{len}(\text{LL2})}{\text{len}(\text{LL1})} = \text{len}(\text{LL1}) + \frac{n}{\text{len}(\text{LL1})}$$

– Minimized when

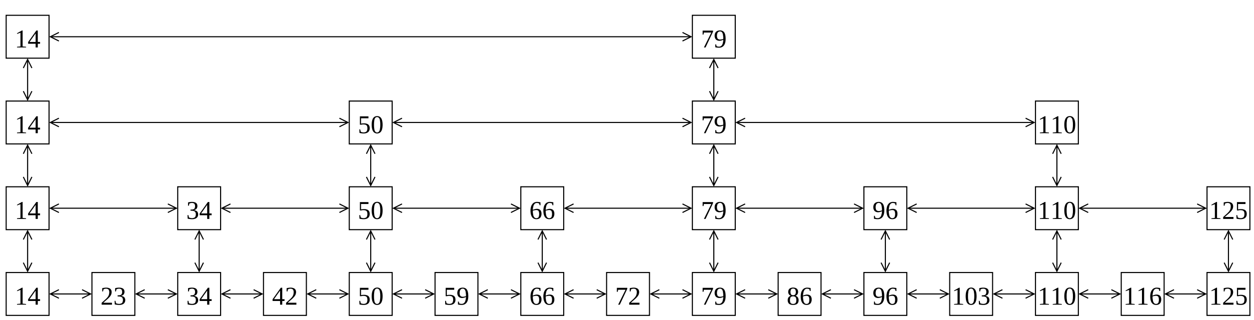
$$\begin{aligned} & \text{len}(\text{LL1}) = \frac{n}{\text{len}(\text{LL1})} \\ \Rightarrow\ & \text{len}(\text{LL1})^2 = n \\ \Rightarrow\ & \text{len}(\text{LL1}) = \sqrt{n} \\ \Rightarrow\ & \text{search cost} = 2\sqrt{n} \end{aligned}$$

– Resulting 2-level structure:



- 3 linked lists: $3 \cdot \sqrt[3]{n}$
- k linked lists: $k \cdot \sqrt[k]{n}$
- $\lg n$ linked lists: $\lg n \cdot \sqrt[\lg n]{n} = \lg n \cdot \underbrace{n^{1/\lg n}}_{=2} = \Theta(\lg n)$

– Becomes like a binary tree:



– **Example:** Search for 72

∗ Level 1: 14 too small, 79 too big; go down 14

∗ Level 2: 14 too small, 50 too small, 79 too big; go down 50

∗ Level 3: 50 too small, 66 too small, 79 too big; go down 66

∗ Level 4: 66 too small, 72 spot on

Insert

- New element should certainly be added to bottommost level
  (Invariant: Bottommost list contains all elements)
- Which other lists should it be added to?
  (Is this the entire balance issue all over again?)
- **Idea:** Flip a coin
  - With what probability should it go to the next level?
  - To mimic a balanced binary tree, we'd like half of the elements to advance to the next-to-bottommost level
  - So, when you insert an element, flip a fair coin
  - If heads: add element to next level up, and flip another coin (repeat)
- Thus, on average:
  - $1/2$ the elements go up 1 level
  - $1/4$ the elements go up 2 levels
  - $1/8$ the elements go up 3 levels
  - Etc.
- Thus, "approximately even"

Example

- Get out a real coin and try an example
- You should put a special value $-\infty$ at the beginning of each list, and always promote this special value to the highest level of promotion
- This forces the leftmost element to be present in every list, which is necessary for searching

... many coins are flipped ...
(Isn't this easy?)

- The result is a skip list.
- It probably isn't as balanced as the ideal configurations drawn above.
- It's clearly good on average.
- Claim it's really really good, almost always.