1 Introduction

How to do binary search on a list?

Idea of binary search: Assuming the list is sorted. Look at the middle and eliminate the half of it.

- Working on array perfectly.
- For list, you need to find the middle item. It is slow.
- In order to do it fast, you need to store something around the middle item and recurse on that part.
- You need to store a whole set of middle items.
- Scarier part: you need to update those middle items during insertion.
- This idea motivates another data structure.

2 Skip List

Basic Idea. From 1980. The bottom stores the list of the full items. Above that list, you have a list of hierarchies (storing the references.) The higher the hierarchy, the fewer items the list has. The hierarchy sometimes is called level, so it has an idea of height. The ends of each list have negative infinity or positive infinity, called sentinels. The topmost level has only two sentinels. The height of the towel is the number of levels that keep a certain item. Top left node sentinel is what passes into the argument. Every node knows what is to the right and to the bottom of it. We will chose the height of tower randomly, not guaranteeing the upper level chooses medians, but by luck.

Searching steps

- Searching from the top level.
- If the key is too small, we move forward.
- If too big, we come back and drop down.

We are interested in finding the path of predecessors that we keep track of during searching. This is very helpful for insertion.

Inserting steps

To determine a height of a tower: flip a coin until you get tails.

So that

$$\mathbb{P}(\text{tower has height} \geq i) = \mathbb{P}(\text{we get head at least } i \text{ items}) = \frac{1}{2^i}$$

- Figure out how high to make the tower **first**, by the coin flipping above.
- You need to figure out the height first because you may need to expand the entire skip list beforehand (when you have to have items on the top level.)

- The path of predecessors pre-pend the newly item.

Runtime for search and insert

$$O(\text{height of skip list}) = \max_k \{\underbrace{\text{height of the tower of key } k}_{=:X_k \text{ (random variable)}}\}$$

- $\mathbb{P}(X_k = 0) = \frac{1}{2}$
- $\mathbb{P}(X_k = 1) = \frac{1}{4}$
- $\mathbb{P}(X_k = i) = \frac{1}{2^{i+1}}$

So let's calculate the expected height (of one tower, denoted X_k.)

$$\begin{aligned} E[X_k] &= \sum_{i=0}^{\infty} i \cdot \mathbb{P}(X_k = i) \\ &= \sum_{i=0}^{\infty} \frac{i}{2^{i+1}} = \mathbf{1} \quad \text{(wow)} \end{aligned}$$

Another way to calculate the expected height. (This can be thought of a trick to prove the above sum.)

$$\begin{aligned} E[X_k] &= \sum_{i=1}^{\infty} \mathbb{P}(X_i \geq i) \\ &= \sum_{i=1}^{\infty} \frac{1}{2^i} \\ &= 1 \quad \text{(geometric series)} \end{aligned}$$

The expected tower of one key is 1, not quite the expected height of the entire skip list.

The expected height of the skip list.

$$\begin{aligned} E[\text{height of skip list}] &= E[\max_k \{X_k\}] \\ &= \mathbb{P}(\max_k \{X_k\} \geq i) \\ &= 1 - \mathbb{P}(\max_k \{X_k\} < i) \\ &= 1 - \prod_k \mathbb{P}(X_k < i) \\ &= 1 - \left(1 - \frac{1}{2^i}\right)^n \\ &= \text{It is getting really messy not, although feasible...} \end{aligned}$$

Instead, we will prove something related.

$$\begin{aligned}\mathbb{P}(\max_k\{X_k\} \geq i) =& \mathbb{P}(\text{at least one } X_k \text{ is} \geq i)\\ &\leq \sum_k \underbrace{\mathbb{P}(X_k \geq i)}_{\frac{1}{2^i}}\\ &= \frac{n}{2^i}\end{aligned}$$

$$\begin{aligned}\mathbb{P}(\text{height of skip list } \geq 3\cdot\log(n)) &\leq \frac{n}{2^{3\log(n)}}\\ &= \frac{1}{n^2}\end{aligned}$$

$$\mathbb{P}(\text{height of skip list } < 3\cdot\log(n) = 1 - \frac{1}{n^2}$$

In the practical purposes, n is very huge, so $1-\frac{1}{n^2}$ is approaching 0. In other words, **the height of a skip list is** $\leq 3\log(n)$ **with high probability.** For searching, the height is bounded by order $\log(n)$, but we do not know how often we need to go forward.

The expected space of a skip list

Space is proportional to $n + \sum X_k$. (The number of nodes that are not sentinels.) The expected space is proportional to $E[n + \sum X_k] = n + \sum_k E[X_k] = 2n$.

Back to search

The number of drop downs is the expected height $\in O(\log(n))$.

The number of forward steps is not easy, so we will do a different approach:

Consider the search path Π and its reverse Π^{-1}. Define $C(j)$ = the expected # of steps to get to some node on Π on layer $h-j$, where h is the top layer.

- $C(0) = 0$
- $C(j) = \mathbb{P}(\text{tower has height } h-j)\cdot C(\text{if } I \text{ came from left}) +$
  $\mathbb{P}(\text{tower has height} > h-j)\cdot C(\text{if I came from above}) + 1$

$$\begin{aligned}C(j) &= \begin{cases}0 & \text{if } j = 0\\ \frac{1}{2}C(j) + \frac{1}{2}C(j-1) + 1 & \text{if } j > 0\end{cases}\\ &= C(j-1) + 2 = 2j\end{aligned}$$

Time for search: $O(C(h)) = O(2h) = O(\log(n))$.

Summary Skip list has expected time $O(\log(n))$ for search, insert, (and delete). You can do *range-search* more easily with skip list than trees.

3 Dictionaries for biased search requests

Idea.

- Some requests are much more frequent.
- Tailer the dictionary so that these are fast.
- Two scenarios:
  - We know access-probabilities of all items. (Access is insert of search.)
  - We don't.

Array

- Put the highest probability to the left and the lowest to the right.
- The other ones are sorted in descending order.
- We have to know the items beforehand, so there is no insert in this case.
- The actual cost to retrieve an item x during search. Then, the access costs are $O(\text{index of x} + 1)$. The plus one is just for index of zero.

Proof of expected cost of the above implementation: find two ones that they are out of order if the array is not sorted by probability. Then exchange them, so the cost will go down.

Binary Search Tree What would you put on the root? Hint: not the one with highest probability, at least not always the case.