Skip Lists and Bloom Filters

Srinidhi Krishnamurthy

May 12, 2017

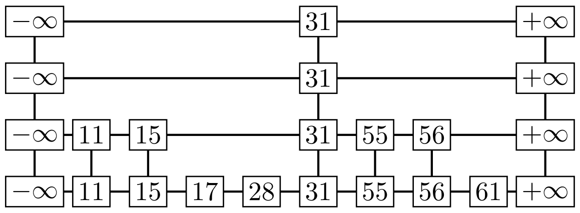
1 Introduction

Adding and element of randomness can often aid in creating an efficient data structure. With the advent of large and vast amounts of data, traditional techniques will not suffice. Let us investigate Skip Lists and Bloom Filters, two data structures that use randomization and clever techniques to solve problems and make implementaion easier than traditional techniques.

2 Skip Lists

2.1 Overview

Skip lists were first described in 1989 by William Pugh and can be seen as a pretty modern data structure. Skip lists are a probabilistic data structure and have the same goal as balanced binary search trees (yet have a simpler implementation method for many applications). Skip lists allow search within an ordered sequence of elements. By adding multiple layers to a sorted linked list, we can make the the $O(N)$ search time for a linked list much faster.



2.2 Search

There are vertical arrows between the common nodes on different levels in the linked list. A Skip list can perform a membership query in $O(\log(N))$ on average and can be $O(N)$ in the worst case. To achieve this, we take advantage of the multiple layers so that we will be able to "skip" over other elements in a search routine.

Algorithm 1 Skip List Search [n levels]

```
Result: Finding element $x$ within a skip list
Start at the top layer.
while Have not found element do
    Traverse the topmost layer until traversing further would result in an element that is larger than the one being searched for.
    Go down a level.
    Traverse the next layer until traversing further would result in an element that is larger than the one being searched for.
    if Found Element then
        Break out of loop
    end
end
```

We can show that we can achieve logarithmic search time almost every time if we have $log(N)$ levels and this would be ideal. We can also see that this looks pretty similar to a binary tree because of the structure and search routine. If our goal was only search, skip lists with $log(N)$ levels would perform well. While they may not be as good as balanced binary search trees, we can clearly see that it is easier to implement.

However, if we want to insert and delete elements we are going to have to do this cleverly so that we can maintain the ideal structure of the skip list. We don't want to take $O(N)$ time for these updates so we are going to employ some randomness and introduce some probabilistic aspects.

2.3 Insertion

A Skip list can insert a member in $O(\log(N))$ on average and can be $O(N)$ in the worst case.

Algorithm 2 Skip List Insertion [n levels]

```
Result: Inserting element $x$ within a skip list
Using the Search routine defined above, see where the element fits into the bottom most list. Insert the element
  in this position.
  while Fair Coin is not Tails do
      if Heads then
          Promote Element to Next Level
      end
  end
If we get to the highest layer we can make decisions bases on what we are trying to implement.
```

2.4 Deletion

A Skip list can delete a member in $O(\log(N))$ on average and can be $O(N)$ in the worst case.

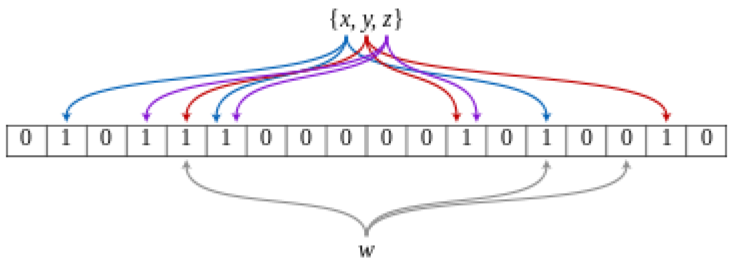
Algorithm 3 Skip List Deletion [n levels]

```
Result: Deleting element $x$ within a skip list
Using the Search routine defined above, see where the element fits into the bottom most list.
  Follow the pointers up the ladder to remove x from all the other levels above it.
```

3 Bloom Filters

3.1 Overview

The Bloom Filter data structure answers the set membership question. Bloom filters are pretty space efficient but their is also a randomness associated with them. A Bloom Filter can determine whether an element is NOT a member of the set or if an element is PROBABLY a member of the set. We can never know for sure with a Bloom Filter if an element exists in a set because of the Type I errors (false positives) its produces. Furthermore, this structure does not support deletion of elements from the set meaning that the user would have to be careful during the construction of the structure.



3.1.1 Construction

Consider a bit array of n elements with all the elements set as false initially and k unique hash functions that map input elements to values in the range of the indices of the bit array $[0,\ n-1]$. When we add an element to the bit array, we simply compute all the hash function for the element and generate a list of indices. If the value of the bit array at these indices is already true, we keep it that way, else, we flip the false values to true values. We keep adding elements in this fashion to construct our Bloom filter.

3.1.2 Query

To query for an element in the set, pass it to each of the k hash functions to get k integer array positions. If any of the bits at these positions are false, the element is definitely not in the set. If all of the bits are 1, then the element is PROBABLY in the set; specifically, either the element is in the set or the bits were coincidentally set to 1 by hashes of different elements. Now we can see why it is impossible to remove elements from the set because there is no way to know for sure if a given element is in the set.

3.2 Applications of Bloom Filters

- Quora has a shared bloom filter in their backend to filter out questions that people have seen before. It is much faster and more memory efficient than previous solutions (Redis, Tokyo Cabinet and DB) and is a situation where false positives are okay.
- Yahoo mail requests a bloom filter representing your contact list from Yahoo servers. When you send an email to say 3 people, the client quickly checks the cached bloom filter for those 3 email addresses to see if they are contacts to look up their 'contact' name (Google probably does this too).
- Facebook uses bloom filters for typeahead search, to fetch friends and friends of friends to a user typed query.
- Google uses a bloom filter in Chrome to identify malicious URLs. Rather than checking every malicious URL, they use a bloom filter and warn the user of a possible malicious website. This is a case where a false positive is okay because we don't really care if the user takes an extra step to get to a website.
- All in all, any situations where a false positive is okay can use a Bloom filter to quickly assess set membership.