Data structures
Skip lists

1 Introduction

Two commonly used data structures are sorted arrays and sorted linked lists. A sorted array makes it very easy to locate a certain key (binary search is O(lg n)). Adding or removing items however is O(n). A linked list on the other hand allows for O(1) insertions and removals but requires O(n) to search. Note that we use the data structure as a dictionary here since we access the data through the keys.

A Skip list is an advanced data structure that is a compromise between these two. It provides efficient (O(lg n)) performance for the search and insertion operations. It is a somewhat special since it is a probabilistic data structure that uses a random factor to build the structure.

2 Skip lists

A skip list is based on a linked list but instead of just a single linear structure, it has multiple layers. The bottom layer is an ordered linked list that stores all the data points. The higher layers provide shortcuts or "express lanes" to quickly locate a data point. They contain references to a subset of the datapoints from the previous level. For a perfect Skip list, the ratio of the elements at each level compared to the previous level is fixed (e.g. 1/2 or 1/4). This is however difficult to enforce when the data structure supports dynamic adding and removing of elements. In practice, a randomized technique is used where an element in layer i appears in

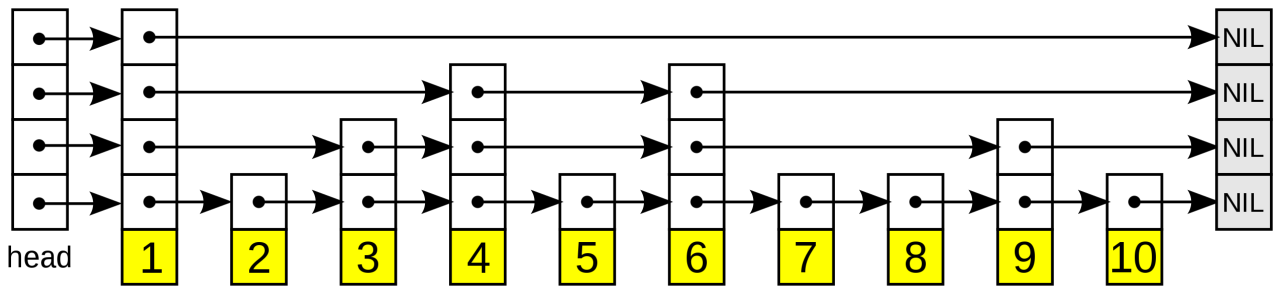

Figure 1: Skiplist: source https://en.wikipedia.org/wiki/Skip_list

layer $i+1$ with some fixed probability p. For $p = 1/2$ we then expect that the second level will contain $n/2$ references to elements, the third level $n/4$ and so on.

3 Operations

- A search for a target element begins at the head element in the top list, and proceeds horizontally until the current element is greater than or equal to the target. If the current element is equal to the target, it has been found. If the current element is greater than the target, or the search reaches the end of the linked list, the procedure is repeated after returning to the previous element and dropping down vertically to the next lower list.
- Insertion starts with searching for the location where the element should be inserted. A new node is created at that location in the lowest level. Then, a set of random trials is performed (flip a coin) to decide if this element should also be present in the higher levels.
- Deletion again starts with searching. The linked list structure makes it easy to delete the node. We just have to make sure that we also delete it from the higher levels if needed.

4 Assignment

- The start code contains a Java interface "SortedList.java" together with a linked list based implementation "SortedLinkedList.java". There is some example benchmark code in "Main.java" and a unit test to check the operations in "UnitTests.java".
- Implement a SkipList based implementation of the SortedList interface.
- Start small by adding a few elements to the list. Use the debugger to make sure the operations work the way you expect.
- The ideal number of levels depends on the number of elements in the list ($lg\ n$). To make the implementation easier, you can pass the number of levels as an argument in the constructor.