


Introduction

- Skip Lists can be used to implement the Dictionary ADT
- We will see that Skip Lists provide an efficient, practical solution to efficient insert, delete, and search dictionary operations. Furthermore Skip Lists are not very hard to implement.

Skip Lists

- Alternative implementation of ordered dictionaries
- Inspired by binary search
- Improvement over lookup tables:

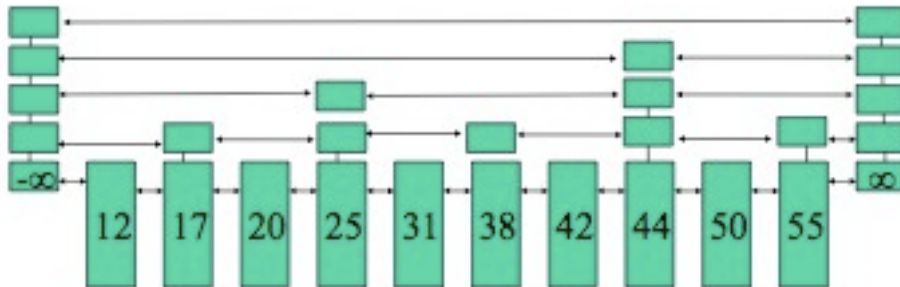
	Skip List	Array(sorted)	Array(unsorted)
lookup:	$O(\log n)$	$O(\log n)$	$O(n)$
insert:	$O(\log n)$	$O(n)$	$O(1)$
remove:	$O(\log n)$	$O(n)$	$O(n)$



- The Skip List structure uses randomization to obtain good average case performance.

Skip Lists

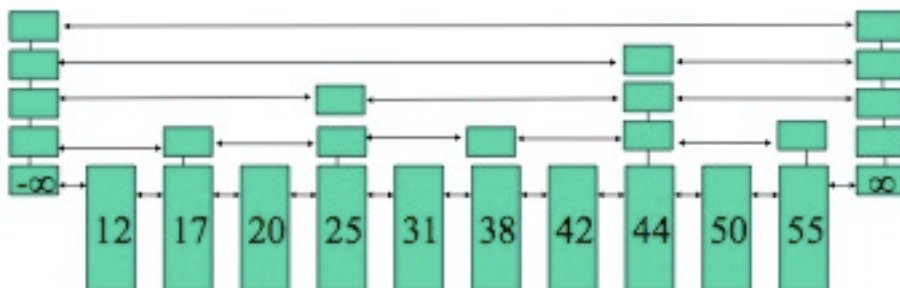
- Idea is to achieve binary search performance with linked lists.
- We can use several linked lists where the distance between consecutive elements (from the original list) double (approximately) as the height of the list increases.



Skip Lists

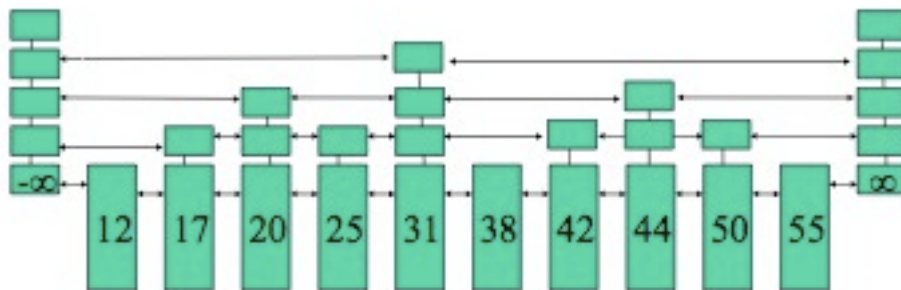
Considerations for convenience of exposition.

- We can double link the lists in the horizontal and vertical directions.
- Sentinels are used at the beginning and end of the list.



Skip Lists

- Maintaining this type of structure through inserts and deletes would be difficult.
- The key idea behind Skip Lists is to approximate this type of behaviour, in a probabilistic sense.



Determinism vs Non-determinism

- A program P is **deterministic** if P exhibits the exact same behaviour each time it is run from the same initial state.
- A program P is **non-deterministic** if P exhibits different behaviour each time it is run from the same initial state.
- Skip List operations introduce random/non-deterministic behaviour by using values obtained from a **pseudo-random number generator**.
- A good pseudo-random number generator uses a deterministic algorithm to generate a sequence of values that appear to be randomly distributed. Strictly speaking, the behaviour is only **pseudo-random**. However, actual performance is equivalent to performance one would obtain using truly random numbers.

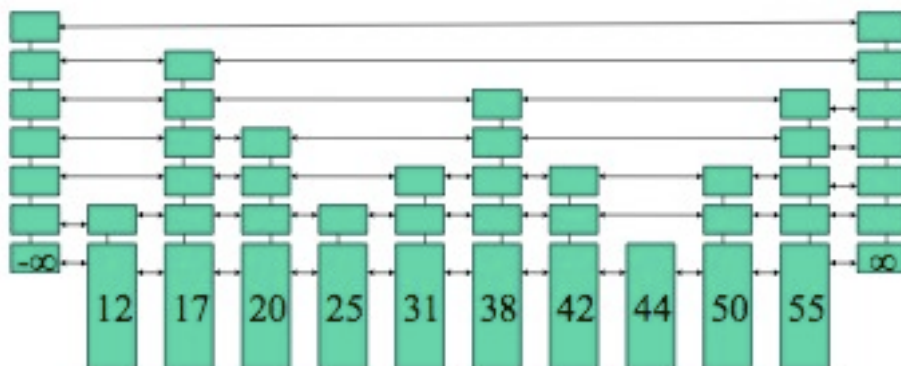
Skip Lists Formally

- A **skip list** S for dictionary D consists of a number of **sub-lists**

$$\{S_0, S_1, S_2, \dots, S_h\}$$

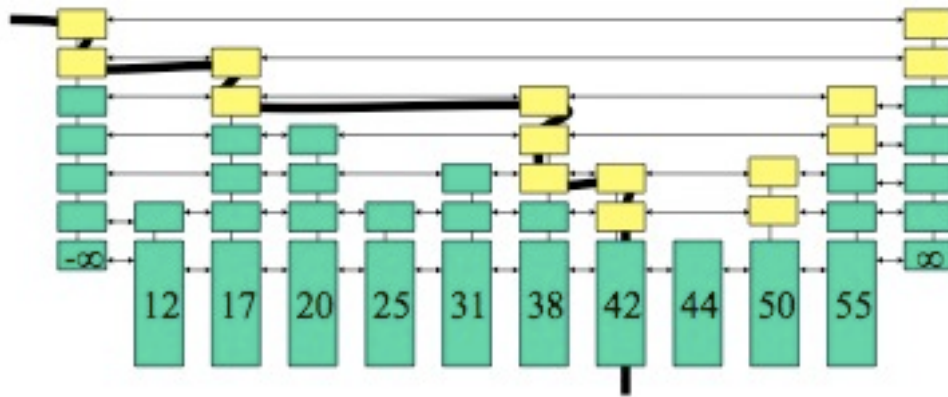
- h is called the **height** of the skip list
- Each list S_i stores
 - a **subset** of the items in D sorted by **non-decreasing** keys
 - **two special** values $-\infty$ and $+\infty$, where $-\infty$ is smaller than every possible key that may be inserted into D and $+\infty$ is larger
- **Additionally,**
 - S_0 contains every item of D and $-\infty, +\infty$
 - For $i=1, \dots, h-1$, list S_i contains (in addition to $-\infty$ and $+\infty$) randomly generated subset of the items in list S_{i-1}
 - List S_h contains only $-\infty$ and $+\infty$

A Skip List



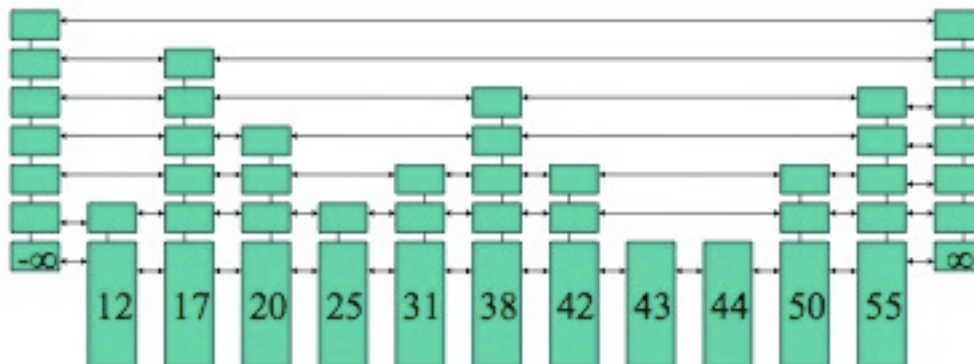
Inserting into a Skip List

- Insert 43. First find out where to put it, so search.



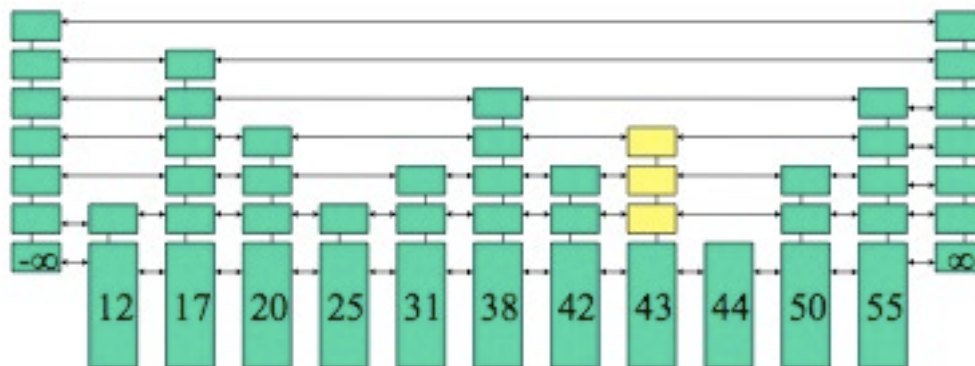
Inserting into a Skip List

- Now insert. How high should the tower be?



Inserting into a Skip List

- Flip a coin until we get a head. For example three tails, followed by one head.



Algorithm SkipInsert(k)

Input: Item(k, e).

Output: None.

$p \leftarrow$ SkipSearch(k);

$q \leftarrow$ insertAfterAbove(p , null, (k, e))

NewLevel \leftarrow RandomLevel();

for I from 0 to NewLevel do

 while above(p) == null do find the correct

$p \leftarrow$ before(p) predecessor

$p \leftarrow$ above(p)

$q \leftarrow$ insertAfterAbove($p, q, (k, e)$)

Algorithm RandomLevel

Input: None

Output: A value between 0 and MaxLevel, where MaxLevel denotes the height of the highest allowable tower.

```
h ← 0;
while (Random() < 1/2 and h ≤ MaxLevel) do
    h++;
return h
```

Algorithm insertAfterAbove(p,q,I)

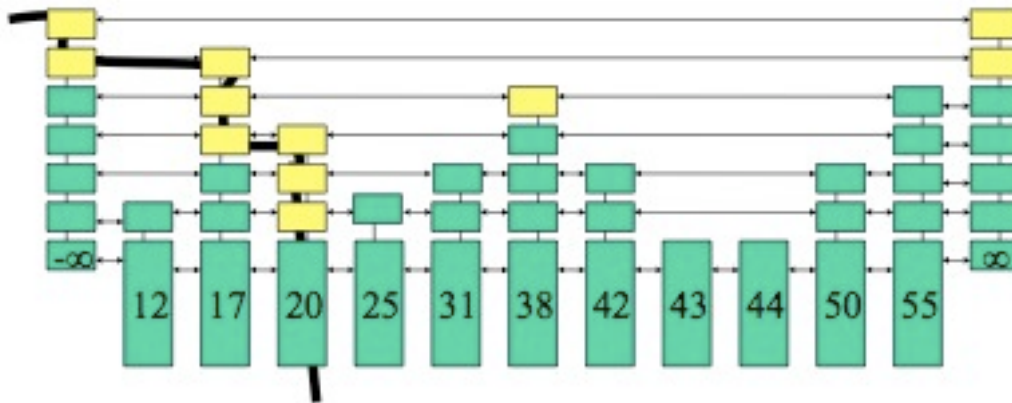
Input: Positions p and q, and Item I.

Output: A position for an item inserted after p and above q.

```
SLNode r ← new SLNode() ;
r.setItem(I);
r.setAfter(after(p)); after(p).setBefore( r );
r.setBefore(p);p.setAfter( r );
r.setBelow(q); if (q ≠ null) q.setAbove( r);
r.setAbove (null);
return r;
```

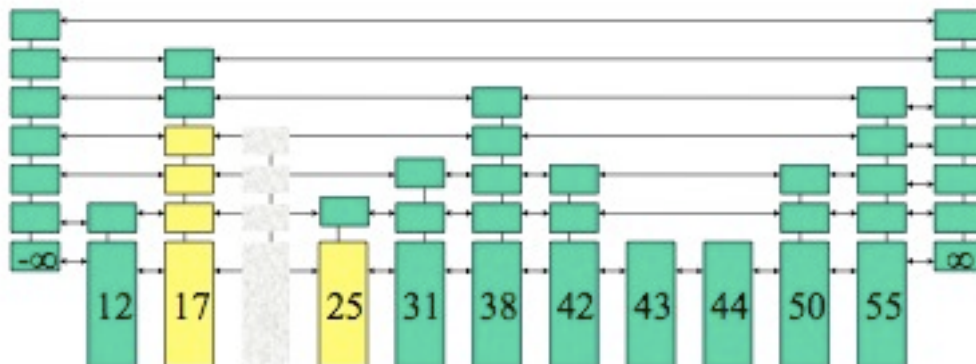

Deletions in a Skip List

- Delete 20. Find it first.



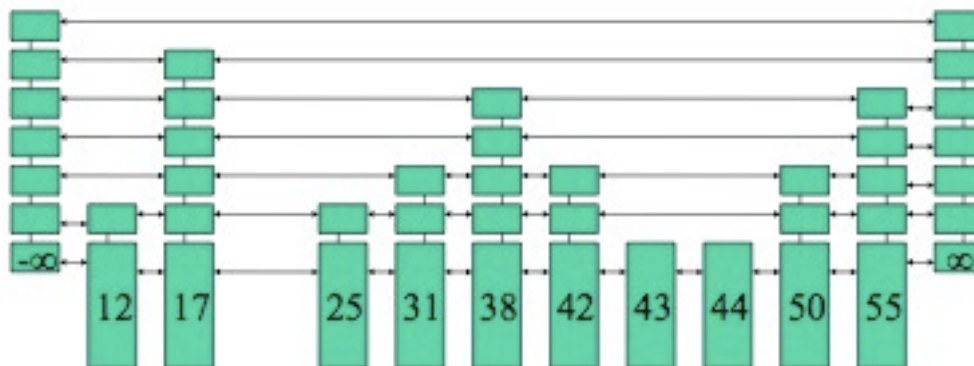
Deletions in a Skip List

- Delete 20. Then remove it.



Deletions in a Skip List

- Delete 20. Then remove it.



Analysis.

- In the worst case Skip Lists may behave very poorly. (i.e. $O(N + \text{MaxLevel})$ each for insert, delete, and find.)
- We use probabilistic analysis to show that the Dictionary operations are expected to be performed very quickly.

Space Analysis

- The probability that a tower is of height h , is the same as the probability of getting h tails in a row when flipping a coin, which is $1/2^h$. A good choice for MaxLevel is $O(\log N)$ for a Skip List of N elements. Even if we don't bound the levels with MaxLevel the probability that we go beyond it is extremely small.
- This immediately allows us to determine the total average space complexity. In a skip list storing n elements we expect $n/2^h$ elements to be in the list at level h . This leads to the following familiar sum:

$$n + n/2 + n/4 + \dots < 2n$$

- Which we recognize as $O(n)$.

Skip Lists: Average Height

- For any single element, probability it gets to level i ?

$$\frac{1}{2} * \frac{1}{2} * \frac{1}{2} * \dots * \frac{1}{2} \text{ (i times)} = \frac{1}{2^i}$$

- Probability that at least one of n elements got to level i $\leq \frac{n}{2^i}$

$$\text{Try } i = \log_2 n: \quad \text{prob} \leq \frac{n}{2^i} = \frac{n}{2^{\log_2 n}} = \frac{n}{n} = 1$$

$$\text{Try } i = 3\log_2 n: \quad \text{prob} \leq \frac{n}{2^i} = \frac{n}{2^{3\log_2 n}} = \frac{n}{n^3} = \frac{1}{n^2}$$

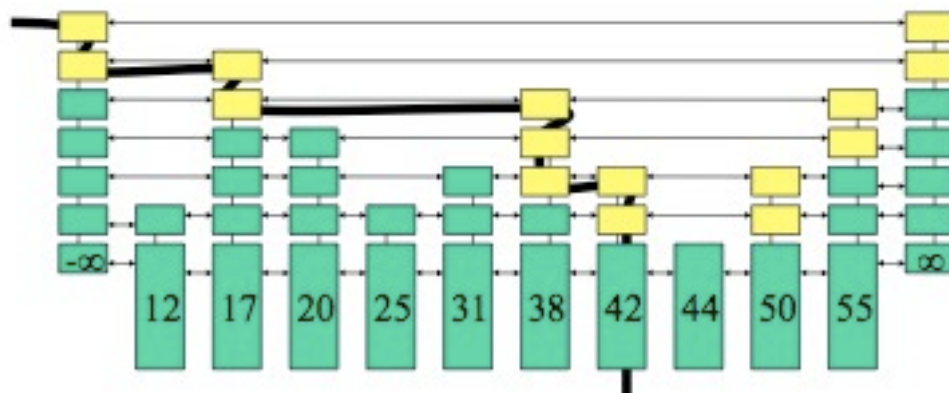
- **Conclusion:** Height h of skip list without MaxLevel bound
 - **very likely** to be at least $\log_2 n$
 - **very unlikely** to more than $3\log_2 n$
 - So, expected height h is $O(\log n)$

Analysis

- We give a probabilistic analysis of the cost of SkipSearch. The cost of Insert and Delete follow immediately.
- Assume that we have found an item and let p denote its position at level 0. We analyze the cost of the search by following the search path backwards.

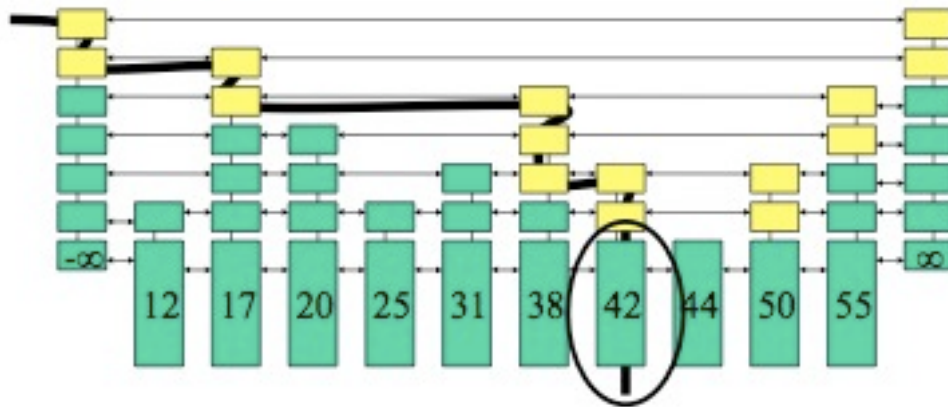
Backtrack analysis

- Found 42.
- Our backtracking can go up or to the left.



Backtrack analysis

- The probability that we go up is $1/2$. Observe that once we are up at the top level we get to the front of the list in one more step.



Backtrack analysis

- The number of steps needed to reach the top level of the Skip List in effect counts the total number of steps needed to find an item.
- We go up one level every time we flip and get a tail. The expected number of flips until we get a tail is 2.
- Thus the expected number of steps needed to get to the top level, say level j , is $2j$.
- With high probability j is $O(\log n)$. Thus the cost of a search is $O(\log n)$ with high probability.

Backtrack analysis

- Therefore the expected cost to search, insert, delete, in a Skip List with N elements is $O(\log N)$.

Skip Lists: Lessons Learnt

- inspired by binary search
- randomization for algorithm & data structure design
- coin flipping in C++
- deterministic vs. non-deterministic programs
- guaranteed worst case analysis vs. average worst case analysis
- There are no bad insertion sequences. A binary search tree has expected height $O(\log n)$ for a random insertion sequence. However for bad insertion sequences the height can be $O(n)$. In the case of Skip Lists the length of the search path depends on random "coin flips". We have to get a highly unlikely sequence of coin flips to realize poor performance.