Indian Institute of Science
Bangalore, India
भारतीय विज्ञान संस्थान
बंगलौर, भारत

DS286 | 2016-09-14

L10: Dictionary & Skip List

Yogesh Simmhan

simmhan@cds.iisc.ac.in

Slides courtesy Venkatesh Babu, CDS, and
Skip Lists, CMSC 420: Data Structures, UMaryland Spring 2008

CDS
The Department of Computational and Data Science

Dictionaries

- Dictionaries store elements so that they can be located quickly using keys
- A Dictionary may hold bank accounts
  - Identified by a/c no. (key)
  - Stores additional information (value)
    - Current balance
    - Name, address
    - History …
  - Unique key, i.e. no two keys are the same
- A Dictionary of words and meanings, people and contact info (email, telephone)
  - Word forms the key
  - May have multiple definitions (values) for same word (key)
  - i.e. a Dictionary with duplicate keys…not our focus

Dictionary ADT

- Collection of pairs.
  - (key, element)
  - Pairs have distinct keys (i.e. no duplicates)
  - The main operation: Searching by key
- Operations.
  - `find(key)`
  - `insert(key, value)`
  - `erase(key)`
  - `size()`
  - `isEmpty()`

Dictionary ADT

- Different data structures to realize dictionaries
  - Arrays, linked lists
  - Hash tables
  - Binary trees
  - Red/Black trees
  - AVL trees
  - B-trees

Dictionary using List

- Dictionary stored as a List of <key,value> items
  - Insertion time? Searching time?
- Dictionary stored as a Ordered List of <key,value> elements, ordered by key
  - What's the advantage?

Dictionary using Ordered Linked List

- insert(key, value)
  - Start from head of linked list
  - Locate item whose key is just larger than insertion key
  - Insert before that item
  - Update head of linked list if necessary
- find(key)
  - Start from head of linked list
  - Search till item's key is equal to search key (or) item's key is larger than search key
  - Return item if found
  - Stop search if key is larger
- Time complexity? Worst case? Average?

Dictionary as a Sorted List

- Idea: **Divide and Conquer**
- Narrow down the search range in stages
- E.g. find (8)
- Start with `floor(|search space| / 2)`
- 2 5 8 **9** 11 17 20 22
- 2 **5** 8 9 11 17 20 22
- 2 5 **8** 9 11 17 20 22

Binary search over array

Dictionary as a Sorted List

```
int bsearch(KVP[] list, int start, int end, int k) {
   if (end < start) return -1  // No match!
   i = start+(end-start)/2     // midpoint
   if (list[i].key == k)  // Found!
      return list[i].value
   if (list[i].key < k)   // check 2nd half
      return bsearch(list, i+1, end, k)
   else                   // check 1st half
      return bsearch(list, start, i-1, k)
}
```

Usual problem with arrays!

- Unused capacity
- Costly to update and maintain sorted list…many shifts

Dictionary as a Skip List

- Combine storage efficiency of linked list with random access of arrays for search
  - Intuition: Access is not really *random* in binary search!
  - Maintain pointers to midpoints of ranges ☺

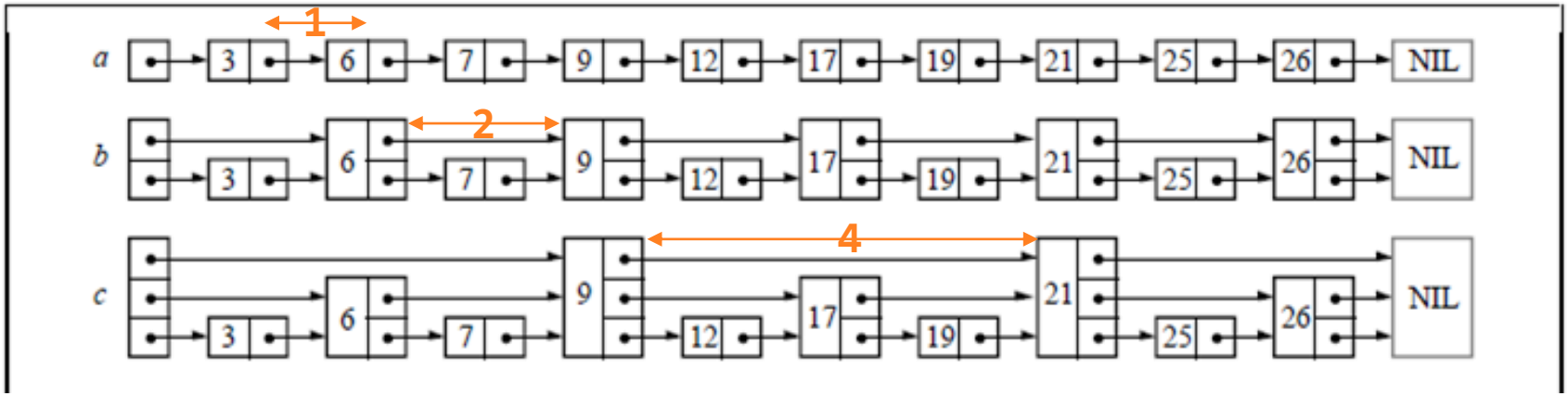


Fig 1. Skip Lists: A Probabilistic Alternative to Balanced Trees, William Pugh, *Communications of the ACM*, 1990

Skip Lists

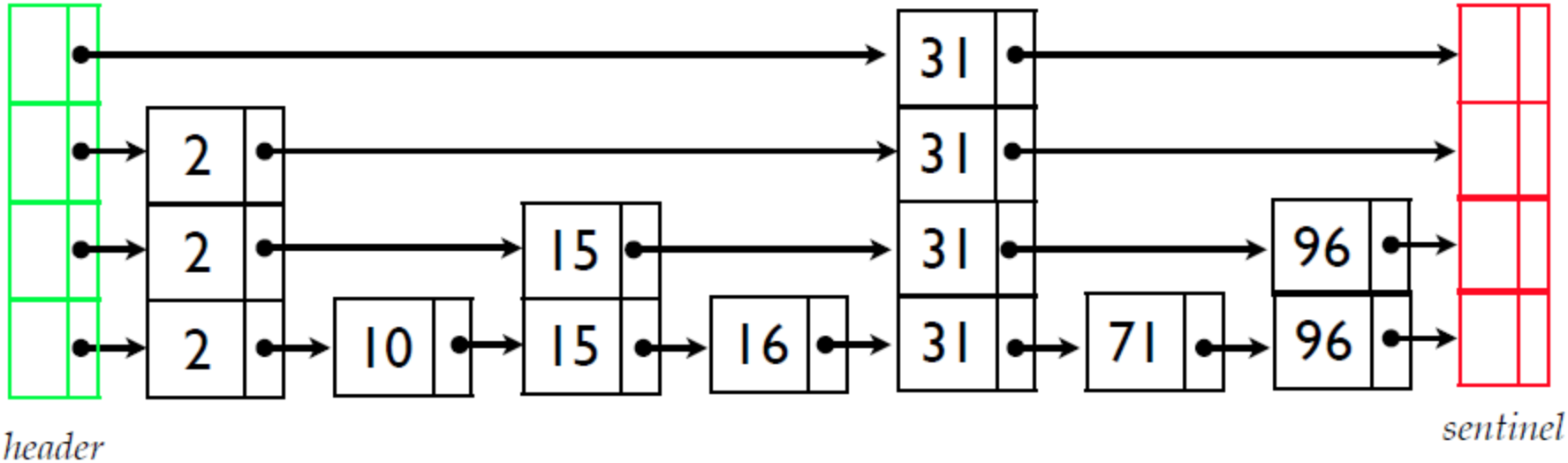
Skip lists are a data structure that can be used in place of balanced trees. Skip lists use probabilistic balancing rather than strictly enforced balancing and as a result the algorithms for insertion and deletion in skip lists are much simpler and significantly faster than equivalent algorithms for balanced trees.

- Expected search time is O(log n)
- *Randomized* data structure:
  - use random coin flips to build the data structure

Skip Lists: A Probabilistic Alternative to Balanced Trees, William Pugh, *Communications of the ACM*, 1990
Skip Lists, CMSC 420: Data Structures, UMaryland Spring 2008

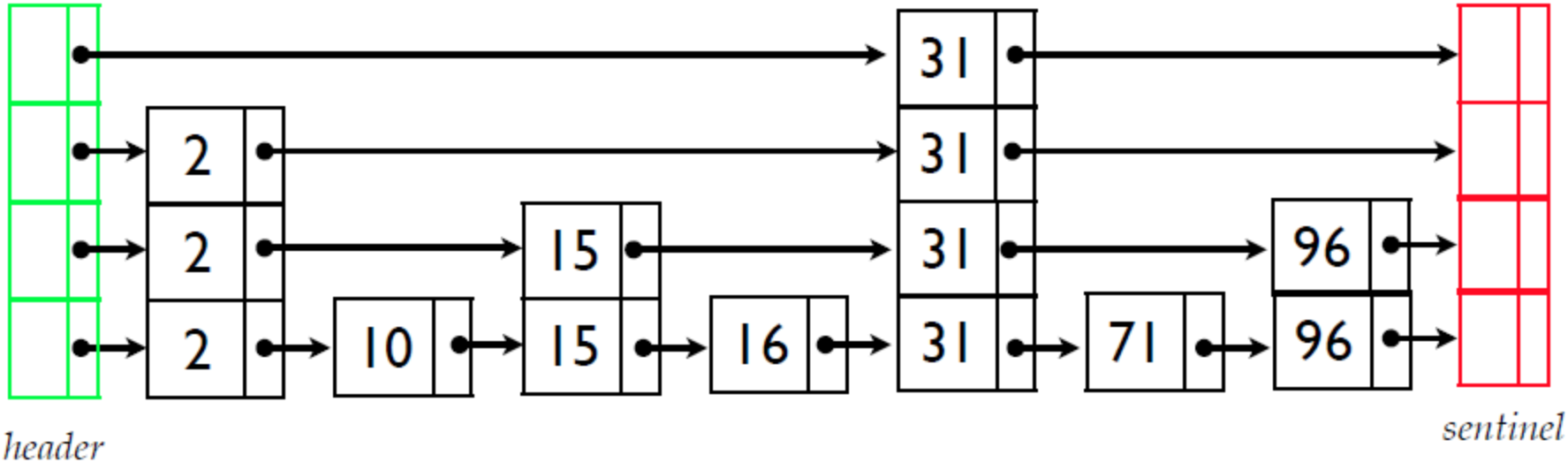
Perfect Skip List

- Keys in sorted order
- O(log n) levels
- Each higher level contains 1/2 the elements of the level below it
- Header & sentinel nodes have every level



Perfect Skip List

- Nodes are of variable size
  - Contain between 1 and O(log n) pointers
  - *Allows you to control storage vs. time complexity (How?)*
- Pointers point to the start of each node
- Called skip lists because higher level lists let you skip over many items



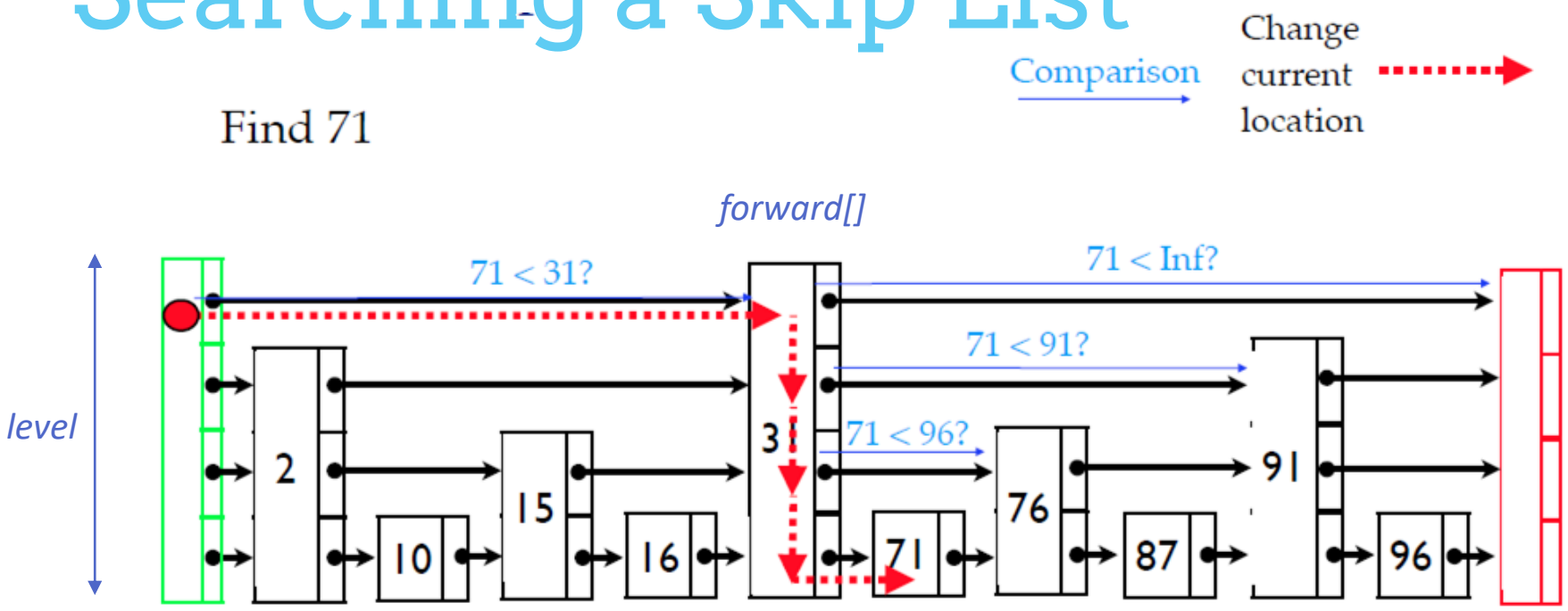
Searching a Skip List

```
Search(list, searchKey)
    x := list→header
    -- loop invariant: x→key < searchKey
    for i := list→level downto 1 do
        while x→forward[i]→key < searchKey do
            x := x→forward[i]
    -- x→key < searchKey ≤ x→forward[1]→key
    x := x→forward[1]
    if x→key = searchKey then return x→value
        else return failure
```

FIGURE 2 - Skip list search algorithm

Skip Lists: A Probabilistic Alternative to Balanced Trees, William Pugh, *Communications of the ACM*, 1990

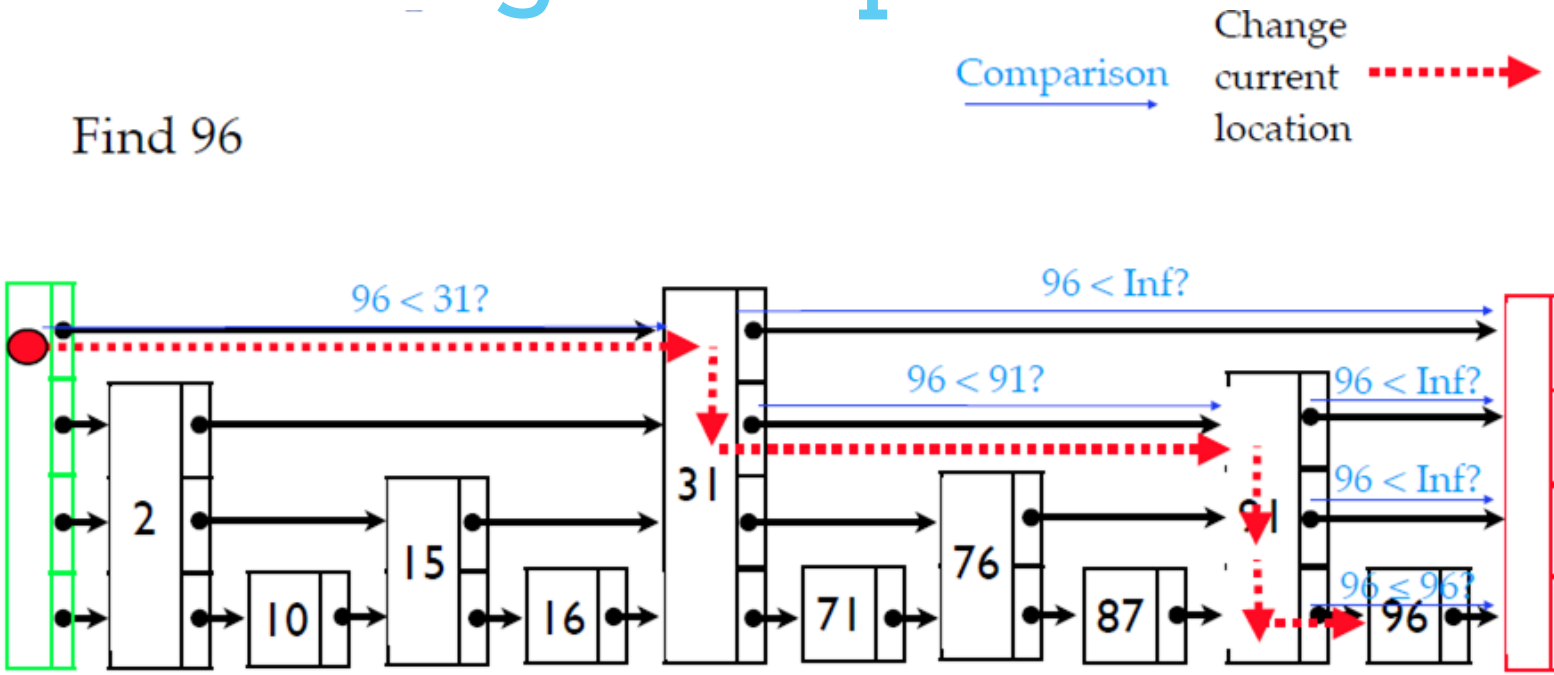
Searching a Skip List



- When searching for *k*:
  - If k = key, done!
  - If k < next key, go down a level
  - If k ≥ next key, go right

To find an item, we scan along the shortest list until we would "pass" the desired item.
At that point, we drop down to a slightly more complete list at one level lower.

Searching a Skip List



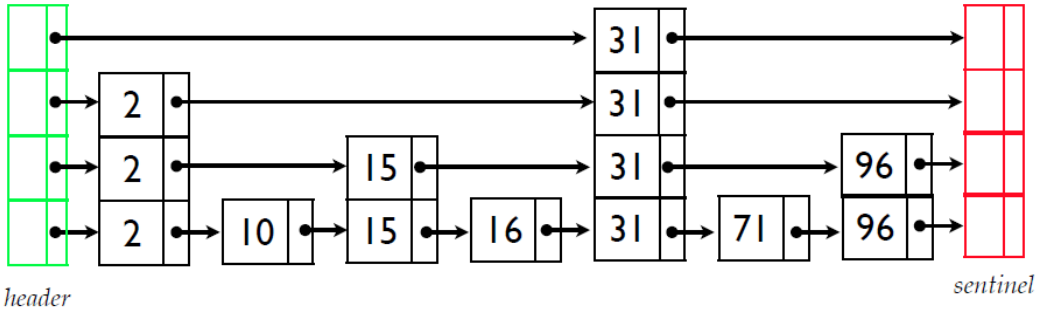
- When searching for *k*:
  - If k = key, done!
  - If k < next key, go down a level
  - If k ≥ next key, go right

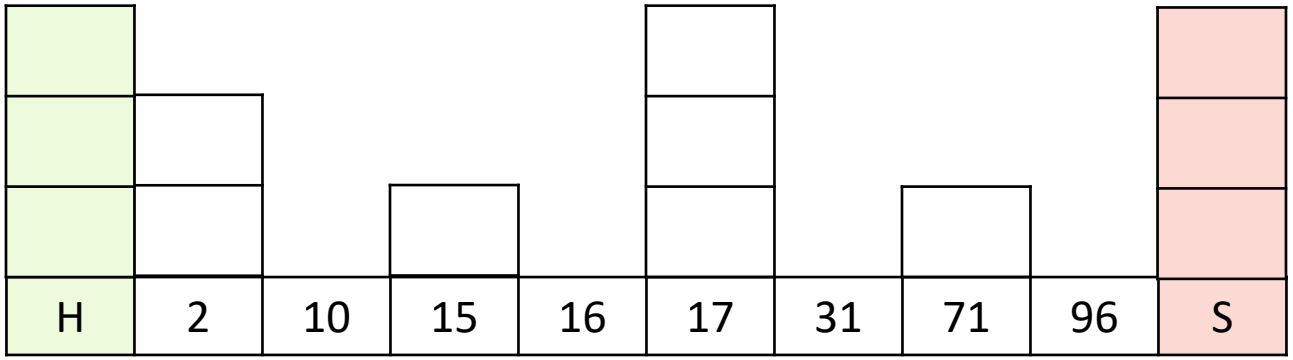
Search Time

- O(log n) levels --- because you cut the # of items in half at each level
- Will visit at most 2 nodes per level:
  - If you visit more, then you could have done it on one level higher up
- Therefore, search time is O(log n).

Insert & Delete into Skip List

- Insert & delete might need to rearrange entire list
- Perfect Skip Lists are ***too*** structured support efficient updates
- E.g. Insert 17





Insert & Delete into Skip List

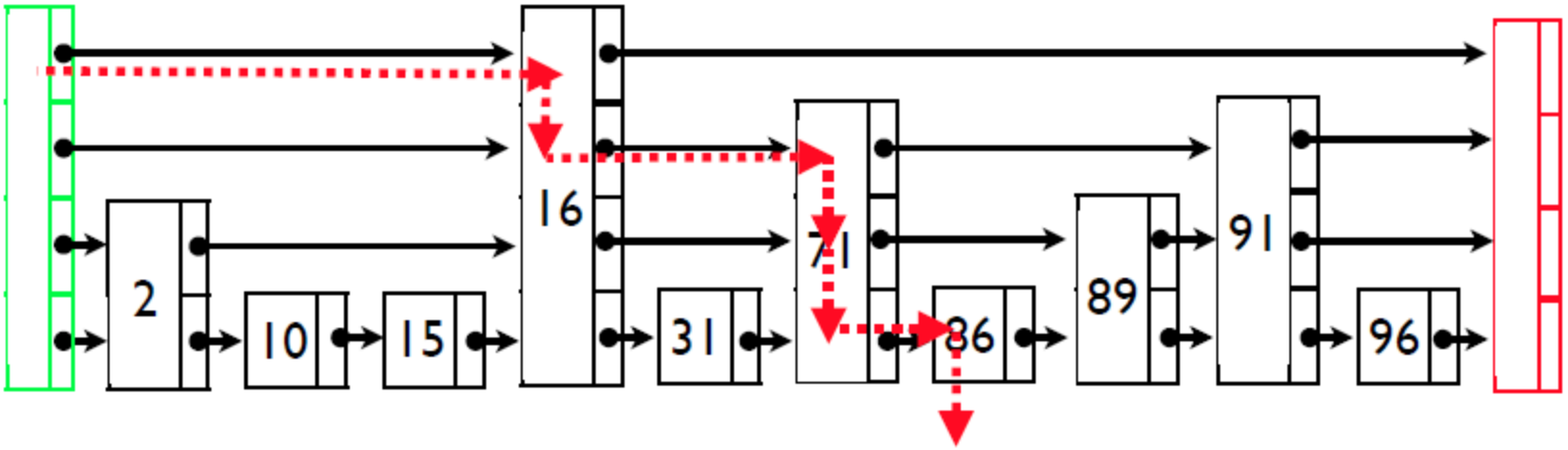
- Idea:
  - Relax the requirement that each level have exactly half the items of the previous level
  - Instead: design structure so that we expect 1/2 the items to be carried up to the next level
- Skip Lists are a randomized data structure: the same sequence of inserts / deletes may produce different structures depending on the outcome of random coin flips.

Randomization

- Allows for some imbalance (like the +1/-1 in AVL trees)
- Expected behavior (over the random choices) remains the same as with perfect skip lists.
- Idea: Each node is promoted to the next higher level with probability 1/2
  - Expect 1/2 the nodes at level 1
  - Expect 1/4 the nodes at level 2
  - ...
- Therefore, expect # of nodes at each level is the same as with perfect skip lists.
- Also: expect the promoted nodes will be well distributed across the list

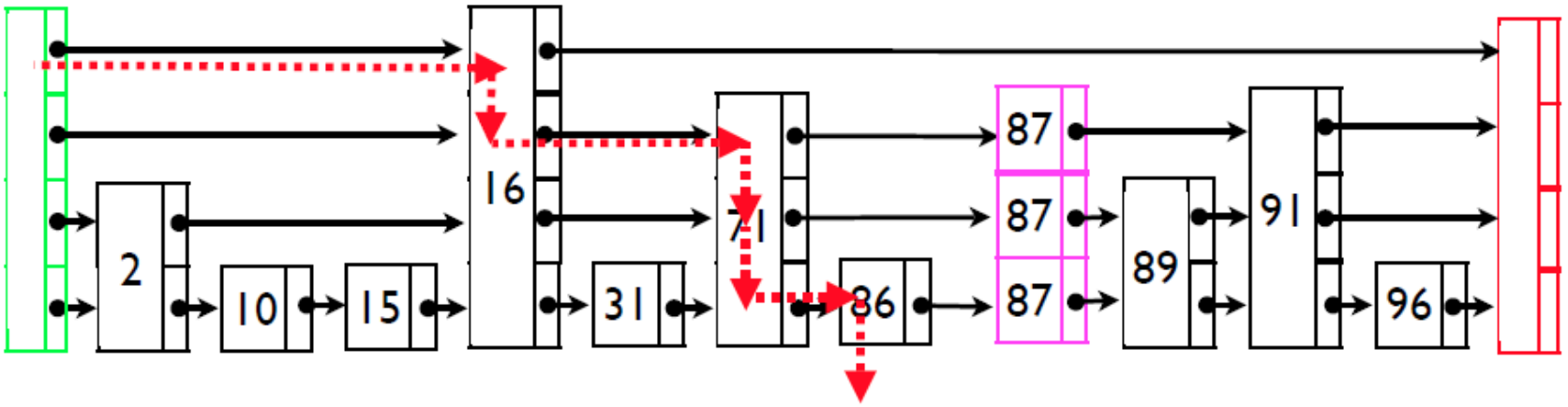
Insertion

Insert 87



Insertion

Insert 87

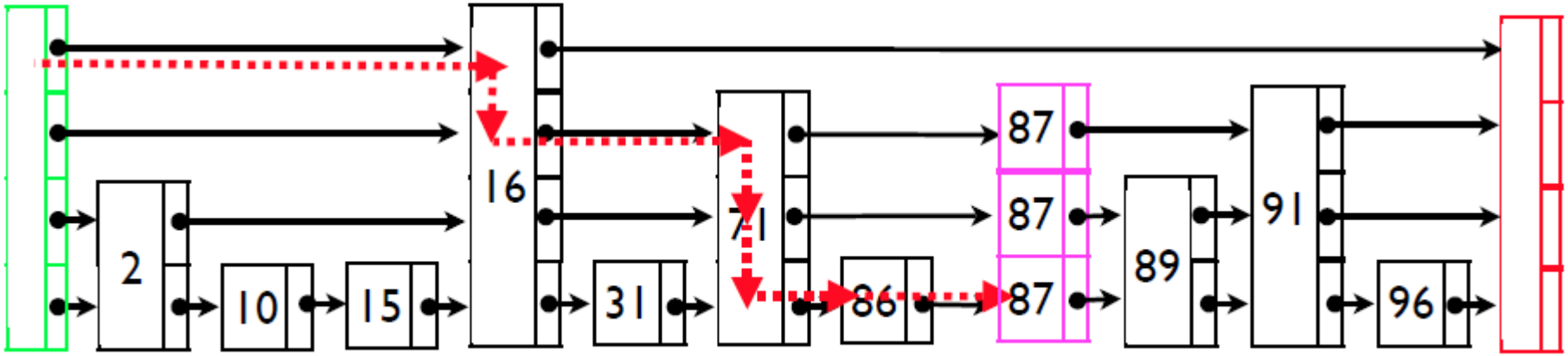



```
Find k
Insert node in level 0
let i = 1
while FLIP() == "heads":
    insert node into level i
    i++
```

Just insertion into a linked list after last visited node in level *i*

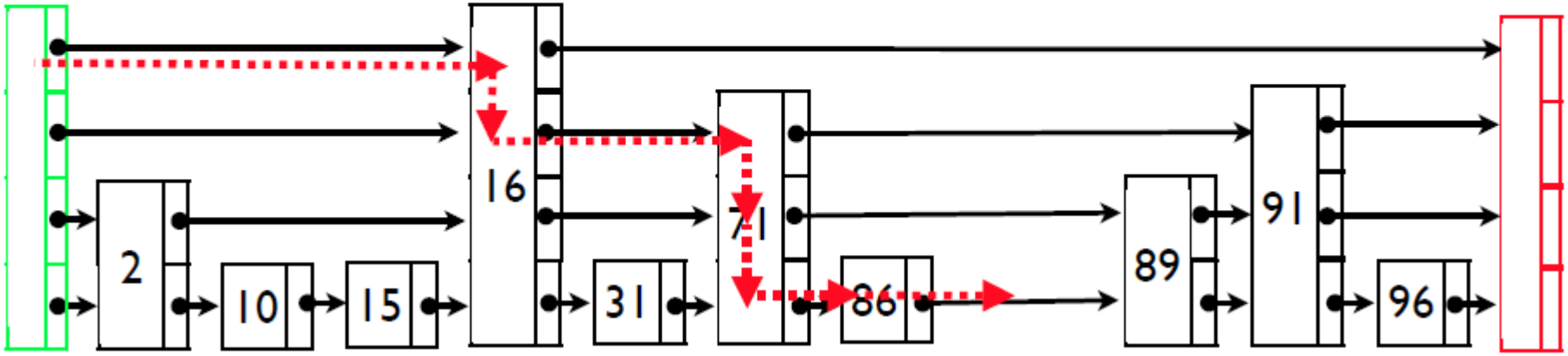
Deletion

Delete 87



Deletion

Delete 87



If we delete the item with max level in the list, we remove that level from head and sentinel, e.g. 16

Tasks

- Solve sanity check problem on CodeChef by **Sep 14**
  - **`https://www.codechef.com/problems/FLOW004`**
- Self study (Sahni Textbook)
  - **Check**: Have you read Chapter 9 "Queues"? Solved problems?
  - **Read**: Chapter 10.0-10.4, Skip Lists from textbook
  - **Read:** Skip Lists: A Probabilistic Alternative to Balanced Trees, William Pugh, *Communications of the ACM*, 1990
  - **Try:** Exercise 3, 6-8 from Chapter 10 of textbook
- Finish Assignment 3 by Wed Sep 28 *(75 points)*

Questions?