CS61B Lecture #31

Today:

- More balanced search structures (*DS(IJ)*, Chapter 9

Coming Up:

- Pseudo-random Numbers (*DS(IJ)*, Chapter 11)

Really Efficient Use of Keys: the Trie

- Haven't said much about cost of comparisons.
- For strings, worst case is length of string.
- Therefore should throw extra factor of key length, L, into
  - $\Theta(M)$ comparisons really means $\Theta(ML)$ operations.
  - So to look for key X, keep looking at same chars of X M
- Can we do better? Can we get search cost to be $O(L)$?

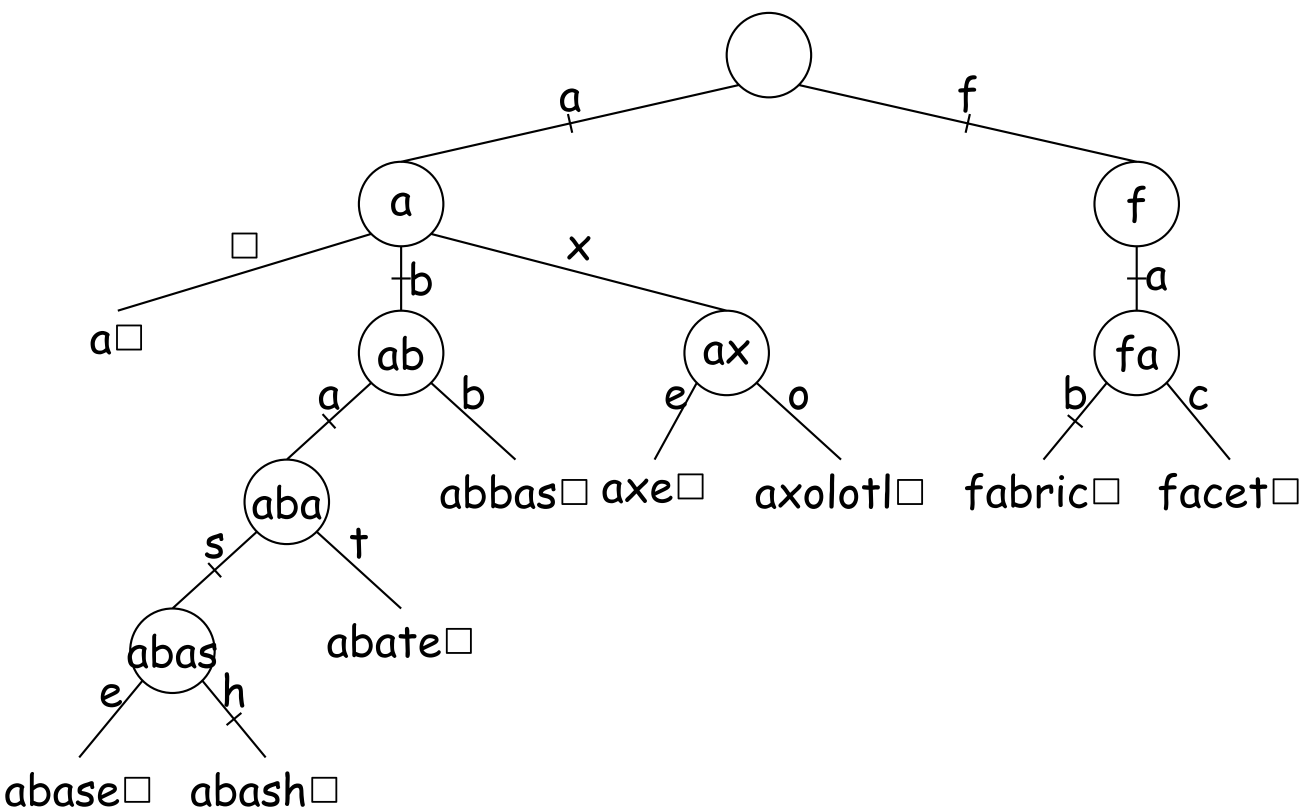
Idea: Make a *multi-way decision tree,* with one decision per ch
of key.

The Trie: Example

- Set of keys

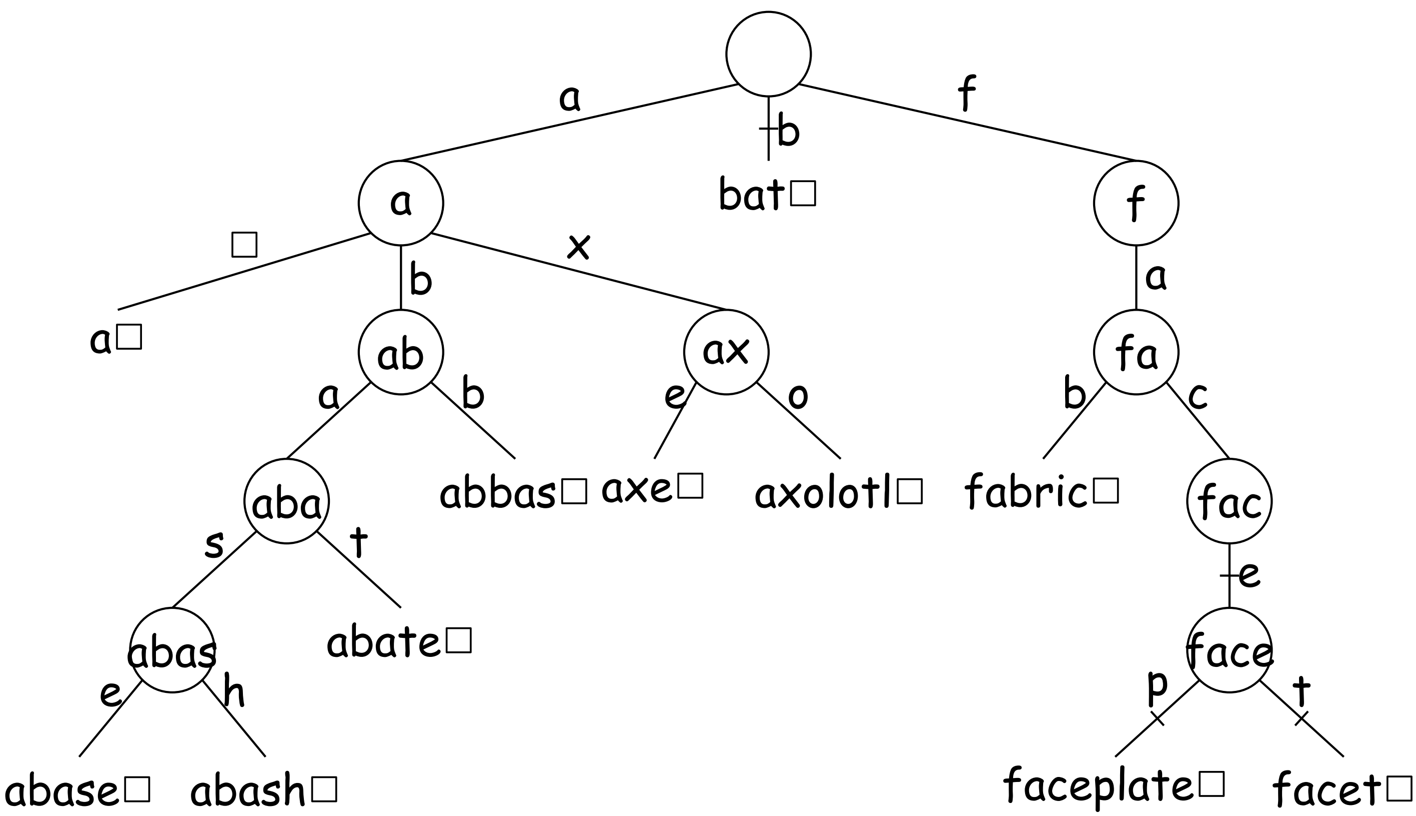
  `{a, abase, abash, abate, abbas, axolotl, axe, fab`

- Ticked lines show paths followed for "abash" and "fabric"
- Each internal node corresponds to a possible prefix.
- Characters in path to node = that prefix.



Adding Item to a Trie

- Result of adding `bat` and `faceplate`.
- New edges ticked.



A Side-Trip: Scrunching

- For speed, obvious implementation for internal nodes is a dexed by character.
- Gives $O(L)$ performance, L length of search key.
- [Looks as if independent of N, number of keys. Is there a dence?]
- **Problem:** arrays are *sparsely populated* by non-null values—w space.

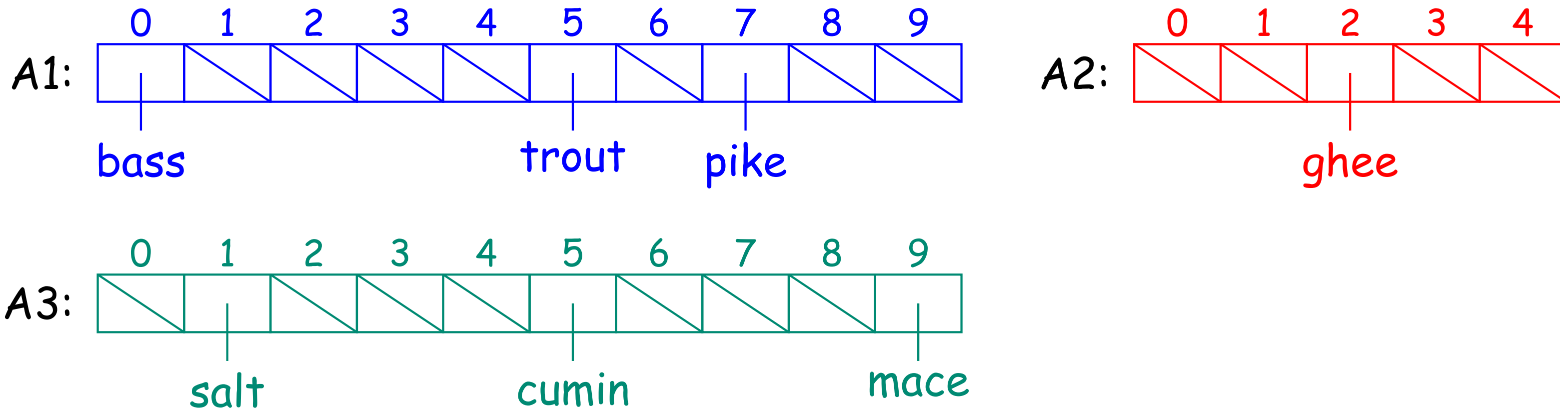
Idea: Put the arrays on top of each other!

- Use null (0, empty) entries of one array to hold non-null elem another.
- Use extra markers to tell which entries belong to which arr

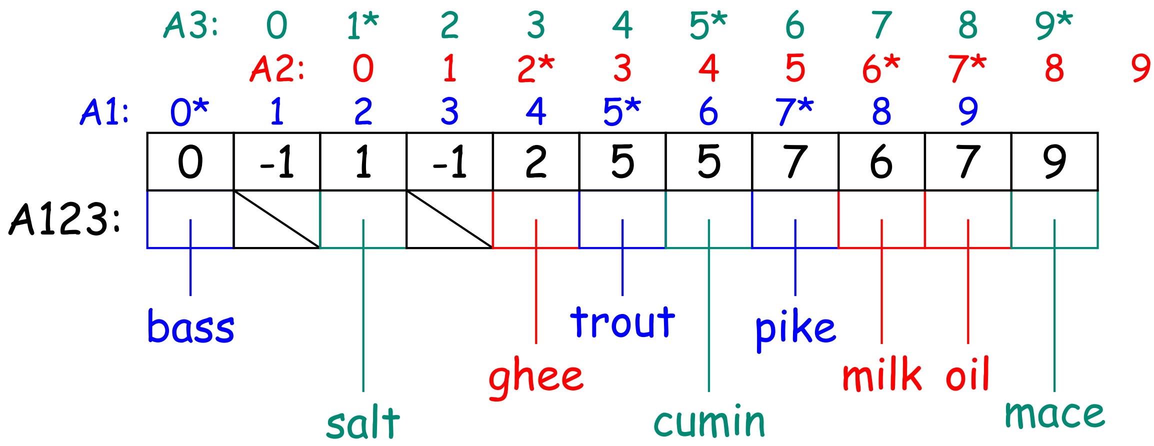
Scrunching Example

Small example: (unrelated to Tries on preceding slides)

- Three leaf arrays, each indexed 0..9



- Now overlay them, but keep track of original index of each i

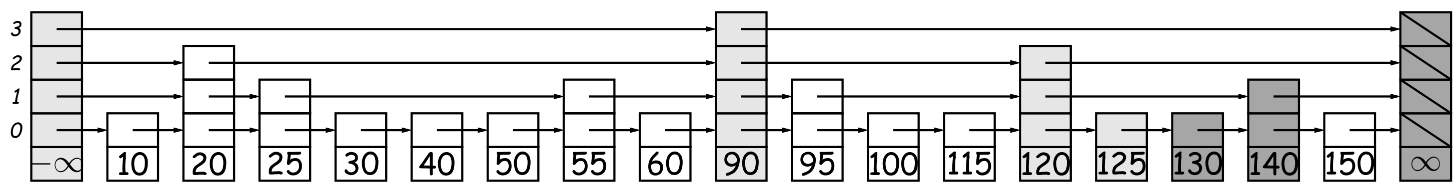


Practicum

- The scrunching idea is cute, but
  - Not so good if we want to expand our trie.
  - A bit complicated.
  - Actually more useful for representing large, sparse, fixe
    with many rows and columns.
- Furthermore, number of children in trie tends to drop dr
  when one gets a few levels down from the root.
- So in practice, might as well use linked lists to represen
  node's children...
- ...but use arrays for the first few levels, which are likely
  more children.

Probabilistic Balancing: Skip Lists

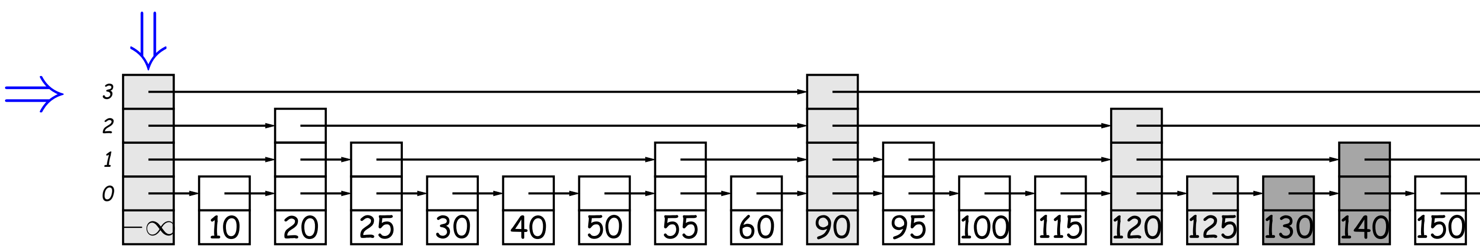
- A *skip list* can be thought of as a kind of n-ary search tree
  we choose to put the keys at "random" heights.
- More often thought of as an ordered list in which one can sk
  segments.
- Typical example:



- To search, start at top layer on left, search until next ste
  overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are loo
  darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there ar
  1/2 as many nodes that are $> k$ high as there are that are k
- Makes searches fast *with high probability.*

Probabilistic Balancing: Skip Lists

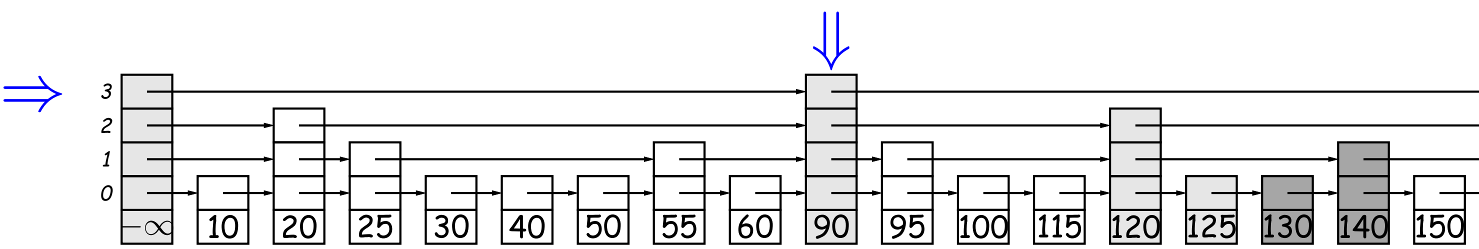
- A *skip list* can be thought of as a kind of n-ary search tree
  we choose to put the keys at “random” heights.
- More often thought of as an ordered list in which one can sk
  segments.
- Typical example:



- To search, start at top layer on left, search until next ste
  overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are lo
  darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there ar
  1/2 as many nodes that are $> k$ high as there are that are k
- Makes searches fast *with high probability.*

Probabilistic Balancing: Skip Lists

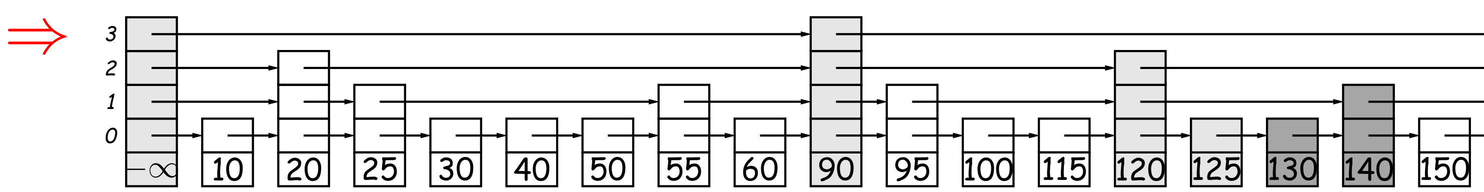
- A *skip list* can be thought of as a kind of n-ary search tree
  we choose to put the keys at "random" heights.
- More often thought of as an ordered list in which one can sk
  segments.
- Typical example:



- To search, start at top layer on left, search until next ste
  overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are loo
  darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there ar
  1/2 as many nodes that are $> k$ high as there are that are k
- Makes searches fast *with high probability.*

Probabilistic Balancing: Skip Lists

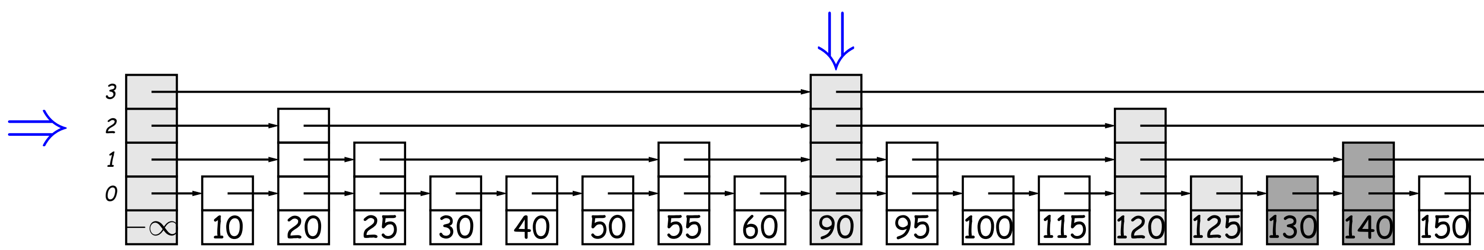
- A *skip list* can be thought of as a kind of n-ary search tree
  we choose to put the keys at "random" heights.
- More often thought of as an ordered list in which one can sk
  segments.
- Typical example:



- To search, start at top layer on left, search until next ste
  overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are lo
  darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there ar
  1/2 as many nodes that are $> k$ high as there are that are k
- Makes searches fast *with high probability.*

Probabilistic Balancing: Skip Lists

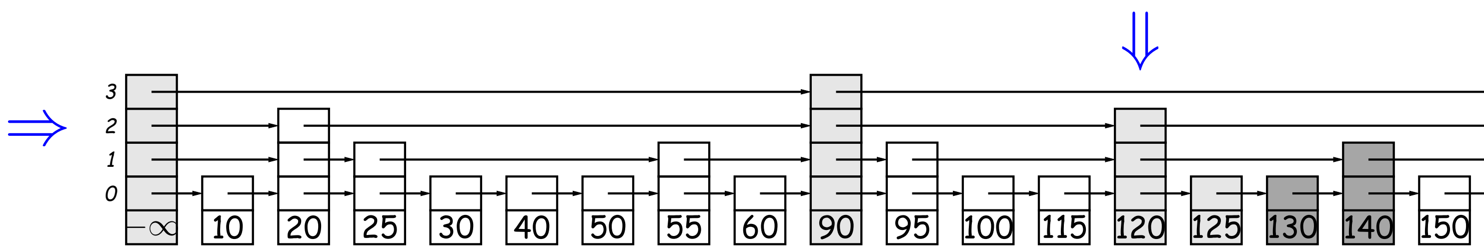
- A *skip list* can be thought of as a kind of n-ary search tree
  we choose to put the keys at "random" heights.
- More often thought of as an ordered list in which one can sk
  segments.
- Typical example:



- To search, start at top layer on left, search until next ste
  overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are lo
  darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there ar
  1/2 as many nodes that are $> k$ high as there are that are k
- Makes searches fast *with high probability.*

Probabilistic Balancing: Skip Lists

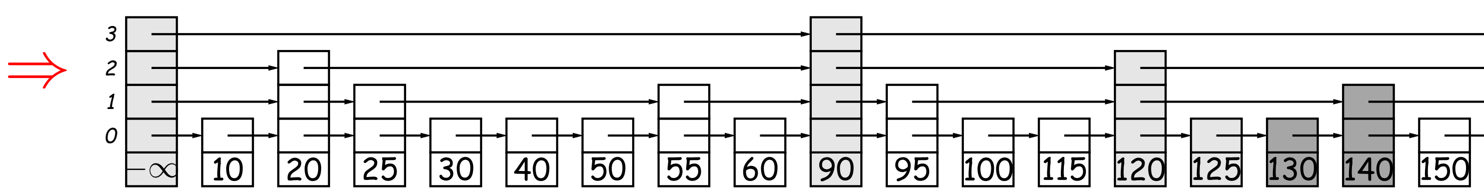
- A *skip list* can be thought of as a kind of n-ary search tree
  we choose to put the keys at "random" heights.
- More often thought of as an ordered list in which one can sk
  segments.
- Typical example:



- To search, start at top layer on left, search until next ste
  overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are loo
  darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there ar
  1/2 as many nodes that are $> k$ high as there are that are k
- Makes searches fast *with high probability.*

Probabilistic Balancing: Skip Lists

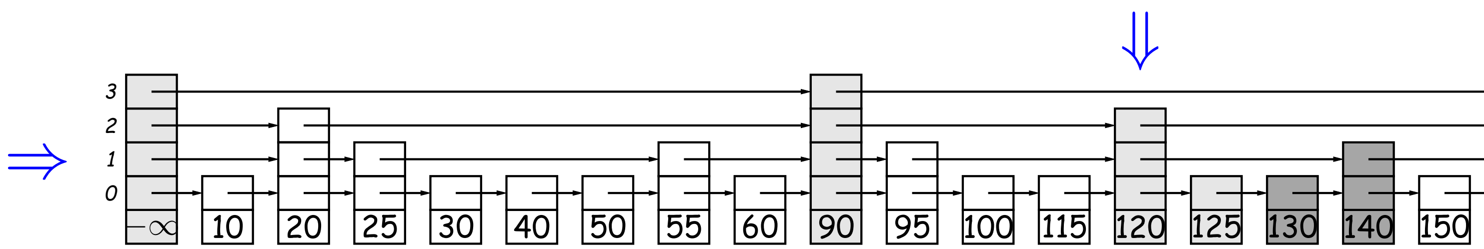
- A *skip list* can be thought of as a kind of n-ary search tree
  we choose to put the keys at "random" heights.
- More often thought of as an ordered list in which one can sk
  segments.
- Typical example:



- To search, start at top layer on left, search until next ste
  overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are lo
  darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there ar
  1/2 as many nodes that are $> k$ high as there are that are k
- Makes searches fast *with high probability.*

Probabilistic Balancing: Skip Lists

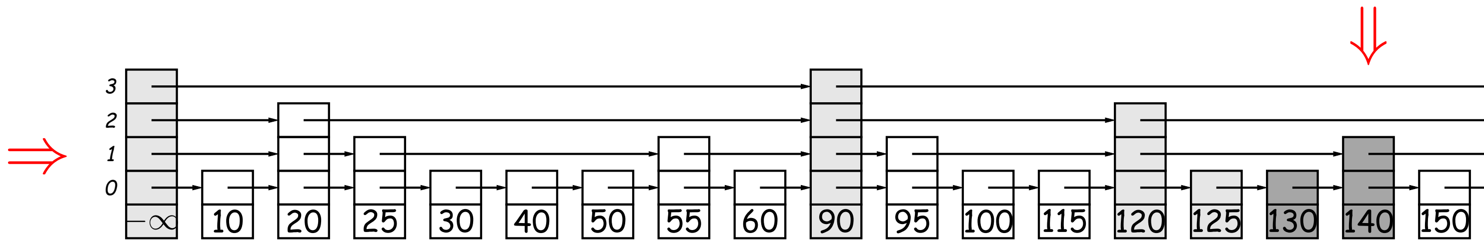
- A *skip list* can be thought of as a kind of n-ary search tree
  we choose to put the keys at "random" heights.
- More often thought of as an ordered list in which one can sk
  segments.
- Typical example:



- To search, start at top layer on left, search until next ste
  overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are lo
  darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there ar
  1/2 as many nodes that are $> k$ high as there are that are k
- Makes searches fast *with high probability.*

Probabilistic Balancing: Skip Lists

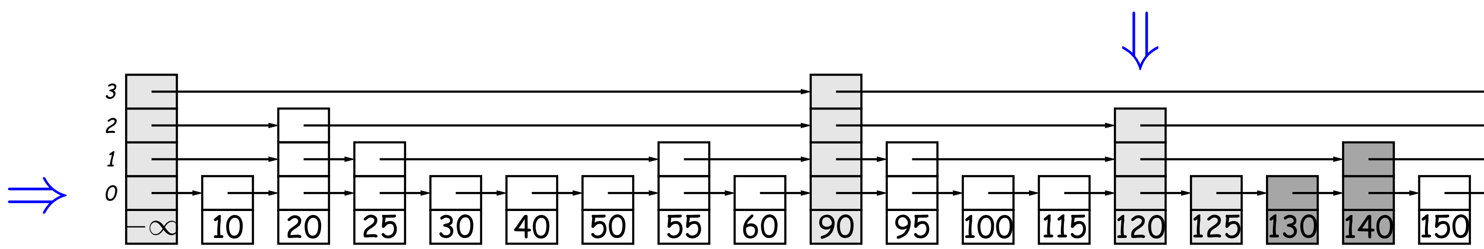
- A *skip list* can be thought of as a kind of n-ary search tree
  we choose to put the keys at "random" heights.
- More often thought of as an ordered list in which one can sk
  segments.
- Typical example:



- To search, start at top layer on left, search until next ste
  overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are loo
  darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there ar
  1/2 as many nodes that are $> k$ high as there are that are k
- Makes searches fast *with high probability.*

Probabilistic Balancing: Skip Lists

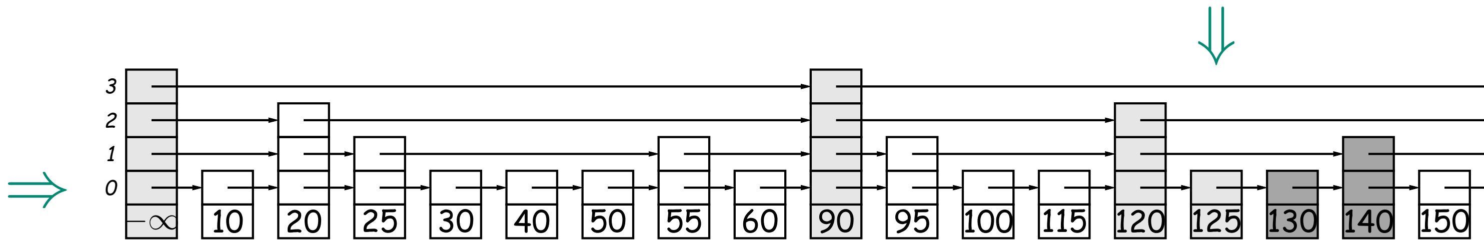
- A *skip list* can be thought of as a kind of n-ary search tree
  we choose to put the keys at "random" heights.
- More often thought of as an ordered list in which one can sk
  segments.
- Typical example:



- To search, start at top layer on left, search until next ste
  overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are lo
  darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there ar
  1/2 as many nodes that are $> k$ high as there are that are k
- Makes searches fast *with high probability.*

Probabilistic Balancing: Skip Lists

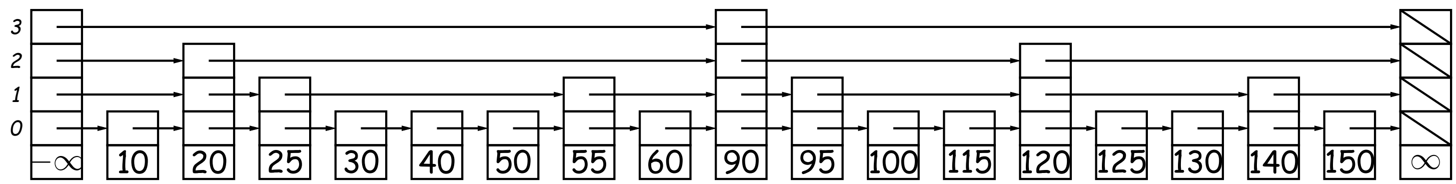
- A *skip list* can be thought of as a kind of n-ary search tree
  we choose to put the keys at "random" heights.
- More often thought of as an ordered list in which one can sk
  segments.
- Typical example:



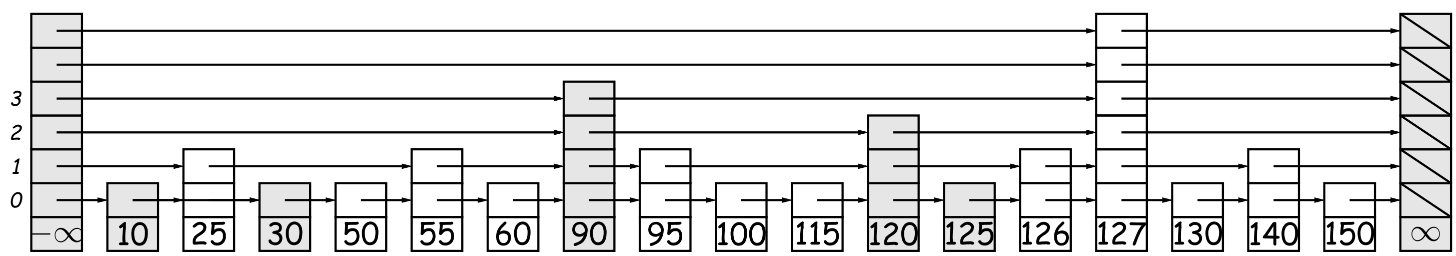
- To search, start at top layer on left, search until next ste
  overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are loo
  darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there ar
  1/2 as many nodes that are $> k$ high as there are that are k
- Makes searches fast *with high probability.*

Example: Adding and deleting

- Starting from initial list:



- In any order, we add 126 and 127 (choosing random heig
them), and remove 20 and 40:



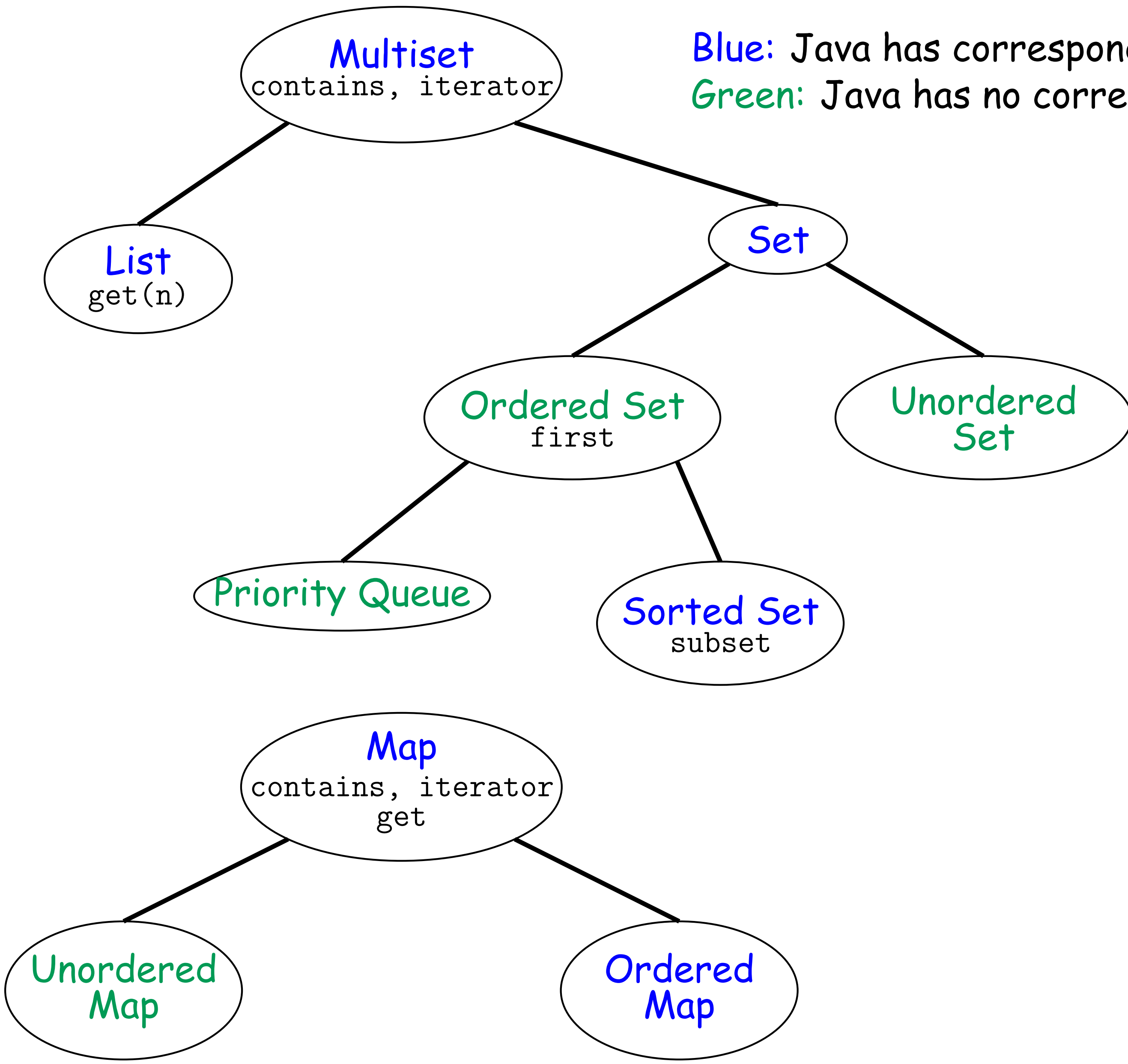
- Shaded nodes here have been modified.

Summary

- Balance in search trees allows us to realize $\Theta(\lg N)$ perform
- B-trees, red-black trees:
  - Give $\Theta(\lg N)$ performance for searches, insertions, deleti
  - B-trees good for external storage. Large nodes minimi I/O operations
- Tries:
  - Give $\Theta(B)$ performance for searches, insertions, and de where B is length of key being processed.
  - But hard to manage space efficiently.
- *Interesting idea:* scrunched arrays share space.
- Skip lists:
  - Give probable $\Theta(\lg N)$ performace for searches, insertio tions
  - Easy to implement.
  - Presented for *interesting ideas:* probabilistic balance, r ized data structures.

Summary of Collection Abstractions

Blue: Java has corresponding interfa
Green: Java has no corresponding in



Data Structures that Implement Abstraction

Multiset

- List: arrays, linked lists, circular buffers
- Set
  - OrderedSet
    - Priority Queue: heaps
    - Sorted Set: binary search trees, red-black trees, [cut off] sorted arrays or linked lists
  - Unordered Set: hash table

Map

- Unordered Map: hash table
- Ordered Map: red-black trees, B-trees, sorted arrays or lin

Corresponding Classes in Java

Multiset (Collection)

- List: ArrayList, LinkedList, Stack, ArrayBlockingQueue, ArrayDeque
- Set
  - OrderedSet
    - Priority Queue: PriorityQueue
    - Sorted Set (SortedSet): TreeSet
  - Unordered Set: HashSet

Map

- Unordered Map: HashMap
- Ordered Map (SortedMap): TreeMap