

CS61B Lecture #31

Today:

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

Coming Up:

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

Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 1

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 costs:
 - $\Theta(M)$ comparisons really means $\Theta(ML)$ operations.
 - So to look for key X , keep looking at same chars of X M times.
- Can we do better? Can we get search cost to be $O(L)$?

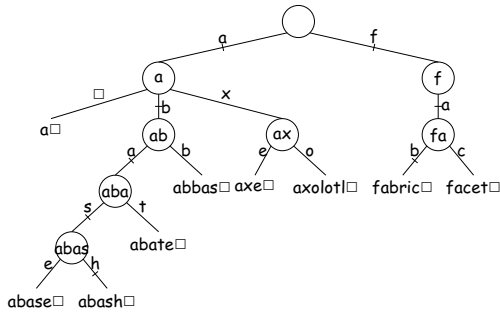
Idea: Make a *multi-way decision tree*, with one decision per character of key.

Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 2

The Trie: Example

- Set of keys
 $\{a, abase, abash, abate, abbas, axolotl, axe, fabric, facet\}$
- Ticked lines show paths followed for "abash" and "fabric"
- Each internal node corresponds to a possible prefix.
- Characters in path to node = that prefix.

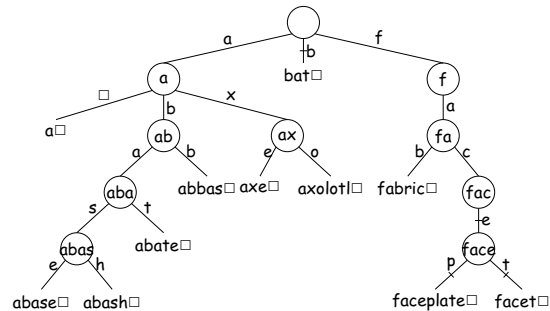


Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 3

Adding Item to a Trie

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



Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 4

A Side-Trip: Scrunching

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

Idea: Put the arrays on top of each other!

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

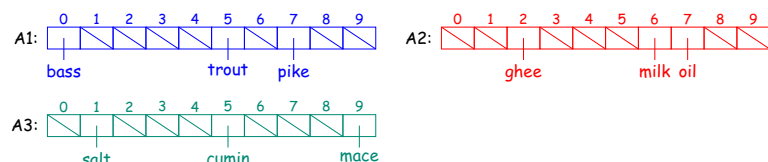
Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 5

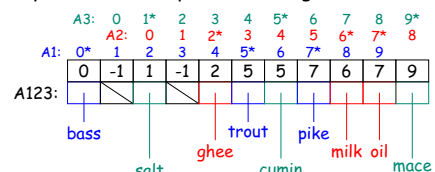
Scrunching Example

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 item:



Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 6

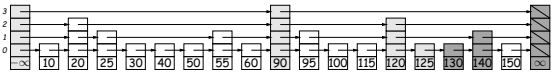
Practicum

- 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, fixed tables with many rows and columns.
- Furthermore, number of children in trie tends to drop drastically when one gets a few levels down from the root.
- So in practice, might as well use linked lists to represent set of node's children...
- ...but use arrays for the first few levels, which are likely to have more children.

Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 7

Probabilistic Balancing: Skip Lists

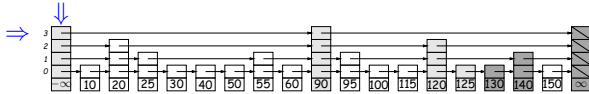
- A **skip list** can be thought of as a kind of n-ary search tree in which we choose to put the keys at "random" heights.
- More often thought of as an ordered list in which one can skip large segments.
- Typical example:
 
- To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there are about 1/2 as many nodes that are $> k$ high as there are that are k high.
- Makes searches fast **with high probability**.

Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 8

Probabilistic Balancing: Skip Lists

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

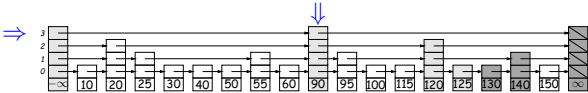


- To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there are about 1/2 as many nodes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

Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 9

Probabilistic Balancing: Skip Lists

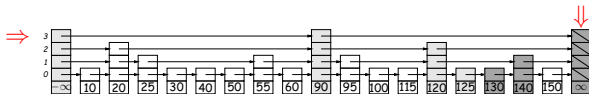
- A **skip list** can be thought of as a kind of n-ary search tree in which we choose to put the keys at "random" heights.
- More often thought of as an ordered list in which one can skip large segments.
- Typical example:
 
- To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there are about 1/2 as many nodes that are $> k$ high as there are that are k high.
- Makes searches fast **with high probability**.

Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 10

Probabilistic Balancing: Skip Lists

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

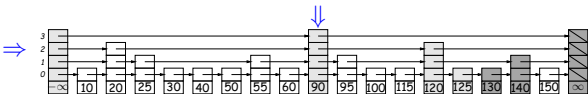


- To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there are about 1/2 as many nodes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

Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 11

Probabilistic Balancing: Skip Lists

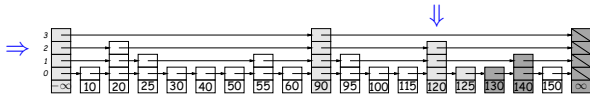
- A **skip list** can be thought of as a kind of n-ary search tree in which we choose to put the keys at "random" heights.
- More often thought of as an ordered list in which one can skip large segments.
- Typical example:
 
- To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there are about 1/2 as many nodes that are $> k$ high as there are that are k high.
- Makes searches fast **with high probability**.

Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 12

Probabilistic Balancing: Skip Lists

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



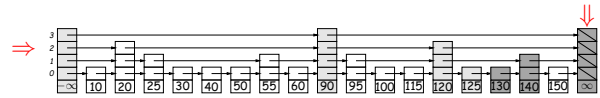
- To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there are about $1/2$ as many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

Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 13

Probabilistic Balancing: Skip Lists

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



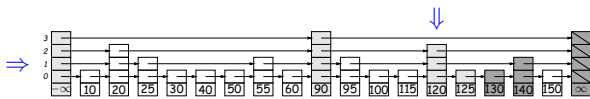
- To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there are about $1/2$ as many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

Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 14

Probabilistic Balancing: Skip Lists

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



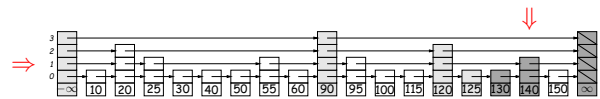
- To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there are about $1/2$ as many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

Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 15

Probabilistic Balancing: Skip Lists

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



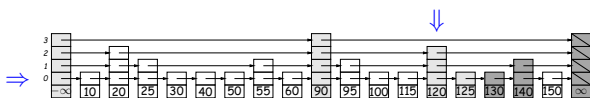
- To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there are about $1/2$ as many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

Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 16

Probabilistic Balancing: Skip Lists

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



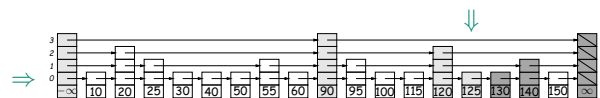
- To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there are about $1/2$ as many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

Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 17

Probabilistic Balancing: Skip Lists

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



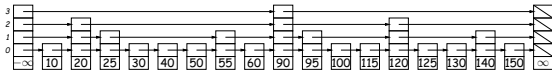
- To search, start at top layer on left, search until next step would overshoot, then go down one layer and repeat.
- In list above, we search for 125 and 127. Gray nodes are looked at; darker gray nodes are overshoots.
- Heights of the nodes were chosen randomly so that there are about $1/2$ as many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

Last modified: Thu Nov 1 19:39:39 2018

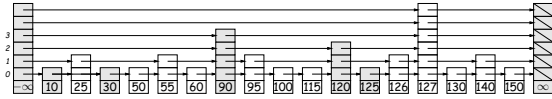
CS61B: Lecture #31 18

Example: Adding and deleting

- Starting from initial list:



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



- Shaded nodes here have been modified.

Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 19

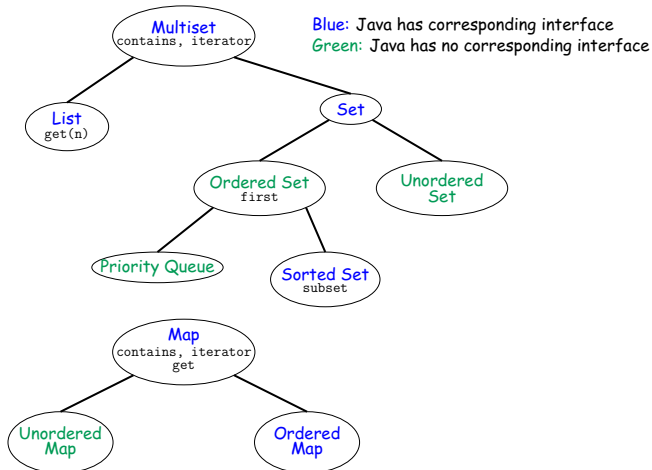
Summary

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

Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 20

Summary of Collection Abstractions



Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 21

Data Structures that Implement Abstractions

Multiset

- List**: arrays, linked lists, circular buffers
- Set**
 - **Ordered Set**
 - * **Priority Queue**: heaps
 - * **Sorted Set**: binary search trees, red-black trees, B-trees, sorted arrays or linked lists
 - **Unordered Set**: hash table

Map

- Unordered Map**: hash table
- Ordered Map**: red-black trees, B-trees, sorted arrays or linked lists

Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 22

Corresponding Classes in Java

Multiset (Collection)

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

Map

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

Last modified: Thu Nov 1 19:39:39 2018

CS61B: Lecture #31 23