Algorithms

Skip Lists

Introduction to Hashing

Review: Red-Black Trees

- *Red-black trees*:
  - Binary search trees augmented with node color
  - Operations designed to guarantee that the height $h = O(\lg n)$
- We described the properties of red-black trees
- We proved that these guarantee $h = O(\lg n)$
- We described operations on red-black trees
  - Only tricky operations: insert, delete
  - Use *rotation* to restructure tree

Review: Skip Lists

- A relatively recent data structure
  - "A probabilistic alternative to balanced trees"
  - A randomized algorithm with benefits of r-b trees
    - O(lg *n*) expected time for Search, Insert
    - O(1) time for Min, Max, Succ, Pred
  - *Much* easier to code than r-b trees
  - Fast!

Review: Linked Lists

- Think about a linked list as a structure for dynamic sets. What is the running time of:
  - **`Min()`** *and* **`Max()`** *?*
  - **`Successor()`** *?*
  - **`Delete()`** *?*
    - *How can we make this O(1)?*
  - **`Predecessor()`** *?*
  - **`Search()`** *?*
  - **`Insert()`** *?*

These all take O(1) time in a doubly linked list. Can you think of a way to do these in O(1) time in a red-black tree?

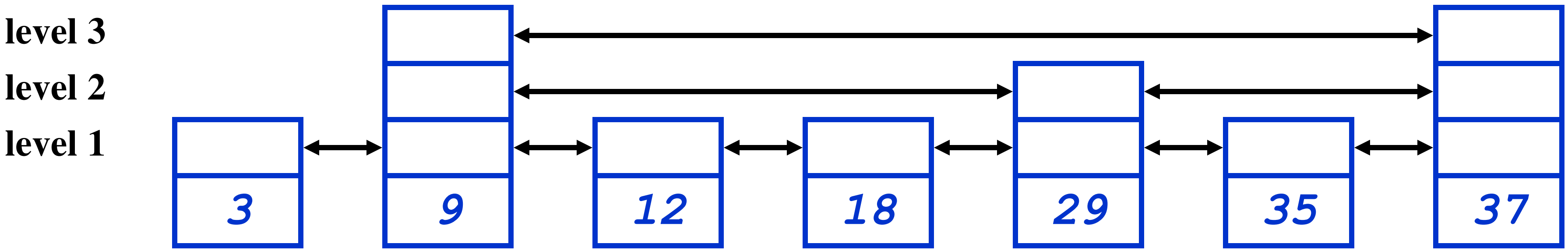
A: *threaded* red-black tree w/ doubly linked list connecting nodes in sorted order

Goal: make these O(lg n) time in a linked-list setting

Idea: keep several levels of linked lists, with high-level lists skipping some low-level items

Skip Lists

- The basic idea:



- Keep a doubly-linked list of elements
  - Min, max, successor, predecessor: O(1) time
  - Delete is O(1) time, Insert is O(1)+Search time
- During insert, add each level-i element to level i+1 with probability p (e.g., p = 1/2 or p = 1/4)

Skip List Search

- To search for an element with a given key:
  - Find location in top list
    - Top list has O(1) elements with high probability
    - Location in this list defines a range of items in next list
  - Drop down a level and recurse
- O(1) time per level on average
- O(lg n) levels with high probability
- Total time: O(lg n)

Skip List Insert

- Skip list insert: analysis
  - Do a search for that key
  - Insert element in bottom-level list
  - With probability *p*, recurse to insert in next level
  - Expected number of lists = $1 + p + p^2 + \ldots$ = *???*

    = $1/(1-p) = O(1)$ if *p* is constant
  - Total time = Search + $O(1) = O(\lg n)$ expected
- Skip list delete: $O(1)$

Skip Lists

- $O(1)$ expected time for most operations
- $O(\lg n)$ expected time for insert
- $O(n^2)$ time worst case (*Why?*)
  - But random, so no particular order of insertion evokes worst-case behavior
- $O(n)$ expected storage requirements (*Why?*)
- Easy to code

Review: Hashing Tables

- Motivation: symbol tables
  - A compiler uses a *symbol table* to relate symbols to associated data
    - Symbols: variable names, procedure names, etc.
    - Associated data: memory location, call graph, etc.
  - For a symbol table (also called a *dictionary*), we care about search, insertion, and deletion
  - We typically don’t care about sorted order

Review: Hash Tables

- More formally:
  - Given a table T and a record x, with key (= symbol) and satellite data, we need to support:
    - Insert (T, x)
    - Delete (T, x)
    - Search(T, x)
  - We want these to be fast, but don't care about sorting the records
- The structure we will use is a *hash table*
  - Supports all the above in O(1) expected time!

Hashing: Keys

- In the following discussions we will consider all keys to be (possibly large) natural numbers
- *How can we convert floats to natural numbers for hashing purposes?*
- *How can we convert ASCII strings to natural numbers for hashing purposes?*

Review: Direct Addressing

- Suppose:
  - The range of keys is 0..m-1
  - Keys are distinct
- The idea:
  - Set up an array T[0..m-1] in which
    - T[i] = x if $x \in T$ and key[x] = i
    - T[i] = NULL otherwise
  - This is called a *direct-address table*
    - Operations take O(1) time!
    - *So what's the problem?*

The Problem With Direct Addressing

- Direct addressing works well when the range *m* of keys is relatively small
- But what if the keys are 32-bit integers?
  - Problem 1: direct-address table will have 2^{32} entries, more than 4 billion
  - Problem 2: even if memory is not an issue, the time to initialize the elements to NULL may be
- Solution: map keys to smaller range 0..*m*-1
- This mapping is called a *hash function*