CS 361, Lecture 25

Jared Saia
University of New Mexico

Outline

- B-Trees
- Skip Lists
- Graph Theory Intro

B-Trees

- B-Trees are balanced search trees designed to work well on disks
- B-Trees are *not* binary trees: each node can have many children
- Each node of a B-Tree potentially contains *several* keys, not just one
- When doing searches, we decide which child link to follow by finding the correct interval of our search key in the key set of the current node.

Disk Accesses

- Consider any search tree
- The number of disk accesses per search will dominate the run time
- Unless the entire tree is in memory, there will usually be a disk access every time an arbitrary node is examined
- The number of disk accesses for most operations on a B-tree is proportional to the height of the B-tree
- I.e. The info on each node of a B-tree can be stored in main memory

B-Tree Properties

The following is true for every node x

- x stores keys, $key_1, \ldots key_l(x)$ in sorted order (nondecreasing)
- x contains pointers, $c_1(x), \ldots, c_{l+1}(x)$ to its children
- Let k_i be any key stored in the subtree rooted at the i-th child of x, then $k_1 \leq key_1(x) \leq k_2 \leq key_2(x) \cdots \leq key_l(x) \leq k_{l+1}$

B-Tree Properties

- All leaves have the same depth
- Lower and upper bounds on the number of keys a node can contain. Given as a function of a fixed integer t
  - If the tree is non-empty, the root must have at least one key, and 2 children
  - Every node other than the root must have at least $(t-1)$ keys, and all internal nodes other than the root must have at least t children.
  - Every node can contain at most $2t-1$ keys, and so any internal node can have at most $2t$ children

Note

- The above properties imply that the height of a B-tree is no more than $\log_t \frac{n+1}{2}$, for $t \geq 2$, where n is the number of keys.
- If we make t, larger, we can save a larger (constant) fraction over RB-trees in the number of nodes examined
- A (2-3-4)-tree is just a B-tree with $t = 2$

In-Class Exercise

We will now show that for any B-Tree with height h and n keys, $h \leq \log_t \frac{n+1}{2}$, where $t \geq 2$.

Consider a B-Tree of height $h > 1$

- Q1: What is the minimum number of nodes at depth 1, 2, and 3
- Q2: What is the minimum number of nodes at depth i?
- Q3: Now give a lowerbound for the total number of *keys* (e.g. $n \geq ???$)
- Q4: Show how to solve for h in this inequality to get an upperbound on h

Splay Trees

- A Splay Tree is a kind of BST where the standard operations run in $O(\log n)$ *amortized* time
- This means that over l operations (e.g. Insert, Lookup, Delete, etc), where l is sufficiently large, the total cost is $O(l * \log n)$
- In other words, the average cost per operation is $O(\log n)$
- However a single operation could still take $O(n)$ time
- In practice, they are very fast

Skip Lists

- Technically, not a BST, but they implement all of the same operations
- Very elegant randomized data structure, simple to code but analysis is subtle
- They guarantee that, with high probability, all the major operations take $O(\log n)$ time
- We'll discuss them more next class

High Level Analysis

Comparison of various BSTs

- RB-Trees: $+$ guarantee $O(\log n)$ time for each operation, easy to augment, $-$ high constants
- AVL-Trees: $+$ guarantee $O(\log n)$ time for each operation, $-$ high constants
- B-Trees: $+$ guarantee $O(\log n)$ time for each operation, works well for trees that won't fit in memory, $-$ inserts and deletes are more complicated
- Splay Tress: $+$ small constants, $-$ amortized guarantees only
- Skip Lists: $+$ easy to implement, $-$ runtime guarantees are probabilistic only

Which Data Structure to use?

- Splay trees work very well in practice, the "hidden constants" are small
- Unfortunately, they can not guarantee that *every* operation takes $O(\log n)$
- When this guarantee is required, B-Trees are best when the entire tree will not be stored in memory
- If the entire tree will be stored in memory, RB-Trees, AVL-Trees, and Skip Lists are good

Skip List

- Technically, not a BST, but they implement all of the same operations
- Very elegant randomized data structure, simple to code but analysis is subtle
- They guarantee that, with high probability, all the major operations take $O(\log n)$ time

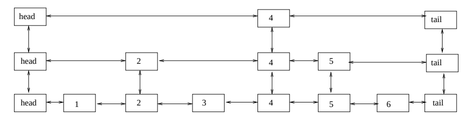
Skip List

- A skip list is basically a collection of doubly-linked lists, $L_1, L_2, \ldots, L_x$, for some integer x
- Each list has a special head and tail node, the keys of these nodes are assumed to be $-$MAXINT and $+$MAXINT respectively
- The keys in each list are in sorted order (non-decreasing)

Skip List

- Every key is in the list L_1.
- For all $i > 2$, if a key x is in the list L_i, it is also in L_{i-1}. Further there are up and down pointers between the x in L_i and the x in L_{i-1}.
- All the head(tail) nodes from neighboring lists are interconnected

Example



Search

```
Search(k){
  pLeft = L_x.head;
  for (i=x;i>=0;i--){
    Search from pLeft in L_i to get the rightmost elem, r,
      with value <= k;
    pLeft = pointer to r in L_(i-1);
  }
  if (pLeft==k)
    return pLeft
  else
    return nil
  }
}
```


Insert

p is a constant between 0 and 1, typically $p = 1/2$

```
Insert(k){
First call Search(k), let pLeft be the leftmost elem <= k in L_1
Insert k in L_1, to the right of pLeft
i = 2;
while (rand()<p){
  insert k in the appropriate place in L_i;
}
```

Deletion

- Deletion is very simple
- First do a search for the key to be deleted
- Then delete that key from all the lists it appears in from the bottom up, making sure to "zip up" the lists after the deletion

In-Class Exercise

A trick for computing expectations of discrete positive random variables:

- Let X be a discrete r.v., that takes on values from 1 to n

$$E(X) = \sum_{i=1}^{n} P(X \geq i)$$

Why?

$$\sum_{i=1}^{n} P(X \geq i) = 1 * P(X = 1) + 2 * P(X = 2) + \ldots \quad (1)$$

$$= E(X) \quad (2)$$

In-Class Exercise

Q: How much memory do we expect a skip list to use up?

- Let X_i be the number of lists elem i is inserted in
- Q: What is $P(X_i \geq 1)$, $P(X_i \geq 2)$, $P(X_i \geq 3)$?
- Q: What is $P(X_i \geq k)$ for general k?
- Q: What is $E(X_i)$?
- Q: Let $X = \sum_{i=1}^{n} X_i$. What is $E(X)$?

Height of Skip List

- Assume there are n nodes in the list
- Q: What is the probability that a particular key i achieves height $k \log n$ for some constant k?
- A: If $p = 1/2$, $P(X_i \geq k \log n) = \frac{1}{n^k}$

Height of Skip List

- Q: What is the probability that *any* of the nodes achieve height higher than $k \log n$?
- A: We want

$$P(X_1 \geq k \log n \text{ or } X_2 \geq k \log n \text{ or } \ldots \text{ or } X_n \geq k \log n)$$

- By a Union Bound, this probability is no more than

$$P(X_1 \geq k \log n) + P(X_2 \geq k \log n) + \cdots + P(X_n \geq k \log n)$$

- Which equals $\frac{n}{n^k} = n^{1-k}$

Height of Skip List

- If we choose k to be, say 10, this probability gets very small as n gets large
- In particular, the probability of having a skip list of size exceeding $k \log n$ is $o(1)$
- So we say that the height of the skip list is $O(\log n)$ *with high probability*

Search Time

- Note that the expected number of "siblings" of a node, x, at any level i is 2
- Why? Because for a node to be a sibling of x at level i, it must have failed to advance to the next level
- The first node that advances to the next level ends the possibility of further siblings.
- This is the same as asking expected number of times we need to flip a coin to get a heads - the answer is 2

Search Time

- The expected number of "siblings" of a node, x, at any level i is 2
- The number of levels is $O(\log n)$ with high probability
- From these two facts, we can argue that the expected search time is $O(\log n)$
- (Warning: The argument is not as simple as multiplying these two values. We can't do this since the two random variables are not independent.)