CS 361, Lecture 27

Jared Saia
University of New Mexico

Outline

- Skip List Wrapup
- Master Theorem

Height of Skip List

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

Height of Skip List

- Q: What is the probability that *any* of the nodes exceed height $k \log n$?
- 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.

Flipping to get Heads

- How many times in expectation do we need to flip a coin to get heads, if the coin is heads with probability p?
- Let X be a r.v. giving the number of times needed to flip to get heads, then $E(X)$ is the expected number of times needed to flip to get heads
- Then $E(X) = 1 + (1-p)E(X)$ since we take 1 flip, plus in the case where the coin is tails (which happens with probability $(1-p)$), we then take "the expected number of times needed to flip to get heads" (i.e. we're no better off than when we started)
- Solving for $E(X)$ gives $E(X) = 1/p$
- If $p = 1/2$, then $E(X) = 2$

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.)

Master Theorem

- Divide and conquer algorithms often give us running-time recurrences of the form

$$T(n) = a\,T(n/b) + f(n) \tag{1}$$

- Where a and b are constants and $f(n)$ is some other function.
- The so-called 'Master Theorem' gives us a general method for solving such recurrences $f(n)$ is a simple polynomial.

Master Theorem

- Unfortunately, the Master Theorem doesn't work for all functions $f(n)$
- Further many useful recurrences don't look like $T(n)$
- However, the theorem allows for fast solution of recurrences when it applies

Recursion Tree Review

- Master Theorem is just a special case of the use of recursion trees
- Consider equation $T(n) = a\,T(n/b) + f(n)$
- We start by drawing a recursion tree

Recursion Tree Review

- The root is a box containing the value $f(n)$
- It has a children, each of which is the root of a recursion tree for $T(n/b)$
- Each of these nodes has a children, etc., etc.

Another View

- Equivalently, a recursion tree is a complete a-ary tree where each node at depth i contains the value $f(n/b^i)$.
- The tree stops when we get to the base case for the recurrence
- We'll assume $T(1) = f(1)$ is the base case
- Then there are $\log_b n$ levels to the recursion tree

Recursion Tree

- For this tree, $T(n)$ is just the sum of all values stored in all levels of the tree

  $T(n) = f(n)+a\,f(n/b)+a^2\,f(n/b^2)+\cdots+a^i\,f(n/b^i)+\cdots+a^L\,f(n/b^L)$
- Where $L = \log_b n$ is the depth of the tree
- Since $f(1) = \Theta(1)$, the last term of this summation is $\Theta(a^L) = \Theta(a^{\log_b n}) = \Theta(n^{\log_b a})$

A "Log Fact" Aside

- It's not hard to see that $a^{\log_b n} = n^{\log_b a}$

$$a^{\log_b n} = n^{\log_b a} \qquad (2)$$
$$a^{\log_b n} = a^{\log_a n * \log_b a} \qquad (3)$$
$$\log_b n = \log_a n * \log_b a \qquad (4)$$

- We get to the last eqn by taking $\log_a$ of both sides
- The last eqn is true by our third basic log fact

Master Theorem

- We can now state the Master Theorem
- We will state it in a way slightly different from the book
- Note: The Master Method is just a "short cut" for the recursion tree method. It is less powerful than recursion trees.

Master Method

The recurrence $T(n) = aT(n/b) + f(n)$ can be solved as follows:

- If $a\,f(n/b) \le f(n)/K$ for some constant $K > 1$, then $T(n) = \Theta(f(n))$.
- If $a\,f(n/b) \ge K\,f(n)$ for some constant $K > 1$, then $T(n) = \Theta(n^{\log_b a})$.
- If $a\,f(n/b) = f(n)$, then $T(n) = \Theta(f(n)\log_b n)$.

Proof

- If $f(n)$ is a *constant factor larger* than $a\,f(b/n)$, then the sum is a descending geometric series. The sum of any geometric series is a constant times its largest term. In this case, the largest term is the first term $f(n)$.
- If $f(n)$ is a *constant factor smaller* than $a\,f(b/n)$, then the sum is an ascending geometric series. The sum of any geometric series is a constant times its largest term. In this case, this is the last term, which by our earlier argument is $\Theta(n^{\log_b a})$.
- Finally, if $a\,f(b/n) = f(n)$, then each of the L terms in the summation is equal to $f(n)$.

Example

- $T(n) = T(3n/4) + n$
- If we write this as $T(n) = aT(n/b) + f(n)$, then $a = 1$,$b = 4/3$,$f(n) = n$
- Here $a\,f(n/b) = 3n/4$ is smaller than $f(n) = n$ by a factor of $4/3$, so $T(n) = \Theta(n)$

Example

- **Karatsuba's multiplication algorithm: $T(n) = 3T(n/2) + n$**
- If we write this as $T(n) = aT(n/b) + f(n)$, then $a = 3$,$b = 2$,$f(n) = n$
- Here $a\,f(n/b) = 3n/2$ is bigger than $f(n) = n$ by a factor of $3/2$, so $T(n) = \Theta(n^{\log_2(3/2)})$

Example

- **Mergesort: $T(n) = 2T(n/2) + n$**
- If we write this as $T(n) = aT(n/b) + f(n)$, then $a = 2$,$b = 2$,$f(n) = n$
- Here $a\,f(n/b) = f(n)$, so $T(n) = \Theta(n \log n)$

Example

- $T(n) = T(n/2) + n \log n$
- If we write this as $T(n) = aT(n/b) + f(n)$, then $a = 1$,$b = 2$,$f(n) = n \log n$
- Here $a\,f(n/b) = n/2 \log n/2$ is smaller than $f(n) = n \log n$ by a constant factor, so $T(n) = \Theta(n \log n)$

In-Class Exercise

- Consider the recurrence: $\boldsymbol{T(n) = 4T(n/2) + n \lg n}$
- Q: What is $f(n)$ and $a\,f(n/b)$?
- Q: Which of the three cases does the recurrence fall under (when n is large)?
- Q: What is the solution to this recurrence?

In-Class Exercise

- Consider the recurrence: $\boldsymbol{T(n) = 2T(n/4) + n \lg n}$
- Q: What is $f(n)$ and $a\,f(n/b)$?
- Q: Which of the three cases does the recurrence fall under (when n is large)?
- Q: What is the solution to this recurrence?