Homework

Any questions on the homework?

Outline

- Skip Lists

Administrative

- This week and next, you can get one extra participation check by going to section and informing Nate that you’re there and want a check.
- Sections are Thursday 5:30-6:20 DSH 134 and Friday 1:00-1:50 TAPY 218
- Good chance to get info on hw, projects and to review material for final
Project

- Project will be due May 6th in class
- Late projects will *not* be accepted
- You can get partial credit for an unfinished project turned in on time but will get no credit for a finished project turned in late

Final

- Final will be May 11th 5:30-7:30pm in our regular classroom
- Closed book, but two pieces of paper and calculators are allowed

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

- 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 $-\text{MAXNUM}$ and $+\text{MAXNUM}$ respectively
- The keys in each list are in sorted order (non-decreasing)
Skip List

- Every node is stored in the bottom list
- For each node in the bottom list, we flip a coin over and over until we get tails. For each heads, we make a duplicate of the node.
- The duplicates are stacked up in levels and the nodes on each level are strung together in sorted linked lists
- Each node \( v \) stores a search key (\( \text{key}(v) \)), a pointer to its next lower copy (\( \text{down}(v) \)), and a pointer to the next node in its level (\( \text{right}(v) \)).

**Example**

```

```

Search

- To do a search for a key, \( x \), we start at the leftmost node \( L \) in the highest level
- We then scan through each level as far as we can without passing the target value \( x \) and then proceed down to the next level
- The search ends either when we find the key \( x \) or fail to find \( x \) on the lowest level

```
```
Insert

$p$ is a constant between 0 and 1, typically $p = 1/2$, let rand() return a random value between 0 and 1

```cpp
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

Analysis

- Intuitively, each level of the skip list has about half the number of nodes of the previous level, so we expect the total number of levels to be about $O(\log n)$
- Similarly, each time we add another level, we cut the search time in half except for a constant overhead
- So after $O(\log n)$ levels, we would expect a search time of $O(\log n)$
- We will now formalize these two intuitive observations
• For some key, $i$, let $X_i$ be the maximum height of $i$ in the skip list.
• Q: What is the probability that $X_i \geq 2\log n$?
• A: If $p = 1/2$, we have:

$$P(X_i \geq 2\log n) = \left(\frac{1}{2}\right)^{2\log n} = \frac{1}{(2\log n)^2} = \frac{1}{n^2}$$

• Thus the probability that a particular key $i$ achieves height $2\log n$ is $\frac{1}{n^2}$

• This probability gets small as $n$ gets large
• In particular, the probability of having a skip list of size exceeding $2\log n$ is $o(1)$
• If an event occurs with probability $1 - o(1)$, we say that it occurs with high probability
• Key Point: The height of a skip list is $O(\log n)$ with high probability.

Q: What is the probability that any key achieves height $2\log n$?
A: We want

$$P(X_1 \geq 2\log n \text{ or } X_2 \geq 2\log n \text{ or } \ldots \text{ or } X_n \geq 2\log n)$$

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:

$$\sum_{i=1}^{n} \frac{1}{n^2} = \frac{n}{n^2} = \frac{1}{n}$$

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?

\[ \sum_{i=1}^{n} P(X \geq i) = P(X = 1) + P(X = 2) + P(X = 3) + \ldots \]
\[ \quad + P(X = 2) + P(X = 3) + P(X = 4) + \ldots \]
\[ \quad + P(X = 3) + P(X = 4) + P(X = 5) + \ldots \]
\[ \quad + \ldots \]
\[ = 1 \cdot P(X = 1) + 2 \cdot P(X = 2) + 3 \cdot P(X = 3) + \ldots \]
\[ = E(X) \]

In-Class Exercise

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

- Let \( X_i \) be the number of lists that element \( i \) 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 \( E(X) \)?

Search Time

- It’s easier to analyze the search time if we imagine running the search backwards.
- Imagine that we start at the found node \( v \) in the bottommost list and we trace the path backwards to the top leftmost sentinel, \( L \).
- This will give us the length of the search path from \( L \) to \( v \) which is the time required to do the search.

Backwards Search

\[
\text{SLFback}(v)\{
\quad \text{while } (v! = L)\{
\quad \quad \text{if } \text{(Up}(v)! = \text{NIL})
\quad \quad \quad v = \text{Up}(v);
\quad \quad \text{else}
\quad \quad \quad v = \text{Left}(v);
\quad \}
\}
\]
Backward Search

- For every node $v$ in the skip list $\text{Up}(v)$ exists with probability $1/2$. So for purposes of analysis, SLFBack is the same as the following algorithm:

  ```java
  FlipWalk(v){
      while (v != L){
          if (COINFLIP == HEADS)
              v = Up(v);
          else
              v = Left(v);
      }
  }
  ```

Analysis

- For this algorithm, the expected number of heads is exactly the same as the expected number of tails.
- Thus the expected run time of the algorithm is twice the expected number of upward jumps.
- Since we already know that the number of upward jumps is $O(\log n)$ with high probability, we can conclude that the expected search time is $O(\log n)$.