Lecture 20 — Balanced Trees Continued

Parallel and Sequential Data Structures and Algorithms, 15-210 (Fall 2011)

Lectured by Guy Blelloch — Nov 3, 2011

Today:
- Treaps continued
- Law of large numbers and high probability bounds

1 Treaps Continued

Recall that a treap is a binary search tree in which we associate with each key a random priority. The tree is maintained so that the priorities associated with the keys are in (max) heap order, i.e. the priority at a node is larger than the priorities of both of its children.

We will now analyze the expected depth of a key in the tree. This analysis pretty much the same as the analysis we did for quicksort.

Consider a set of keys K and associated priorities $p : \mathtt{key} \rightarrow int$. We assume the priorities are unique. Consider the keys laid out in order, and as with the analysis of quicksort we use i and j to refer to two positions in this order.

```
| | | | | | | | | | | | | | | | | | | |
          i             j
```

If we calculate the depth starting with zero at the root, the expected depth of a key is equivalent to the number of ancestors in the tree. So we want to know how many ancestor. We use the random indicator variable A_{ij} to indicate that j is an ancestor of i. Now the expected depth can be written as:

$$\mathbf{E}\left[\text{depth of } i \text{ in } T\right] = \sum_{j=1}^{n} \mathbf{E}\left[A_{ij}\right].$$

To analyze A_{ij} lets just consider the $|j - i| + 1$ keys and associated priorities from i to j inclusive of both ends. We consider 3 cases:

1. The element i has the highest priority.
2. One of the elements k in the middle has the highest priority (i.e., neither i nor j.
3. The element j has the highest priority.

What happens in each case?

In the first case j cannot be an ancestor of i since i has a higher priority. In the second case note that i can only be a descendant of j if k is also. This is because in a BST every branch covers a contiguous region, so if i is in the left (or right) branch of j, then k must also be.

```
          o j
       /

o i    o k
```

But since the priority of k is larger than that of j this cannot be the case so j is not an ancestor. Finally in the third case j is an ancestor of i since to separate i from j would require a key in between with a higher priority. We therefore have that j is an ancestor of i exactly when it has a priority greater than all elements from i to j (inclusive on both sides). Since priorities are selected randomly, this has a chance of $1/(j-i+1)$ and we have $\mathbf{E}\left[A_{ij}\right] = \frac{1}{|j-i|+1}$. Note that if we include the probability of either j being an ancestor of i or i being an ancestor of j then the analysis is identical to quicksort. Think about why. Recall from last lecture that the recursion tree for quicksort is identical to the structure of the corresponding treap (assuming the same keys and priorities).

Now we have

$$\begin{aligned}
\mathbf{E}\left[\text{depth of } i \text{ in } T\right] &= \sum_{j=1}^{n} \frac{1}{|j-i|+1} \\
&= \sum_{j=1}^{i-1} \frac{1}{i-j+1} + \sum_{j=i+1}^{n} \frac{1}{j-i+1} \\
&= H_{i-1} + H_{n-i-1} \\
&= O(\log n)
\end{aligned}$$

Exercise 1. *Including constant factors how does the expected depth for the first key compare to the expected depth of the middle ($i = n/2$) key.*

Split and Join on Treaps

As mentioned last week for any binary tree all we need to implement is split and join and these can be used to implement the other operations.

We claim that the split code given in lecture 18 for unbalanced trees does not need to be modified at all for Treaps.

Exercise 2. *Prove that the code for split given in lecture 18 need not be modified.*

The join code, however, does need to be changed. The new version has to check the priorities of the two roots and use whichever is greater as the new root. Here is the algorithm.

```
1   fun $join(T_1, m, T_2)$ =
2   let
3     fun $singleton(k, v) = Node(Leaf, Leaf, k, v)$
4     fun $join'(T_1, T_2)$ =
5       case $(T_1, T_2)$ of
6         $(Leaf, \_) \Rightarrow T_2$
7       | $(\_, Leaf) \Rightarrow T_1$
8       | $(Node(L_1, R_1, k_1, v_1), Node(L_2, R_2, k_2, v_2)) \Rightarrow$
9           if $(priority(k_1) > priority(k_2))$ then
10            $Node(L_1, join'(R_1, T_2), k_1, v_1)$
11          else
12            $Node(join'(T_1, L_2), R_2, k_2, v_2)$
13  in
14      case $m$ of
15        $NONE \Rightarrow join'(T_1, T_2))$
16      | $SOME(k, v) \Rightarrow join'(T_1, join'(singleton(k, v), T_2))$
17  end
```

In the code $join'$ is a version of join that does not take a middle element. Note that line 9 compares the priorities of the two roots and then places the key with the larger priority in the new root causing a recursive call to join on one of the two sides.

Theorem 1.1. *For treaps the cost of* $join(T_1, m, T_2)$ *returning* T *and of* $split(T)$ *is* $O(\log |T|)$ *expected work and span.*

Proof. The cost of split is proportional to the depth of the node where we are splitting at. Since the expected depth of a node is $O(\log n)$, the expected cost of split is $O(\log n)$. For $join(T_1, m, T_2)$ note that the code only follows T_1 down the right child and terminates when the right child is empty. Similarly it only follows T_2 down the left child and terminates when it is empty. Therefore the work is at most proportional to the sum of the depth of the rightmost key in T_1 and the depth of the leftmost key in T_2. The work of join is therefore the sum of the expected depth of these nodes which is expected $O(\log |T|)$. $\square$

We note that these bounds for split and join give us the $O(m \log(n/m))$ work bounds for union and related functions in expectation.

2 Not So Great Expectations

So far, we have argued about the expected work of quick sort and the expected depth of a node in a treap, but *is this good enough?* Let's say my company DontCrash is selling you a new air traffic control system and I say that in expectation, no two planes will get closer than 500 meters of each other—would you be satisfied? More relevant to this class, let's say you wanted to run 1000 jobs on 1000 processors and I told you that in expectation each finishes in an hour—would you be happy? How long might you have to wait?

There are two problems with expectations, at least on their own. Firstly, they tell us very little if anything about the variance. And secondly, as mentioned in an earlier lecture, the expectation of a maximum can be much higher than the maximum of expectations. The first has implications in real time systems where we need to get things done in time, and the second in getting efficient parallel algorithms (e.g., span is the max span of the two parallel calls).

Fortunately, in many cases we can use expectations to prove much stronger bounds at least when we have independent events. These bounds are what we call high probability bounds, again as mentioned in an earlier lecture. Basically, it says that we guarantee some property with probability very close to 1—in fact, so close that the difference is only inverse polynomially small in the problem size. To get some intuitions for high probability bounds, we'll take a detour into the law of large numbers.

3 The Law of Large Numbers

The *law of large numbers* says that the average of the results of a large number of independent trials converges on the expected value of a single trial. The law of large numbers is not just a law of mathematics, but also a hugely practical law that is relevant to everyday life.

Consider, for example, throwing a six sided die. The expected value is 3.5 ((1+2+3+4+5+6)/6). If we throw it just once it is reasonably likely we will not be close to this value. We might roll 1 or 6, for example, and we certainly will not roll a 3.5, at least not with any dice I've used. If we throw it 10 times it is still reasonably likely that the average is, for example, less than 3. However, if we throw it a million times then it becomes very unlikely the average is less than 3. Can anyone guess what the probability is? Well it is about e^{-1000}. That is a very-very-very small number (way smaller than one over the number of particles in the universe). In fact the probability that the average is less than 3.4 or more than 3.6 is less than e^{-40}, which is still a very-very small number. This indicates that as we throw the dice many times the average converges on the mean.

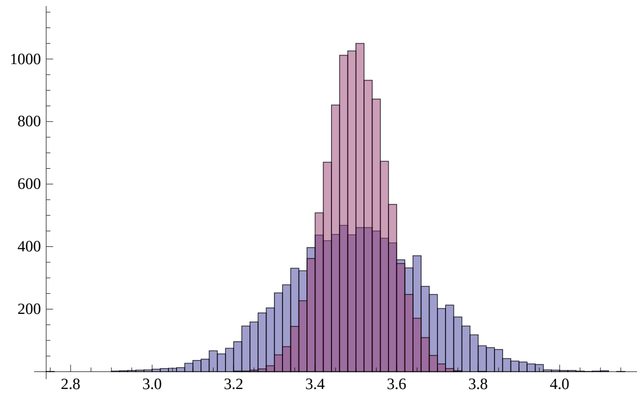

Figure 1: In blue: a $10,000$-trial histogram of the average value of throwing a six-sided die 100 times. In purple: a $10,000$-trial histogram of the average value of throwing a six-sided die 500 times

Another example was the extra credit in assignment 5 where you had to tell us how many trials you had to run to get a reasonable estimate on the expected radius of a random point in a circle.

OK, perhaps throwing dice is not relevant to everyday life, except perhaps for compulsive gamblers, and the radius of random points only to 210 assignments. However, the law of large numbers can be applied to many other things. For example, when investing in stocks, a well-known strategy involves choosing a large number of stocks instead of a single stock to make it much less likely that you will loose big (or gain big). Or, in determining how well a student is doing at Carnegie Mellon: the student might be unlucky and do badly in one class or on one test, but on average across all their classes, their performance will be a good prediction of how well they are understanding or are interested in the material.

The law of large numbers should be intuitive, we hope, but can also be formalized. In particular, for any $\varepsilon > 0$, it[1] states that

$$\lim_{n\to\infty} \mathbf{Pr}\left[|\overline{X}_n - \mu| > \varepsilon\right] = 0\,.$$

where $\mu = E[X]$ is the expected value of X (e.g. 3.5 for a roll of the dice) and $\overline{X}_n$ is the average of the measured value over n independent trials (e.g., the average over n rolls of the dice). This statement says that as n approaches infinity the average over n trials converges exactly to the expected value for a single trial. This statement, however, does not tell us anything about how many trials we need to run before our confidence is high. Who cares what happens at infinity; we will never get there.

Fortunately, there are many theorems applicable in different circumstances that do tell us about the rate of convergence. One that is particularly useful in algorithm analysis is the following bound, known as the Chernoff bound—well, actually, *a* Chernoff bound[2].

Theorem 3.1 (Chernoff Bound). *Given a set of independent random variables $X_1, \ldots, X_n$, each taking a real value between 0 and 1, with $X = \sum_{i=1}^n X_i$ and $\mathbf{E}[X] = \mu$, then for any $\lambda > 0$*

$$\mathbf{Pr}\left[X > (1+\lambda)\mu\right] \;<\; \exp\left(\frac{-\mu\lambda^2}{2+\lambda}\right).$$

The Chernoff bounds are named after Harvard Statistics professor Herman Chernoff—although the very form[3] we're using had been proved in 1924, almost 30 years before Chernoff showed a generalization of it. Proving this is beyond the scope of this course. But we want to make a few remarks about the theorem and tell you how to apply it.

First, let's try to understand what the bound says in words. The factor λ specifies a distance away from the mean. So, for example, by setting $\lambda = .2$, we are asking what the probability is that the actual result is more than 20% larger than the expected result. As we expected, the farther we are away from the mean, the less likely our value will be—this probability, in fact, decays exponentially fast in (roughly) λ^2. Note that this adds to our repertoire of deviation bounds (you have seen Markov's inequality from 15-251).

Second, independence is key in getting this sharp bound. By comparison, Markov's inequality which doesn't require independence can only give $\mathbf{Pr}\left[X > (1+\lambda)\mu\right] < \frac{1}{1+\lambda}$, which is a much weaker bound. As weak as Markov's bound is, it is in certain cases the best we can make of if we can't say anything about independence. Indeed, independence is key here: it makes it unlikely that the random variables will "collude" to pull the mean one way or the other.

[1]This is technically known as the weak law of large numbers; there's another form with a stronger guarantee, known as the strong law of large numbers.

[2]In the literature, Chernoff bounds are more or less a family of inequalities—a generic term for various bounds that deal with the idea that the aggregate of many independent random variables concentrates around the mean.

[3]There is a sharper—and messier—form: for $\lambda > 0$, $\mathbf{Pr}\left[X > (1+\lambda)\mu\right] \le \left(\frac{e^\lambda}{(1+\lambda)^{(1+\lambda)}}\right)^\mu$.

Third, the Chernoff bound can be seen as a quantitive version of the law of large numbers. At first glance, you might ask: since there is no n in the equation, how this relates to the law of large numbers? The law shows up indirectly since each variable X_i is a number between 0 and 1. Therefore, we have $n \geq \mu$ and hence when μ goes up, n goes up and the probability of being far from the mean goes down, as we would expect from the law of large numbers.

Now let's apply this theorem to determine the probability that if we throw $1,000$ coins that more than 60% come up heads. In this case, $\mu = 500$ and $\lambda = .2$ (since $(1 + .2)500 = 600$). We plug this in and get

$$\begin{aligned} \textbf{Pr}\left[X > 600\right] &< e^{\frac{-500(.2)^2}{2+.2}} \\ &< 0.00012 \end{aligned}$$

If we apply it to $10,000$ coins flips and ask what the probability that more than 60% come up heads we have $\mu = 5,000$ (again, $(1 + 0.2)5000 = 6000$), giving

$$\begin{aligned} \textbf{Pr}\left[X > 6000\right] &< e^{\frac{-5000(.2)^2}{2+0.2}} \\ &< 3.3 \times 10^{-40} \end{aligned}$$

This is a significant decrease in probability, which demonstrates the power of the law of large numbers. Once again, it is important to remember that the Chernoff bounds in particular and the law of large numbers in general are only true if each random variable is independent of the others[4] Consider, for example, the stocks for a set of companies that are all in the same business. These are clearly highly correlated, and hence not independent. Thus, investing in them does not give you the same safety of large numbers as investing in a set of unrelated companies.[5] We also note that the Chernoff bounds are sloppy and the actual bounds are typically much stronger. Since we are interested in upper bounds on the probability, this is OK (our upper bounds are just a bit loose).

4 High Probability Bounds for Quick Sort and Treaps

We now use the Chernoff bounds to get high-probability bounds for the depth of any node in a Treap. Since we said the recursion tree for quick sort has the same structure as a Treap, this will also give us high probability bounds for the depth of the quick sort tree, which can then be used to bound the span of quick sort with high probability.

Recall that the random variable A_{ij} indicates that j is an ancestor of i. It turns out all we have to do is argue that for a given i the random variables A_{ij} are independent. If they are, we can then use the Chernoff bounds. In particular, we want to analyze for any key i, $A_i = \sum_{j=1}^{n} A_{ij}$ (recall that this corresponds to the depth of key i in a treap). As derived earlier the expectation $\mu = \mathbf{E}\left[A_i\right]$ is $H_{i-1} + H_{n-i-1}$ which lies between $\ln n$ and $2 \ln n$. If we set $\lambda = 4$, we have

[4]For full disclosure, we can relax this guarantee a bit; indeed, we can tolerate certain amount of dependence at the expense of a weaker bound, but this is beyond the scope of the class.

[5]Of course, no two stocks are truly independent, so the theorem cannot be directly applied to stocks, although one might be able to separate out the common trend.

$$\mathbf{Pr}\left[X > (1+4)\mu\right] \;<\; e^{\frac{-\mu(4)^2}{2+4}}$$

Now since $\mu \geq \ln n$ we have

$$\begin{aligned}\mathbf{Pr}\left[X > 5\mu\right] &< e^{-2\ln n} \\ &= \frac{1}{n^2}\end{aligned}$$

This means that the probability that a node is deeper than fives times its expectation is at most $\frac{1}{n^2}$, which is very small for reasonably large n (e.g. for $n = 10^6$ it is 10^{-12}). Since the expected depth of a key is at most $2\ln n$, we have

Theorem 4.1. *For a Treap with n keys, the probability that a key i is deeper than $10\ln n$ is at most $1/n^2$.*

Note that this just gives us the probability that any one key is deeper than 5 times its expectation. To figure out the worst case over all keys, we have to multiply the probability that any one of them is deeper than 5 times the expectation by the number of keys. More formally, let $\mathcal{E}_x$ be the "bad" event that the key x is deeper than 5 times its expectation. So then, by applying the union bound, we have

$$\begin{aligned}\mathbf{Pr}\left[\text{any key is deeper than } 10\ln n\right] &= \mathbf{Pr}\left[\exists x.\mathcal{E}_x\right] \\ &\leq \sum_x \mathbf{Pr}\left[\mathcal{E}_x\right] \\ &\leq n \times \frac{1}{n^2} = \frac{1}{n}.\end{aligned}$$

This gives us the following theorem:

Theorem 4.2. *For a Treap with n keys, the probability that any key is deeper than $10\ln n$ is at most $1/n$.*

We now return to why for a given i the variables A_{ij} are independent. This argument is a bit subtle, so be impressed with yourself if you get it the first time. Let's just consider the $j > i$ (the others are true by a symmetrical argument), and scan from i forward. Each time we look at the j whether it is an ancestor only depends on whether its priority is larger than all priorities from i to $j-1$. It does not depend on the relative ordering of those priorities. It is therefore independent of those priorities.

Conclusions

- The expected depth of each node i in a treap can be analyzed by determining for every other node j the expectation that j is an ancestor of i. These expectations can then be summed across all j giving $O(\log n)$ for the expected depth of each node.
- The analysis of Treaps is very similar to the analysis of quicksort.

- The split operation for treaps is the same as for unbalanced trees. The join operation is slightly more involved. Since nodes have expected $O(\log n)$ depth, they both run in $O(\log n)$ expected work and span.
- Although expectations can sometimes be useful, sometimes we want a stronger guarantee. In particular we might want to know that we will succeed (e.g. complete an operation in an allotted time) with high probability.
- High probability bounds can be generated from the law of large numbers. A particular form of this law are the so-called chernoff bounds, which give us a specific probability of our variable of concern diverging significantly from the mean.
- We can use Chernoff bounds to prove that the depth of a treap with random priorities is $O(\log n)$ with high probability. Since the recursion tree for quicksort has the same distribution as a treap, this also gives the same bounds for the depth of recursion of quicksort.