1 Back to Treaps

How does this help with our analysis of treaps? Let us assume that you are given a set of elements with randomly generated priorities and you create a treap out of them. What is the expected search cost? The expected search cost is the expected depth of a node in the tree. Let us analyze that.

Consider a set of keys $K$ and associated priorities $p : key \rightarrow int$. We assume the priorities are unique. Consider the keys laid out in order (i.e., by taking an inorder traversal of the Treap), and as with the analysis of quicksort we use $T_i$ and $T_j$ to refer to two positions in the sorted order.

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 $B_{ij}$ to indicate that $T_j$ is an ancestor of $T_i$. Now the expected depth can be written as:

$$E[\text{depth of } i \text{ in } T] = \sum_{j=1, j \neq i}^{n} E[B_{ij}].$$

To analyze $B_{ij}$ let’s just consider the $|j - i| + 1$ keys and associated priorities from $T_i$ to $T_j$ inclusive of both ends. (Note that keys outside of this range is irrelevant, because they don’t effect the probability of $B_{ij}$.) We consider three cases:

1. The element $T_i$ has the highest priority.
2. One of the elements $T_k$ in the middle has the highest priority (i.e., neither $T_i$ nor $T_j$.
3. The element $T_j$ has the highest priority.

What happens in each case?

In the first case $T_j$ cannot be an ancestor of $T_i$ since $T_i$ has a higher priority. In the second case, note that $T_k$ will be picked to be the root of the subtree containing $T_i$ and $T_j$ before either $T_i$ or $T_j$ is picked, and separate $T_i$ and $T_j$ into two different subtrees based on the BST property. Thus, $T_j$ cannot possibly be $T_i$’s ancestor in this case. Finally, in the third case $T_j$ is an ancestor of $T_i$, since there is no key between them with a higher priority to separate them into two subtrees.

We therefore have that $T_j$ is an ancestor of $T_i$ exactly when it has a priority greater than all elements from $T_i$ to $T_j$ (inclusive on both sides). Since priorities are selected randomly, this has a chance of $1/(j - i + 1)$ and we have $E[B_{ij}] = 1/(j - i + 1)$. 

\[ \]
Now we have

\[ E[\text{depth of } i \text{ in } T] = \sum_{j=1, j \neq i}^{n} \frac{1}{|j-i| + 1} \]

\[ = \sum_{j=1}^{i-1} \frac{1}{i-j+1} + \sum_{j=i+1}^{n} \frac{1}{j-i+1} \]

\[ = \frac{1}{i} + \frac{1}{i-1} + ... + \frac{1}{2} + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n-i+1} \]

\[ = H_i + H_{n-i+1} \]

\[ = O(\log n) \]

**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.

Is this similar to the bound you got for red-black trees? What is the red-black tree guarantee? It is that the height of the tree is \(O(\log n)\). The answer is no. The fact that the expected depth of a key (and hence its expected search cost) is \(O(\log n)\) does not imply that the expected height of the tree is \(O(\log n)\). To see why, think about a family of trees where every key has depth \(O(\log n)\) expect for one key which has depth of \(\sqrt{n}\). For this family of trees, the expected depth of a node is \(O(\log n)\), but the expected height of a tree is \(\sqrt{n}\). Or, putting it differently — the height of the tree depends on the depth of the key that has maximum depth. Even though we have the expected depth of a key, that doesn’t tell us the expected value of the height of the tree. Recall that \(E[\text{max depth across all keys}]\) is not the same as \(\max\{E[\text{depth of key 1}], E[\text{depth of key 2}], ...\}\).

It turns out that in order to argue that the height of a tree is \(O(\log n)\), we need a high-probability bound.

## 2 High Probability Bounds

So far, we have generally proved expectation bounds — for example, the running time of Quicksort is \(O(n \log n)\) and the number of rounds in the MST calculation is \(O(\log n)\). However, expectation bounds don’t tell us about the probability of failure — that is, apart from the weak results you can get using Markov Inequality, you can not say how frequently your algorithm may run slower than the expectation bounds indicate.

We will now think about a stronger way of analyzing randomized algorithms — **high probability bounds**. We say that an event \(E\) takes place with high probability if

\[ \Pr \{ E \} \geq 1 - \frac{1}{n^c} \]
for some constant \( c \geq 1 \). Here \( n \) is the instance size. Typically, in bounding the cost of an algorithm, we’re interested in showing that \( X_n < A \) with high probability. In other words, we want to say

\[
\Pr \{ X_n < A \} \geq 1 - \frac{1}{n^k}
\]

or equivalently,

\[
\Pr \{ X_n \geq A \} < \frac{1}{n^k}
\]

This is saying that the probability of a “bad” event (the random variable takes on a value greater than some value \( A \)) is less than \( 1/n^k \) for some constant \( k \). This probability is very low. We don’t normally say something happens “with low probability,” because we are interested in the opposite “good” case. For example, for us the \( X_n \) will be some cost measure (in our case, the number of rounds for MST algorithm or the height of the treap), and we want to say that with high probability the cost will be less than some \( A \), i.e., that the probability that it is greater is small.

An important inequality to know for calculating many high probability bounds is \textit{Chernoff bound}.

\begin{table}[h]
\begin{tabular}{|l|}
\hline
\textbf{Theorem 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 \( \mathbb{E} [X] = \mu \), then for any \( \lambda > 0 \)
\[
\Pr \{ X > (1 + \lambda)\mu \} < \exp \left( \frac{-\mu \lambda^2}{2 + \lambda} \right).
\]
\hline
\end{tabular}
\end{table}

First, let’s try to understand what the bound says in words. The factor \( \lambda \) 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) \( \lambda^2 \).

Second, independence is key in getting this sharp bound. By comparison, Markov’s inequality which doesn’t require independence can only give \( \Pr \{ X > (1 + \lambda)\mu \} < \frac{1}{1+\lambda} \), 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.

Now let’s try to apply Chernoff bounds to the minimum spanning tree problem. Let \( R_i \) be the random variable that represents the number of nodes remaining in the graph after round \( i \). We

\footnote{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. There are different versions of Chernoff bound; we will see one of them, but the idea is similar; some versions give tighter bounds than others in some situations.}
argued in a prior lecture that $E[R_i] \geq 3n/4$. Therefore, using Markov’s Inequality, we can argue that $\Pr\{R_i > 7n/8\} \leq 6/7$.

We now say that a round is successful if at most $7n/8$ nodes remain after it finishes and is unsuccessful otherwise. We know that the total number of successful rounds is at most $\log_8 n$. Therefore, if we define an indicator random variable $X_i$ which is 1 if around is unsuccessful, we get $E[X_i] \leq 6/7$. Say $X$ is a random variable representing the number of unsuccessful rounds — therefore, $X = X_1 + X_2 \ldots$. If we had $\log_8 n$ successful rounds, $E[X] \leq 6\log_8 n$. Also, all $X_i$’s are independent. Therefore, using Chernoff Bounds, we have

$$\Pr\{X > (1 + \lambda)6\log_8 n\} \leq \exp\left(-\frac{(6\log_8 n)\lambda^2}{2 + \lambda}\right) \leq \exp\left(-\frac{(6\ln n/\ln 8)\lambda^2}{2 + \lambda}\right)$$

$$\Pr\{X > 18\log_8 n\} \leq n^{-3}$$

If we set $\lambda = 2$. As we increase $\lambda$, we can make the probability smaller and smaller. Therefore, the probability of having more than $\omega(\lg n)$ unsuccessful rounds (and therefore, $\omega(\lg n)$ total rounds) is very small and gets smaller as $n$ gets larger.

Another reason to use high probability bounds is that they are composable with respect to maximums. Say you had the following code:

```plaintext
FOO()
1   spawn BAR-1()
2   spawn BAR-2()
3   spawn BAR-3()
4       ...
5   spawn BAR-n()
6   sync
```

To calculate the span of this computation, we need to compose the costs across the three functions by taking the max. Unfortunately, knowing the expectation of contracting each component is not always good enough when composing costs if we are computing the span. In analyzing work, we just add everything up. Linearity of expectations turns out to be very useful for this purpose since if we know the expected work for each component, we can just add them to get the overall expected work. When analyzing span over a parallel computation, however, we take the maximum of the span of the components. Unfortunately the following is (in general) not true: $\max(E[X], E[Y]) = E[\max(X, Y)]$. We therefore cannot simply take the maximum of expectations across components to get the overall expected span.

As an example consider two parallel components each which have span either 1 or 5 each with .5 probability. The expected span for each of the components is therefore $.5 \times 1 + .5 \times 5 = 3$. 

4
Now, in combination there are four possible configurations \(((1, 1), (1, 5), (5, 1), (5, 5))\). Since we take the max of the two components the overall span for the four are \(1, 5, 5, 5\). The expected span is the average of these: \(.25 \times (1 + 5 + 5 + 5) = 4\). However if we simply took the maximum of the expected span of each component we would have \(\max(3, 3) = 3\), which is wrong. The difference can be much greater than this example.

We can deal with this issue by using high-probability bounds instead. In particular, if we have the high-probability bounds, we can use the union bound. The union bound (Boole’s inequality) states that for a set of events \(A_1, A_2, \ldots\), that

\[
\Pr \left\{ \bigcup_i A_i \right\} \leq \sum_i \Pr \{A_i\}.
\]

If the events do not overlap (their sets of outcomes do not intersect) then the equation is an equality since we just add up the probabilities of every outcome. If they do overlap then the probability of the union is strictly less than the sum of the probabilities since some outcomes are shared.

Now say that \(X_i\) is the random variable that represents the span of \(\text{BAR-}i()\) and say that we had a high probability bounds that said that \(\Pr \{X_i \geq k \lg n\} \leq 1/n^c\). We can use the union bound to say that \(\Pr \left\{ \bigcup_i (X_i \geq k \lg n) \right\} \leq 1/n^{c-1}\), which also implies that the probability that the span of \(\text{FOO}\) is larger than \(k \lg n\) is smaller than \(1/n^{c-1}\).

We will use similar ideas to calculate the expected height of a treap.