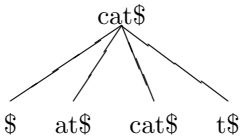
Hash Tables and Tries

Since hash tables and tries are the current topics in lecture, we will briefly introduce *suffix trees*, a particular type of trie, and demonstrate an application utilizing that data structure. We will work an example concerning hash tables.

Suffix Trees

A suffix tree is a trie that contains all of the suffixes of a given string in lexicographic order. For example, consider the string *cat$* (note: this is not the world's most interesting string. Mississipi is the classic choice, but the tree is very big). In lexicographic order, the suffixes are {$, at$, cat$, t$}. The dollar sign indicates the end of the string.



Note: this is a really lousy drawing of a suffix tree. The labels (cat$, etc.) are actually meant to be on the edges, and the nodes are labeled with the suffix number (like 1 for cat$, 2 for at$, 3 for t$ and 4 for $). For a given string of length n, a suffix tree can actually be constructed in linear time. To read more about this result, look at the McCreight algorithm. The timing analysis is very easy to follow (see *Suffix Trees and Suffix Arrays* by Aluru), but it is too long to present in section.

Suffix Tree Example

Given a circular string S, we are asked to find the lexicographically least linearization (arbitrary cut) of S in $O(n)$ time, where $n = |S|$. For example, in trusted *mississippi*, the lexicographically least linearization would be *imississippi*. Our algorithm is as follows:

1. Make an arbitrary cut in the string.
2. Double the string.
3. Build a suffix tree.
4. DFS our suffix tree in lexicographic order. The first string of length m that we find is the lexicographically least linearization.

This algorithm runs in $O(n)$ time because each of the steps takes at most $O(n)$. Cutting and doubling the string are clearly linear in the size of the string. Building the suffix tree is $O(n)$ (using known algorithms) and a DFS of a suffix tree is also $O(n)$. We justify the $O(n)$ bound for the DFS by noting that there are $(n+1)$ leaf nodes, corresponding to each of the $(n+1)$ suffixes (including the end-of-string character), and therefore $2n$ is a very generous upper bound for the total nodes in tree, because each internal node has at least one child. Since our graph is a tree, there are at most $|V-1|$ edges in the tree, and thus our well-known running time $O(V+E)$ for DFS is clearly $O(n)$.

CLRS 11.5-1

Let $p(n,m)$ be the probability that there are no collisions when inserting n keys into a table of size m. We want to find a relation between n and m that minimizes collisions. For this problem, we make two assumptions about our hash function.

1. The hash function is universal. Thus, $Pr[h(x) = h(y)] \leq 1/m$.

2. Our implementation is **open addressing**. Thus, there are no secondary tables or linked lists. All items are stored directly in the hash table.

We calculate $p(n, m)$ as follows:

$$p(n,m) = 1 \cdot (1 - 1/m) \cdot (1 - 2/m) \cdot (1 - 3/m) \cdots (1 - (n-1)/m) \tag{1}$$

We justify this equation in the following manner. The $\{1/m, 2/m, \ldots, (n-1)/m\}$ terms represent the probability of a collision for the first insert, second insert, third insert, etc, because we are using open addressing and an item hashed to the table is an item stored in the table. Thus, we take the product of the complement of those probabilities to determine $p(n, m)$. Now we remind ourselves of the Taylor's expansion of e^x, and also the normal probability density function for a random variable X.

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots \tag{2}$$

$$e^x \geq 1 + x \tag{3}$$

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4}$$

Notice that as x approaches 0, $e^x > 1$. This is equation 3.11 of CLRS. Thus, we see that:

$$p(n,m) = 1 \cdot (1 - 1/m) \cdot (1 - 2/m) \cdot (1 - 3/m) \cdots (1 - (n-1)/m) \tag{5}$$

$$p(n,m) = 1 \cdot (1 + (-1/m)) \cdot (1 + (-2/m)) \cdot (1 + (-3/m)) \cdots (1 + (-(n-1)/m)) \tag{6}$$

$$p(n,m) \leq e^{(-1/m)} \cdot e^{(-2/m)} \cdot e^{(-3/m)} \cdots e^{(-(n-1)/m)} \tag{7}$$

$$p(n,m) \leq e^{1/m} \cdot e^{-(1+2+3+\cdots(n-1))} \tag{8}$$

$$p(n,m) \leq e^{1/m} \cdot e^{-\frac{n(n-1)}{2}} \tag{9}$$

$$p(n,m) \leq e^{-\frac{n(n-1)}{2m}} \tag{10}$$

In order to more clearly, visualize this probability distribution, consider the following simplification to $p(n, m)$:

$$p(n,m) \leq e^{-\frac{n(n-1)}{2m}} \tag{11}$$

$$p(n,m) \leq e^{-\frac{n^2 - n + \frac{1}{4} - \frac{1}{4}}{2m}} \tag{12}$$

$$p(n,m) \leq e^{-\frac{(n^2 - n + \frac{1}{4}) + \frac{1}{4}}{2m}} \tag{13}$$

$$p(n,m) \leq e^{-\frac{(n - \frac{1}{2})^2 + \frac{1}{4}}{2m}} \tag{14}$$

$$p(n,m) \leq e^{-\frac{(n - \frac{1}{2})^2}{2m}} e^{\frac{1}{8m}} \tag{15}$$

$$\tag{16}$$

Notice that $p(n, m)$ is bounded by the normal distribution with expected value $\mu = 1/2$ and $\sigma = \sqrt{m}$. Therefore, because we know that the normal distribution is a bell curve, we know that when $n > \sqrt{m}$, this means that n is one standard deviation from the expected value, and the probability of this occuring rapidly decreases to zero according to the bell curve. This is an important result for perfect hashing because it gives us a ratio of n and m which minimizes collisions. Because $p(n, m)$ is the probability of no collisions, as long as n stays within the standard deviation, the probability of *no* collisions is high. But once $n > \sqrt{m}$, the probability of *no* collisions rapidly decreases, and thus the probability of a collision is increased.