January 19, 2012
Fletcher Stump Smith

Randomized Algorithms on Data Structures

1 Introduction

One application of randomized algorithms in the area of data structures, specifically, lists. A list is a finite sequence of values, $x_1, x_2, \ldots, x_j$. One implementation a list is called a linked list. On a linked list each x_i is a pair (v_i, p_i) where v_i is its value and p_i is a pointer to the next element in the list. One problem with a linked list is that it is not very efficient for some operations. Specifically search, insertion, and deletion, are all $O(n)$. The list can be made more efficient by organizing the data into some type of data structure. One option is a binary tree. A binary tree has a root node, branch nodes, and leaf nodes. Each branch node (including the root node) has a left and right sub tree. The number of steps required to do a binary search only depends on the height of the tree, so search on a binary tree is $O(logn)$.

2 Skip Lists

A skip list is a layered a data structure.

$$L_1 : -\infty \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_5 \rightarrow \ldots x_i$$

$$L_1 - \infty \rightarrow x_0 \rightarrow x_2 \rightarrow x_4 \rightarrow x_6 \rightarrow x_8 \ldots$$

$$L_2 - \infty \rightarrow x_0 \rightarrow x_4 \rightarrow x_8 \ldots$$

Each element is linked to the corresponding element in the previous list.A skip list is basically just a more organized binary tree. The cost of search is $O(n)$.One problem with a skip list is that any insertion or deletion will mess up the structure. The Classical solution to the problem is using self adjusting data structures. Unfortunately changing the data structure too much can be expensive. For example increasing the size from 2^n to $2^n + 1$. For this reason we will instead consider randomized skip lists.

2.1 Randomization

Start with a list

$$L_0 - \infty \rightarrow x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_5 \rightarrow \ldots x_i \rightarrow \ldots$$

For each level below it we keep an element above it with probability p. Let H_i denote the height of x_i, that is the highest level containing x_i.

$$Pr(H_i = k) = p^k(1-p)$$

$$Pr(H_i \geq k) = p^k$$

Let $H = max_i H_i$.

Our goal is to show that $Pr(H \geq k)$ is small

$$\begin{aligned} Pr(H \geq k) &= Pr(\exists i, H_i \geq k) \\ &\leq \sum_i Pr(H_i \geq k) \\ &= np^k \end{aligned} \tag{1}$$

If we want to make $np^k \leq (1/n)$ while still being able to do search in $O(logn)$. Consider the case $p = 1/2, k = 10logn$. In that case there is certainly no problem with having to travel too many vertical arrows while doing a search, but there may be a problem with horizontal arrows. Let $Z = H + W$ where W denotes the width of the search traversal (number of right arrows).

$$\begin{aligned} Pr(Z \geq m) &= Pr(Z > m|H < k)P(H < k) + Pr(Z > m|H \geq k)Pr(H \geq k) \\ &\leq Pr(Z > m|H < k) + Pr(H \geq k) \\ &\leq Pr(Z > m|H < k) + np^k \\ &= Pr(W > m - k) + np^k \end{aligned} \tag{2}$$

Let $m = 30logn$. Observe $\mu = E[W] = pm = m/2$.

$$Pr(W > m - k) = Pr(W > \mu + 10logn) \leq exp(-\frac{2(10logn)^2}{m}) \leq \frac{1}{n^2}$$

As the probablity that any search is too long is less then $\frac{1}{n^2}$ and there are elements to search for, the probablity that some search is longer than $O(logn)$ is less than $\frac{1}{n}$.

3 Quick Sort

Quick Sort is an algorithm for ordering the elements of a list. The algorithm takes as input $x = [x_1, x_2, \ldots, x_n]$. For simplicities sake we will assume the x_i are all distinct numbers. Our goal is to sort the list in $O(nlogn)$ time with probability $p = 1 - 1/n$. We start by picking a pivot element in our list.

$$[y_1, y_2, \ldots, y_{i-1}, v, z_{i+1}, \ldots, z_n]$$

where $v > y1, \ldots y_{i-1}$ and $v < z_{i+1} \ldots z_n$. Quick sort is then preformed recursively on each half list. We want to show that the hight of the recursion tree is $\leq \Theta(logn)$.We will do his by showing

$$Pr(height > 300logn) \leq Pr(\exists a\ path\ length > 300logn) \leq \frac{1}{n^2}$$

A pivot is said to be good if both sublists is at least $\frac{1}{3}$rd the size of the original list. Let

$$X_i = \begin{cases} 0 & \text{ith piviot good} \\ 1 & \text{otherwise} \end{cases}$$

$$X = \sum X_i$$

Let S_i denote the size of the largest sub-list at step i. If we get good pivots $S_i \leq \frac{2}{3} S_{i-1}$. Observe for a list of size n there can be at most $\frac{logn}{log(3/2)}$ good nodes.

Let $E(X) = \frac{2}{3}m = 200logn$.

Using the concentration inequality from last week.

$$Pr(X \geq \frac{3}{2}E(X)) \leq \frac{e^{\delta}}{(1+\delta)^{1+\delta}}^{E(X)} = \frac{e^{\frac{1}{2}}}{(\frac{3}{2})^{\frac{3}{2}}}^{200logn} \leq \frac{1}{n^2}$$

Thus the probability of a given path being bad is less than $\frac{1}{n^2}$ thus the probability of a bad path is less that $\frac{1}{n}$.

References

[1] D. Dubhashi, A. Panconesi, Concentration of Measure for the Analysis of Randomized Algorithms, Cambrige University Press1, 2009