COMPSCI 330: Design and Analysis of Algorithms 11/4/2014

Lecture #19

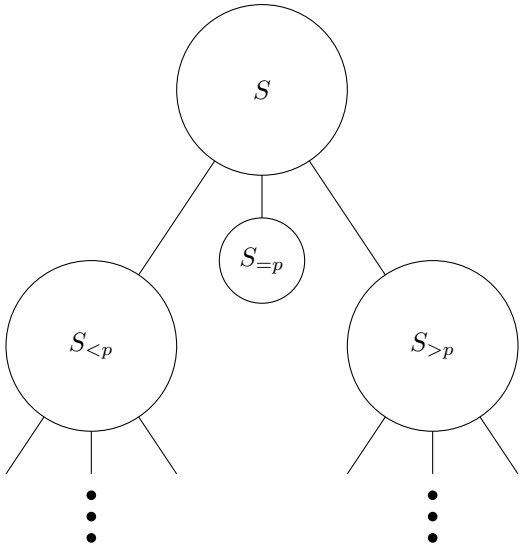
Lecturer: Debmalya Panigrahi *Scribe: Samuel Haney*

1 Overview

In this lecture, we analyze randomized quicksort and study hashing.

2 Randomized Quicksort Analysis

Recall that the randomized quicksort algorithm picks a pivot at random, and then partitions the elements into three sets: all the elements less than the pivot, all elements equal to the pivot, and all elements greater than the pivot.



We analyze the runtime using charging. In the computation tree shown above, we count the number times each element appears. The sum over all elements is equal to the sum of the sizes of all the sets in the computation tree, which is proportional to the runtime.

Claim 1. *The number of times an element appears in sets in the computation tree is* $O(\log n)$ *in expectation.*

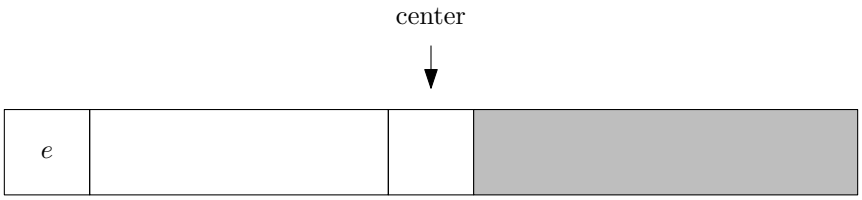
Each set S in the computation tree has three children. Let $S_{=p}$ be child set with elements equal to the pivot. Let $S_{>p}$ and $S_{<p}$ be defined similarly. We color the tree edges to these children as follows:

- Color the edge to $S_{=p}$ red.
- Color the edge to the larger of $S_{>p}$ and $S_{<p}$ black.
- Color the edge to the smaller of $S_{>p}$ and $S_{<p}$ blue.

Now, we trace the path of some element e through the computation tree, and count the number of edges of each color along the path.

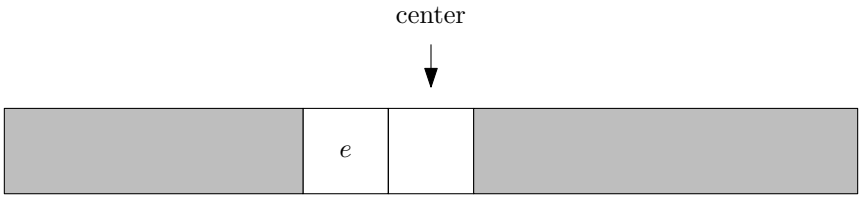
- The number of red edges is 1. Only the last edge can be red.
- The number of blue edges is at most $\log_2 n$, since the smaller of $S_{>p}$ and $S_{<p}$ has at most $\frac{1}{2} \cdot |S|$ elements.
- The number of black edges, using the definitions we have given, could be large.

What is the probability that an edge is black? That is, given that $e \in S$, what is the probability that the next edge in e's path is black? Suppose e is the smallest element in the array.



For this element, any pivot in the right half of the array, illustrated with the shaded region in the image above, will cause e's next edge to black. Therefore, the probability that the next edge is black in this case is roughly $\frac{1}{2}$.

However, consider some element e near the center of the array.



The shaded region shows which pivots will cause the next edge in e's path to be black. For this element, the probability that the next edge is black is nearly 1. The probability will be high for all elements near the center of the array.

We need to modify our definition of black edges and blue edges to fix the problem. If $|S_{>p}| > \frac{3}{4}|S|$, color the edge to $S_{>p}$ black. Otherwise, color the edge to $S_{>p}$ blue. We color the edge to $S_{<p}$ using identical rules. Note that now it is possible for both edges to be blue (but not both to be black). We have the following:

- The number of red edges is still 1.
- The number of blue edges is at most $\log_{4/3} n$.

To calculate the probability of an edge being black, we split the array into four equal parts:

If the pivot is selected from the shaded quadrants, one of the edges will be blue and the other will be black. If the pivot is selected from the non-shaded quadrants, both edges will be blue. Therefore, regardless of which element $e \in S$ we pick, the probability that the next edge on e's path is black is at most $\frac{1}{2}$.

Now, we can bound the number of black edges in a path. Let X_i a random variable, equal to the number of black edges between the ith and the $(i+1)$st blue edge. As we have seen,

$$\mathbb{E}[X_i] \leq \frac{1}{p_{success}} \leq \frac{1}{1/2} = 2.$$

By linearity of expectation, we have

$$\mathbb{E}[\text{total number of black edges}] = \mathbb{E}\left[\sum_i X_i\right] = \sum_i \mathbb{E}[X_i] \leq 2\left(\log_{4/3} n\right) = O(\log n).$$

We now need to bound the total running time, T. Let T_e be the amount of running time charged to e (which is proportional to the number of sets e appears in).

$$\mathbb{E}[T] = \mathbb{E}\left[\sum_e T_e\right] = \sum_e \mathbb{E}[T_e] = \sum_e O(\log n) = O(n \log n).$$

3 Hashing

We use hashing to solve the following problem: we have keys which we want to add, remove, and lookup. Keys belong to some universe. Hashing is useful when the number of keys stored is small compared to the universe, otherwise and array will work.

We now define the problem more formally:

Definition 1. *Let U be a universe. Let S be the hash table, $|S| = n$, $(n \ll U)$. A hash function h is a function which maps from U to the n slots in S.*

We first construct a simple hash function, h_1.

Definition 2. *Let U be a universe and n be the number of keys to be stored in the hash table. Split U into n contiguous sections, and assign key k an index in the hash table as follows:*

$$h_1(x) = i, \text{ where } x \text{ belongs to the } i\text{th section of } U.$$

Our goal is to avoid *collisions*.

Definition 3. *Let $x, y \in U$ and h be a hash function. x and y collide if $h(x) = h(y)$.*

Collisions are unavoidable, but should be minimized. For hash function h_1, the fraction of pairs x, y which collide is

$$\frac{(|U|/n)^2 \cdot n}{|U|^2} = \frac{1}{n}.$$

For any way we divide up universe, $1/n$ is the fewest number of total collisions possible. Instead, we want to find a hash function which satisfies the following property:

Property 1. *For all $x, y \in U$, $\Pr[h(x) = h(y)] \leq \frac{1}{n}$.*

Note that h_1 does not satisfy Property 1, since for any x, y in the same section of U, $\Pr[h(x) = h(y)] = 1$. It is clear that no deterministic algorithm can satisfy Property 1, so we must use randomization. We define h_2 with this in mind.

Definition 4. *Let U be a universe and n be the number of keys to be stored in the hash table. Assign $h_2(x)$*

$$h_2(x) = X, \text{ where } X \text{ is sampled uniformly at random from } \{1, \ldots, n\}.$$

Claim 2. *Hash function h_2 satisfies Property 1.*

Proof.

$$\begin{aligned}
\Pr[x, y \text{ collide}] &= \Pr[h(x) = h(y)] \\
&= \sum_{k \in S} \Pr[h(x) = h(y) = k] \\
&= \sum_{k \in S} \frac{1}{n^2} \\
&= \frac{1}{n}.
\end{aligned}$$

□

The problem with h_2 is that we cannot efficiently find where each key is stored. We would need to store the location of each key in the hash table, which is exactly the problem we are trying to solve. Therefore, we would also like our hash functions to satisfy the following property.

Property 2. *h can be stored succinctly and $h(x)$ can be recovered efficiently.*

We will propose one more hashing scheme which will satisfy both Property 1 and Property 2. First, define field F_p:

Definition 5. *Let F_p be the set $\{0, 1, \ldots, p\}$, where p is prime, with operations $+$ and $\cdot$ defined as follows:*

- $a + b \stackrel{\text{def}}{=} (a + b) \mod p$, *and*
- $a \cdot b \stackrel{\text{def}}{=} (a \cdot b) \mod p$.

Note that F_p is closed under $+$ and $\cdot$.

Definition 6. *Let U be a universe and n be the number of keys to be stored in the hash table. Assume $n = p$ is prime. For $x \in U$, let $x_1, \cdots, x_k$ be the digits of x when written in base p (i.e. $x_i \in F_p$, and $x = \sum_{i=1}^{k} x_i \cdot p^{k-1}$). Then,*

$$h_3(x) = \left(\sum_{i=1}^{k} a_i \cdot x_i \right) \mod p,$$

where a_i is chosen uniformly at random from F_p, for all i.

Claim 3. *Hash function h_3 satisfies Property 1 and Property 2.*

Proof. First, note that Property 2 is satisfied. Only $a_1, \ldots, a_k$ need to be stored, and $h(x)$ is computed with a simple sum. Proving Property 1 is more challenging.

Suppose $x \neq y$ and $x, y \in U$.

$$\begin{aligned}\Pr[h(x) \neq h(y)] &= \Pr\left[\left(\sum_i a_i \cdot x_i\right) \equiv \left(\sum_i a_i \cdot y_i\right) \mod p\right] \\ &= \Pr\left[\sum_i (a_i x_i - a_i y_i) \equiv 0 \mod p\right]. \end{aligned} \tag{1}$$

by definition of h. Because $x \neq y$, there is an index j such that $x_j \neq y_j$ (i.e., $(x_j - y_j) \neq 0$). Continuing from 1, we have

$$\Pr\left[\sum_i (a_i x_i - a_i y_i) \equiv 0 \mod p\right] = \Pr\left[\sum_{i \neq j} (a_i (x_i - y_i) + a_j (x_j - y_j)) \equiv 0 \mod p\right]. \tag{2}$$

Next, fix all a_i for $i \neq j$, and let

$$\alpha = \left(-\sum_{i \neq j} a_i (x_i - y_i)\right) \mod p.$$

α is a fixed number, and $\alpha \in F_p$. Let $\beta = x_j - y_j$ and note that $\beta \neq 0$. Continuing 2 gives

$$\begin{aligned}\Pr\left[\sum_{i \neq j} (a_i (x_i - y_i) + a_j (x_j - y_j)) \equiv 0 \mod p\right] &= \Pr[a_j (x_j - y_j) \equiv \alpha \mod p] \\ &= \Pr[a_j \beta \equiv \alpha \mod p].\end{aligned}$$

Multiplying all the elements of a field by some set nonzero element gives a permutation of the field. Since a_j is chosen uniformly at random from F_p, we get

$$\Pr[a_j \beta \equiv \alpha \mod p] = \frac{1}{p}.$$

Therefore, Property 1 is satisfied. □