CP8201/CPS815 Advanced Algorithms

Assignment 4

Rohaan Ahmed
PhD Student
Department of Computer Science
rohaan.ahmed@ryerson.ca

November 27, 2020

Instructor: Dr. Javad (Jake) Doliskani
November 27, 2020

1 Using a randomized algorithm T to solve the majority problem

The function T gives a correct answer with probability $\frac{1}{2} + \frac{1}{100}$, therefore:

$$p(correct) = p_c = \frac{1}{2} + \frac{1}{100} = \frac{51}{100} = 0.51 \tag{0.1}$$

The desired accuracy for our algorithm is $\geq 1 - 2^{-20}$. In other words, the desired probability of getting the "wrong" result from out algorithm of the majority problem is:

$$p_d = 1 - (1 - 2^{-20}) = 2^{-20} \tag{0.2}$$

Our desire is to call the algorithm T a total of c times such that our probability of getting the wrong solution of the majority problem is $\leq p_d$. Note that since T's accuracy of $p_c = 0.51$ is only marginally better than a coin toss (0.50), we should expect the number of iteratations required (c) to be quite high.

We will use the multiplicative form of the Chernof Bound to find c, found on Wikipedia.

Multiplicative Chernoff Bound: Suppose $X_1, ..., X_n$ are independent random variables taking values in $0, 1$. Let X denote their sum and let $\mu = E[X]$ denote the sum's expected value. Then for any $\delta > 0$,

$$P(X \leq (1-\delta)\mu) \leq exp(\frac{\delta^2 \mu}{2}), \quad 0 \leq \delta \leq 1 \tag{0.3}$$

Setting up the Chernoff Bound

1. Since want the probability of getting the wrong answer to be $\leq 2^{-20}$, we set the above equation to be:

$$P(X \leq (1-\delta)\mu) \leq exp(\frac{\delta^2 \mu}{2}) \leq p_d, \quad p_d = 2^{-20} \tag{0.4}$$

2. The expected value of X is μ, and is:

$$\mu = E[X] = cp_c, \quad p_c = \frac{51}{100} \tag{0.5}$$

3. After c iterations, we want the majority of the outcomes to be correct. Or, alternatively, we want the total number of wrong outcomes after c iterations to be $\leq \frac{c}{2}$. Thus:

$$(1-\delta)\mu = \frac{c}{2} \tag{0.6}$$

$$\implies \delta = 1 - \frac{1}{2p_c} \tag{0.7}$$

Solving for c

Substituting the above values into the Chernoff Bound equation, we solve for c:

$$P(X \leq (1-\delta)\mu) \leq 2^{-20} \tag{0.8}$$

$$P\left(X \leq \frac{c}{2}\right) \leq 2^{-20} \tag{0.9}$$

$$exp\left(\left(\frac{-cp_c}{2}\right)\left(1-\frac{1}{2p_c}\right)^2\right) \leq 2^{-20} \tag{0.10}$$

$$\implies \left(\frac{-cp_c}{2}\right)\left(1-\frac{1}{2p_c}\right)^2 \leq -20\ln(2) \tag{0.11}$$

$$... \tag{0.12}$$

$$c \geq (100)(51)(40)(\ln(2)) \tag{0.13}$$

$$c \geq 141,402.0248 \tag{0.14}$$

$$c := 141,403 \tag{0.15}$$

Randomized Algorithm:

The algorithm below calls the function T a constant c number of times and then returns, with a very high probability ($\geq 1-2^{-20}$), whether a majority element exists in A (1) or not (0):

Function: ***majority_problem(*A, T*)***

```
cnt = 0
for i = 1, ..., c do
  result = T(A) {// Call function T on array A}
  if result = 1 then
    cnt := cnt + 1
  end if
end for
if cnt > (c/2) then
  return 1 {// A majority element exists in A}
else
  return 0 {// A majority element does not exist in A}
end if
```

Using a Probability Tree to find c

Another way of finding the number of iterations required (c) would be to use a Probability Tree diagram, as the one shown below. In this technique, we would be required to find the "mean" of all the paths that lead to the correct final answer, i.e., where T returns true in at least $\frac{c}{2}$ of the iterations *AND* our certainty is $\geq 1-2^{-20}$.

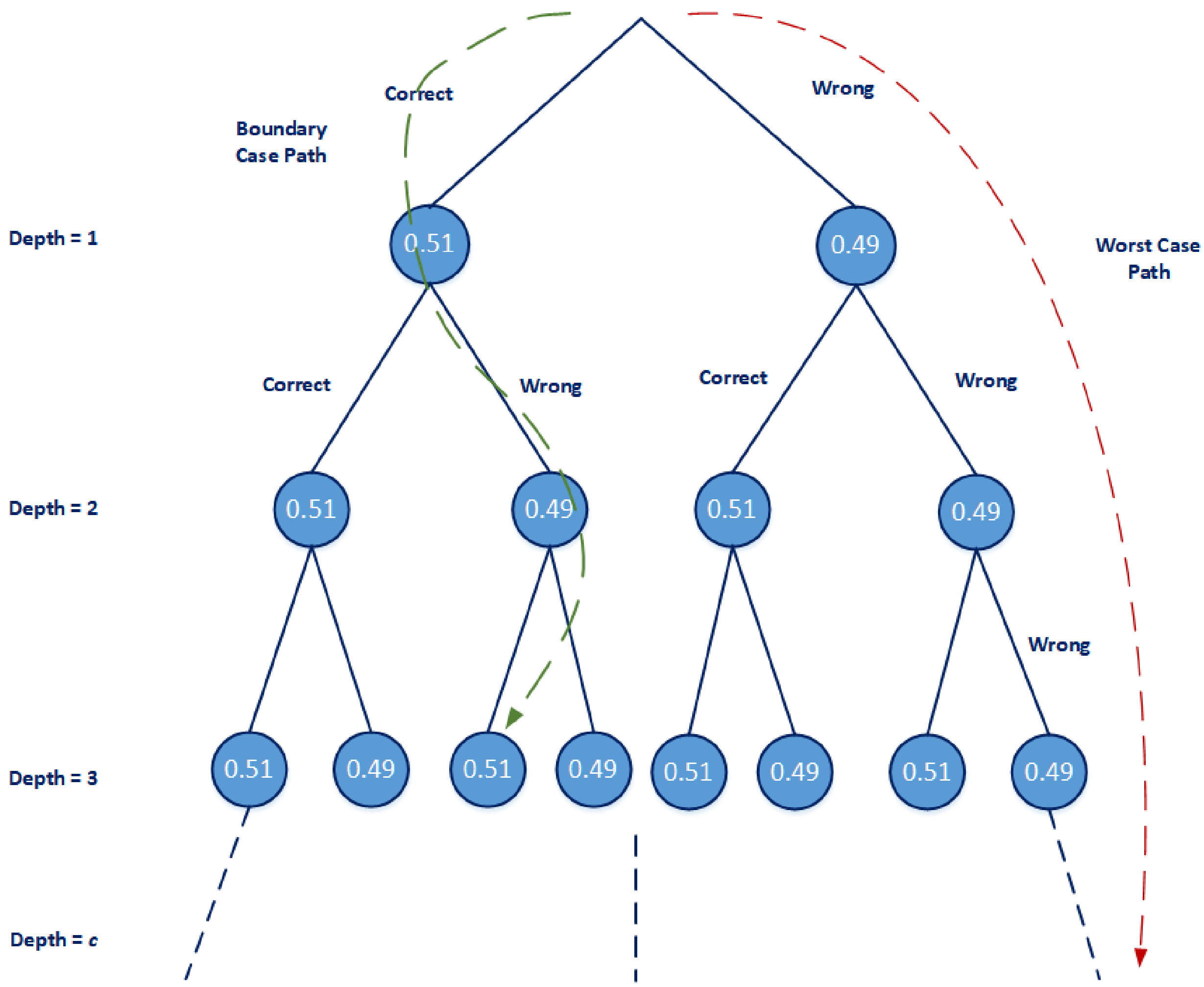

Figure 1: Probability Tree

In the image above, the Red and Green paths show two possible outcomes from our algorithm. The "Worst Case Path" shown in Red represents a scenario where T outputs the wrong answer every time, whereas the "Boundary Case Path" in Green represents one of many paths in which T outputs correctly $\geq \frac{c}{2}$ times. The objective of this technique would be to take the average of all "boundary paths"

The author of this report finds the tree diagram technique easier to visualize when compared with Chernoff Bounds. However, due to time constraints, using this technique to find c is left as a future exercise.

2 Proof that $\mathcal{H} = \{f_M | M \in \mathcal{M}_{n,m}\}$ is a Universal Family of Hash Functions

We will use a similar approach to the one used in the lecture notes and textbook to prove that the following set is a Universal Family of Hash Functions:

$$\mathcal{H} = \{f_M | M \in \mathcal{M}_{n,m}\} \tag{0.16}$$

Where:

$$f_M = \begin{bmatrix} h_1(x) \\ h_2(x) \\ \vdots \end{bmatrix} \tag{0.17}$$

We will do this by showing that, for any distinct keys $x_1, x_2, \ldots$, the following is true for some prime number $P \ni |S| < P < 2|S|, S \subset U$:

$$Pr[h(x_i) = h(x_j)] \leq 1/P \tag{0.18}$$

Note: The key difference between the proof below and the one in the lecture notes / textbook is that we select n random numbers a in total, one for each hash function in f_M. Once this is done, the remainder of the proof is simply a vectorization of the one found in the lecture notes / textbook.

Let $x = [x_1, x_2, ..., x_k]$ and $y = [y_1, y_2, ..., y_k]$ be two distinct elements of U, where $x_i, y_i \in \{0, 1, ..., P-1\}$.

$$h(x) = \sum_{i=1}^{k} a_i x_i \mod P \tag{0.19}$$

$$h(y) = \sum_{i=1}^{k} a_i y_i \mod P \tag{0.20}$$

Note that since $x \neq y$, there must exist some $i \ni x_i \neq y_i$. This is necessary for invertability.

For each hash function h we select a randomly, for a total of n randommly selected a's giving $a_1, a_2, ..., a_n$. We can then say, as in the course notes, that for each hash function, h_k, in f_M, there is a collision between x and y if and only if the following equation is true:

$$h_k(x) = h_k(y) \mod P \tag{0.21}$$

$$a_{k,j}(y_j - x_j) = \sum_{i \neq j} a_{k,i}(x_i - y_i) \mod P \tag{0.22}$$

Since P is prime, $h_k(x) = h_k(y) \mod P$ has at most one solution among P possibilities (based on the lemma presented in the lectures / textbook).

$$Pr[h_k(x) = h_k(y)] \leq \frac{1}{P} \quad \forall k \tag{0.23}$$

Since there are n such hash functions, i.e., h_k for $k = 1, 2, \ldots, n$, the total probability of collision reduces further:

$$Pr[f_M(x) = f_M(y)] = Pr[h(x) = h(y)] \quad \leq \quad \frac{1}{P} \times \frac{1}{P} \times \frac{1}{P} \cdots = \frac{1}{P^n} \tag{0.24}$$

Therefore, $\mathcal{H} = \{f_M | M \in \mathcal{M}_{n,m}\}$, which is a vector of Universal Family of Hash Functions f_M, is also itself a Unversal Family of Hash Functions that maps $\mathbb{Z}_p^{n \times m} \to \mathbb{Z}_p^n$.