## 1 Chernoff basics

- (10 points) Use the Chernoff bounds to prove that, after flipping a fair coin $n$ times, the number of heads will be less than $0.6n$ w.h.p. (Yes, we did this in class, but I want to make sure you have a good handle on the basics)

- (10 points) How many times must we flip a coin to get $\Omega(n)$ heads with high probability?

### Solution

- Let $p$ be the probability of getting heads, $X_i$ the random variable that equals to 1 when the $i$-th flip is heads and 0 otherwise, and $X = \sum X_i$ the total number of heads. For a single flip,

$$E[e^{tX_i}] = (1 - p) + pe^t \leq e^{p(e^t - 1)}$$

Therefore

$$E[e^{tX}] \leq e^{np(e^t - 1)}$$

For any $\delta > 0$, take $t = \ln(1 + \delta) > 0$ and $a = (1 + \delta)np$, then

$$\Pr[X \geq a] \leq \frac{E[e^{tX}]}{e^{ta}} \leq \left(\frac{e^\delta}{(1 + \delta)^{(1+\delta)}}\right)^{np}$$

Let $p = 1/2, \delta = 1/5$ so that $a = 0.6n$. For $0 < \delta < 1$

$$\delta < (1 + \delta) \ln(1 + \delta) \Rightarrow \frac{e^\delta}{(1 + \delta)^{1+\delta}} < 1$$

Thus

$$\Pr[X \geq 0.6n] \to 0 \quad (n \to \infty)$$

- Use the argument above we deduce that with $n$ flips the number of tails will be fewer than $kn$ w.h.p. ($k > 1/2$). So $n$ flips will have $(1-k)n = \Omega(n)$ heads w.h.p.
2 Cascading hashing

Consider the following cascaded hashing scheme. We have $\Theta(\log n)$ hash tables, $A_0, \ldots, A_{c \log n - 1}$. Table $A_i$ has $2^i$ buckets. Each hash table uses an independent hash function, $h_i$. The insert operation works as follows:

\[
\text{insert}(x) \\
\text{for } i = c \log n, \ldots, 0 \\
\quad \text{if not } A_i[h_i(x)].\text{occupied} \\
\quad \quad A_i[h_i(x)].\text{occupied} = \text{true} \\
\quad \quad A_i[h_i(x)].\text{value} = x \\
\quad \text{return OK} \\
\text{return ERROR}
\]

(20 points) Prove that inserting $n$ items into an empty cascading hash table with $c \log n$ levels will succeed w.h.p. (i.e. insert() will not return ERROR for any of the inserts).

Solution Problem withdrawn.

3 Probabilistic Matrix Equality Testing

Suppose you are given three $n \times n$ matrices, $A$, $B$, and $C$, and you want to test whether $AB = C$.

- (5 points) Suppose $v$ is a vector whose entries are each 0 or 1 with probability 1/2. Prove that, if $M$ is any non-zero $n \times n$ matrix, then $\Pr[Mv = 0] \leq 1/2$.

- (5 points) Prove that if $AB \neq C$, then for a random vector $v$ as above, $\Pr[ABv = Cv] \leq 1/2$.

- (5 points) Give a randomized algorithm for determining whether $AB = C$. Your algorithm may have a small error probability (i.e. it may occasional say that $AB = C$ even though they are not equal). However, your algorithm should output the correct answer w.h.p.

- (5 points) What is the running time of your algorithm?

Solution

- Suppose $m_{ij} \neq 0$. Take the product of the $i$-th row of $M$ and $v$

\[
p_i = \sum m_{ik}v_k = m_{ij}v_j + y
\]

where $y$ is the sum of other terms. Using Bayes’ theorem

\[
\Pr[p_i = 0] = \Pr[p_i = 0|y = 0] \Pr[y = 0] + \Pr[p_i = 0|y \neq 0] \Pr[y \neq 0]
\]

2
where

\[ \Pr[p_i = 0|y = 0] = \Pr[v_j = 0] = \frac{1}{2} \]

\[ \Pr[p_i = 0|y \neq 0] = \Pr[v_j \neq 0 \land m_{ij} = -y] \leq \frac{1}{2} \]

Therefore

\[ \Pr[p_i = 0] \leq \frac{1}{2} \Pr[y = 0] + \frac{1}{2}(1 - \Pr[y = 0]) = \frac{1}{2} \]

• Let \( M = AB - C \).
• Generate \( k \) random vectors \( \{v_i\} \) and test if \((AB - C)v_i = 0\) for all \( i \).
• \( O(n^2k) \).

4 Skip Lists with Rank-based Lookups

Consider adding a field, size, to each node in a skip list. The meaning of size is as follows: If two consecutive nodes \( n_1 \) and \( n_2 \) in the same level have \( \text{size}_1 \) and \( \text{size}_2 \), respectively, then the number of elements in the bottom level strictly between \( n_1 \) and \( n_2 \) is \( \text{size}_1 \).

(10 points) Show how to maintain the size fields when inserting a new element in the skip list. (You don’t need to provide pseudo-code or a proof – a picture with a couple of sentences making your idea clear is sufficient).

(10 points) Explain how to find the \( i \)th largest element of the skip list using the size fields. Pseudo-code is probably the simplest way to explain this algorithm clearly.

Solution For convenience, assume the size count is left exclusive, right inclusive.

• Go over each level from bottom to top. If the new element is present in the current level, update the size of the nearest promoted node on the left (LP) with the sum of size’s from LP to this element (exclusive) in the lower level; update the size of this node with the sum of size’s from this element to the nearest promoted element on the right (exclusive) in the lower level.

If the new element is not promoted to this level, increase the size of LP by 1.

• Traverse the top level and add up the size’s along the way; if the sum exceeds \( i \), subtract the last size, go down one level and continue.
5 Merging Skip Lists

(20 points) Write an algorithm to merge two skip lists into a single skip list. If it helps, you may assume the following data structures are used to represent the skip lists:

```c
struct skip_list_node {
  struct skip_list_node *next, *child;
  void *data;
};

struct skip_list {
  int nlevels;
  struct skip_list_node *levels[nlevels];
};
```

(Note: it looks like C, but it’s not C – you only need to write pseudo-code for this assignment.)

Solution Merge the linked list at each level.

6 Universal Hashing

We construct a family of hash functions mapping integer keys to \{0, 1, ..., m - 1\}.

1. Select \(m\) to be prime.
2. Decompose key \(k\) into \(r + 1\) digits: \(k = \langle k_0, k_1, ..., k_r \rangle\) where \(k_i \in \{0, 1, ..., m - 1\}\) (equivalent to writing key \(k\) in base \(m\))
3. Pick \(a = \langle a_0, a_1, ..., a_r \rangle\) at random. Each \(a_i \in \{0, 1, ..., m - 1\}\)
4. Then the hash function is \(h_a(k) = \left( \sum_{i=0}^{r} a_i k_i \right) \mod m\)

Your goal is to prove that this family of hash functions \(H\) is 2-universal. Follow the steps below and for each one give an explanation/proof.

a) (5 points) What is the total number of hash functions in the family? \(|H| = \) 

b) (10 points) Given distinct keys \(x\) and \(y\), how may hash functions cause \(x\) and \(y\) to collide? In other words, how many values of \(a\) make \(h_a(x) = h_a(y)\)? Hint: This is somewhat similar to the matrix family of hash functions we discussed in class. Recall that in that example we made the analysis simpler by assuming the keys differed in a single bit.

c) (5 points) Based on the previous two parts, use the definition to show \(H\) is 2-universal.
Solution

a) \( |H| = m^{r+1} \) since each of the \( r + 1 \) \( a_i \)'s has \( m \) possible values.

b) Let \( x = \langle x_0, x_1, \ldots, x_r \rangle \) and \( y = \langle y_0, y_1, \ldots, y_r \rangle \). \( x \neq y \) so they must differ in at least one digit. WLOG assume they differ in the first digit. That is, \( x_0 \neq x_1 \).

\[ h_a(x) = h_a(y) \]
\[ \Rightarrow (\sum_{i=0}^{r} a_ix_i) \mod m = (\sum_{i=0}^{r} a_iy_i) \mod m \]
\[ \Rightarrow \sum_{i=0}^{r} a_ix_i \equiv_m (\sum_{i=0}^{r} a_iy_i) \]
\[ \Rightarrow \sum_{i=0}^{r} a_ix_i - \sum_{i=0}^{r} a_iy_i \equiv_m 0 \]
\[ \Rightarrow \sum_{i=0}^{r} a_i(x_i - y_i) \equiv_m 0 \]
\[ \Rightarrow a_0(x_0 - y_0) + \sum_{i=1}^{r} a_i(x_i - y_i) \equiv_m 0 \]
\[ \Rightarrow a_0(x_0 - y_0) \equiv_m -\sum_{i=1}^{r} a_i(x_i - y_i) \]
\[ \Rightarrow a_0(x_0 - y_0)(x_0 - y_0)^{-1} \equiv_m (-\sum_{i=1}^{r} a_i(x_i - y_i))(x_0 - y_0)^{-1} \]
\[ \Rightarrow a_0 \equiv_m (-\sum_{i=1}^{r} a_i(x_i - y_i))(x_0 - y_0)^{-1} \]

\( (x_0 - y_0)^{-1} \) exists because \( (x_0 - y_0) \) is non-zero and \( m \) is prime.

The above congruency implies that once the values of \( a_1, \ldots, a_r \) are selected, exactly one value of \( a_0 \) causes \( x \) and \( y \) to collide. Counting the possible values of \( a_0 \)'s that cause \( x \) and \( y \) to collide gives us the number of \( h_a \)'s we have in \( H \) that cause \( x \) and \( y \) to collide. This equals the number possible combinations of \( a_1, \ldots, a_r \) which is \( m^r \) since each of the \( r \) digits has \( m \) possible values.

c) For all \( x, y \) where \( x \neq y \), the probability that \( x \) and \( y \) collide is

\[
\frac{\text{number of functions that cause collisions}}{\text{total number of functions}} = \frac{m^r}{m^{r+1}} = \frac{1}{m}
\]

so \( H \) is 2-universal.

7 Parallel Addition

Function Standard-Binary-Addition is the standard grade school algorithm for adding two \( n \)-bit binary numbers, and Rec-Binary-Addition is a recursive divide-and-conquer algorithm for the same task. In this problem we will try to parallelize the two algorithms and analyze the resulting parallel algorithms. (5 points each)

a) Can you replace the for loop in lines 2-4 of Standard-Binary-Addition with a parallel for loop (without changing anything else)? Justify your answer.

b) Which parts of Rec-Binary-Addition can be executed in parallel? Justify your answer.
c) Write down the recurrence relations for work ($T_1$) and span ($T_\infty$) for your parallel version of Rec-Binary-Addition from part b, and solve them. Assume that the span of a parallel for loop with $n$ iterations is $\Theta(\log n) + k$, where $k$ is the maximum span of one iteration.

d) Find the parallel running time ($T_p$) and the parallelism of the parallel Rec-Binary-Addition from part b.

e) Let $T_S$ be the runtime of the optimal or fastest known serial algorithm. A parallel algorithm is cost-optimal or work-optimal provided $pT_p = \Theta(T_S)$. Is your parallel Rec-Binary-Addition work-optimal? Justify your answer.

Solution
a) No, each iteration of the loop depends on \( c \) which is decided in the previous iteration.

b) The recursive calls in lines 8 and 9 can be done in parallel. Add a spawn to the beginning of line 9. You can also do Copy in parallel by changing the for-loop to be a parallel for-loop.

c) Work is \( T_1(n) = 2T_1(\frac{n}{2}) + O(n) \) because there are 2 subproblems each half as large and then we spend \( O(n) \) time to Copy. Using master theorem case 2, \( T_1(n) = \Theta(n \log n) \).

Span is \( T_\infty(n) = T_\infty(\frac{n}{2}) + \Theta(\log n) \) because both subproblems are worked in parallel and the span of the parallel for-loop in copy is \( \Theta(\log n) \). Using master theorem case 2, \( T_\infty(n) = \Theta(\log^2 n) \).

If you don’t parallelize Copy then \( T_\infty(n) = T_\infty(\frac{n}{2}) + \Theta(n) = \Theta(n) \) by master theorem case 3. So parallelizing Copy is important.

d) Parallelism is \( \frac{T_1(n)}{T_\infty(n)} = \frac{\Theta(n \log n)}{\Theta(\log^2 n)} = \frac{n}{\log n} \).

Parallel runtime is \( T_p(n) = O\left(\frac{T_1(n)}{p} + T_\infty(n)\right) = O\left(\frac{n \log n}{p} + \log^2 n\right) \).

e) \( T_S(n) = \Theta(n) \) from Standard-Binary-Addition and \( pT_p(n) = O(n \log n + p \log^2 n) \neq \Theta(n) \) so this algorithm is not work optimal.