CSE 548 Spring 2016: Homework #2

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?

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:

```plaintext
text
```

(20 points) Prove that inserting \( n \) items into an empty cascading hash table with \( A_0 \) of size \( n \) will succeed w.h.p. (i.e. insert() will not return ERROR for any of the inserts).

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 \( \Pr[M v = 0] \leq 1/2 \).

• (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 occasionally 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?

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 $size_1$ and $size_2$, respectively, then the number of elements in the bottom level strictly between $n_1$ and $n_2$ is $size_1$.

(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.

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.)

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: 
\[
k = \langle k_0, k_1, \ldots, k_r \rangle \text{ where } k_i \in \{0, 1, \ldots, m-1\} \text{ (equivalent to writing key } k \text{ in base } m)\]

3. Pick \( a = \langle a_0, a_1, \ldots, a_r \rangle \) at random. Each \( a_i \in \{0, 1, \ldots, m-1\} \)

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.

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 \( \Theta(n) + k \), 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.
Standard-Binary-Addition: 
\[ x_0 \ldots x_1, y_0 \ldots y_1 \] 
(Inputs are two \( n \)-bit binary numbers \( X = x_0 \ldots x_1 \) and \( Y = y_0 \ldots y_1 \), where \( n \geq 1 \). Output is an \( (n+1) \)-bit binary number \( Z = z_{n+1} \ldots z_1 \), where \( Z \) is the sum of \( X \) and \( Y \) (i.e., \( Z = X + Y \)).)

1. \( c \leftarrow 0 \)
2. for \( i \leftarrow 1 \) to \( n \) do
3. \( s \leftarrow x_i + y_i + c \)
4. \( z_i \leftarrow s \mod 2 \), \( c \leftarrow s \div 2 \)
5. \( z_{n+1} \leftarrow c \)
6. return \( Z \)

Recursive-Binary-Addition: 
\[ x_0 \ldots x_1, y_0 \ldots y_1 \] 
(Inputs are two \( n \)-bit binary numbers \( X = x_0 \ldots x_1 \) and \( Y = y_0 \ldots y_1 \), where \( n \geq 1 \) is assumed to be a power of 2. Outputs are two \( (n+1) \)-bit binary numbers \( Z' = z_{n+1} \ldots z_1 \) and \( Z = z_{n+1} \ldots z_1 \), where \( Z' \) is the sum of \( X \) and \( Y \) assuming an initial carry (i.e., \( Z' = X + Y + 1 \)), and \( Z \) is the same sum without an initial carry (i.e., \( Z = X + Y \)).)

1. if \( n = 1 \) then
2. if \( x_1 = y_1 = 0 \) then return \( (01, 00) \)
3. elseif \( x_1 = y_1 = 1 \) then return \( (11, 10) \)
4. else return \( (10, 01) \)
5. end

6. let \( X_0 = x_0 \ldots x_{\frac{n}{2}+1} \) and \( X_1 = x_{\frac{n}{2}} \ldots x_1 \) \{split \( X \) at the midpoint\}
7. let \( Y_0 = y_0 \ldots y_{\frac{n}{2}+1} \) and \( Y_1 = y_{\frac{n}{2}} \ldots y_1 \) \{split \( Y \) at the midpoint\}
8. \( (L', L) \leftarrow \text{Recursive-Binary-Addition}(X_0, Y_0) \) \{where \( L' = l'_{\frac{n}{2}+1}l'_{\frac{n}{2}} \ldots l'_1 \) and \( L = l_{\frac{n}{2}+1}l_{\frac{n}{2}} \ldots l_1 \}\}
9. \( (H', H) \leftarrow \text{Recursive-Binary-Addition}(X_1, Y_1) \) \{where \( H' = h'_{\frac{n}{2}+1}h'_{\frac{n}{2}} \ldots h'_1 \) and \( H = h_{\frac{n}{2}+1}h_{\frac{n}{2}} \ldots h_1 \}\}
10. if \( h_{\frac{n}{2}+1} = 1 \) then \( \text{Copy}(Z, H', l_{\frac{n}{2}} \ldots l_1) \) \{no carry from position \( \frac{n}{2} \)\} \( Z = h_{\frac{n}{2}+1}h_{\frac{n}{2}} \ldots h_1l_{\frac{n}{2}} \ldots l_1 \)
11. else \( \text{Copy}(Z, H, l_{\frac{n}{2}} \ldots l_1) \) \{no carry from position \( \frac{n}{2} \)\} \( Z = h_{\frac{n}{2}+1}h_{\frac{n}{2}} \ldots h_1l_{\frac{n}{2}} \ldots l_1 \)
12. if \( l'_{\frac{n}{2}+1} = 1 \) then \( \text{Copy}(Z', H', l'_{\frac{n}{2}} \ldots l'_1) \) \{carry from position \( \frac{n}{2} \)\} \( Z' = h'_{\frac{n}{2}+1}h'_{\frac{n}{2}} \ldots h'_1l'_{\frac{n}{2}} \ldots l'_1 \)
13. else \( \text{Copy}(Z', H, l'_{\frac{n}{2}} \ldots l'_1) \) \{no carry from position \( \frac{n}{2} \)\} \( Z' = h'_{\frac{n}{2}+1}h'_{\frac{n}{2}} \ldots h'_1l'_{\frac{n}{2}} \ldots l'_1 \)
14. return \( (Z', Z) \)

Copy: 
\[ z_{2n+1} \ldots z_1, h_{2n+1} \ldots h_1, l_{2n+1} \ldots l_1 \]

1. for \( i \leftarrow 1 \) to \( n \) do
2. \( z_i \leftarrow l_i \)
3. \( z_{2n+i} \leftarrow h_i \)
4. \( z_{2n+i} \leftarrow h_{n+i} \)