Homework 1

Zhehao Wang 404380075 (Dis 1D)

Apr 4, 2016

1 Binary addition algorithm correctness proof

The input number n can be denoted as $n = a_k ... a_0$ in binary, where $a_i = 0$, or $a_i = 1$ ($0 \leq i \leq k$, k is the most significant bit). The flipped number n' can be denoted as $n' = a'_k ... a'_0$ in binary. We have

$$n = \sum_{j=0}^{k} a_j \cdot 2^j \qquad \text{and} \qquad n' = \sum_{j=0}^{k} a'_j \cdot 2^j$$

Denote the position of first 0 in n from right to left to i, we have

$$n = \sum_{j=0}^{i-1} 1 \cdot 2^j + 0 \cdot 2^i + \sum_{j=i+1}^{k} a_j \cdot 2^j = \frac{2^i - 1}{2 - 1} + \sum_{j=i+1}^{k} a_j \cdot 2^j = 2^i - 1 + \sum_{j=i+1}^{k} a_j \cdot 2^j$$

and

$$n + 1 = 2^i + \sum_{j=i+1}^{k} a_j \cdot 2^j$$

After the flip in question, resulting number n' can be denoted as

$$n' = \sum_{j=0}^{k} a'_j \cdot 2^j = \sum_{j=0}^{i-1} 0 \cdot 2^j + 1 \cdot 2^i + \sum_{j=i+1}^{k} a'_j \cdot 2^j = 2^i + \sum_{j=i+1}^{k} a_j \cdot 2^j$$

Thus we have $n' = n + 1$, and the binary addition algorithm in question is correct.

2 Binary tree depth algorithm

Correctness: by definition, the depth of the tree is the longest path from root to a leaf. Thus the depth at each node is $1 + max(the depth of left subtree, the depth of right subtree)$. The algorithm is correct as it recursively traverses both the left and right subtree of each node, and returns the larger depth of the two.

Time complexity: this algorithm is $O(n)$, where n is the number of nodes in the tree. Reason being that each node (including *null* leaves) of the tree will be visited exactly once, and the number of instructions executed during each visit is constant. The number of *null* nodes will be $2n$ at maximum, so the algorithm visits at most $3n$ nodes, which makes it $O(n)$ by asymptotic notation.

3 Elementary-school-division algorithm

Time complexity:

Algorithm 1 Binary tree depth recursive

```
function maxDepth(node)
    if node = nil then
        return 0
    leftDepth ← maxDepth(node.leftChild)
    rightDepth ← maxDepth(node.rightChild)
    if leftDepth > rightDepth then
        return leftDepth + 1
    else
        return rightDepth + 1
```

Algorithm 2 Binary tree depth iterative

```
function getDepth(root)
    queue ← [root]
    height ← 0
    while True do
        nodeCount ← queue.size()
        if nodeCount = 0 then
            return height
        height ← height + 1
        while nodeCount > 0 do
            r ← queue.dequeue()
            if r.hasLeft() then
                queue.enqueue(r.left())
            if r.hasRight() then
                queue.enqueue(r.right())
            nodeCount ← nodeCount − 1
```

Algorithm 3 Elementary-school-division recursive

```
function division_r(a, b)
    if a < 10b then
        return a/b
    else
        temp ← division_r(a, 10b)
        return temp · 10 + division_r(a − temp ∗ 10 ∗ b, b)
```

Algorithm 4 Elementary-school-division iterative

```
function DIVISION_I(a, b)
    temp ← b
    stack ← []
    while a > 10 · temp do
        stack.push(temp)
        temp = 10 · temp
    res ← a/temp
    result ← res
    while !stack.isEmpty() do
        res ← a − temp · result
        temp ← stack.pop()
        res ← res/temp
        result ← 10 · result + res
    return result
```

4 NIM game

(a.1)

Theorem: for any favorable table, there exists a move that makes the table unfavorable.

Proof: denote the columns that have odd number of ones as $c_0...c_k$, where c_0 is the least signifcant column. There exist at least one row r whose number of matches m_r, when expressed in binary, has 1 in column c_k.

Flip m_r's bit in column $c_0...c_k$, we have a new number of matches m'_r. $m'_r < m_r$ since m_r has 1 in the most significant flipped bit (in column c_k), while m'_r has 0. By taking out $m_r - m'_r$ matches from row r, the number of ones in column $c_0...c_k$ is increased or decreased by 1 and the number of ones in other columns remains unchanged. Thus after this removal, all columns will have even numbers of ones, making the table unfavorable.

(a.2)

Theorem: for any unfavorable table, any move makes the table favorable for ones opponent.

Lemma: for any non-empty row r with m_r matches, removing $i = 1..m_r$ matches from row r will at least flip one column of m_r.

This lemma is trivial to prove: denote the number of matches after removing as m'_r, if after removing i matches, none of the bits of m_r changed, then $m_r = m'_r$, and $i = m'_r - m_r = 0$, which contradicts with $i = 1..m_r$.

Proof: by the above lemma, for any non-empty row r, any move will flip at least one column of m_r. Denote the flipped columns of m_r as $c_0..c_k$, the number of ones in columns $c_0...c_k$ will increase or decrease by 1, thus column $c_0..c_k$ will have odd number of ones, making the table favorable for the opponent.

(b)

Winning strategy: if there's only one row left, claim all the matches of that row. Otherwise, starting from a favorable board, pick a move that makes the board unfavorable for the opponent. Such a move always exists according to the proof in (a.1). When the board's unfavorable, whatever the opponent's move is, the board will become favorable again according to the proof in (a.2). The winning strategy is to repeat the steps above until claiming the last match.