CS250 Assignment 3 solutions

March 24, 2003

Question 1. [Masoumeh]

First relation

• (a) Using a recurrence tree we’ll have:

  Level: 0 Size of call: T(n) Total: c
  Level: 1 Size of call: T(2n/3) Total: c
  Level: 2 Size of call: T(4n/9) Total: c
  .
  .
  Level: k (depth of the tree) Size of call: T(1) Total: c

The depth of the tree can be calculated by \( T(1) = n \times (2/3)^K = 1 \) so:

\[
    n = (3/2)^k
\]

or

\[
    k = \log_{3/2} n
\]

the overall time would be \( \sum_{k=0}^{\log_{3/2} n} c < c \times \log n \in O(lg(n)) \)
• (b) Let \( n = (3/2)^k \) for some \( k \). Then
\[
T(n) = c \cdot \log_{3/2}(n) \quad \text{where } c \neq 1
\]
Proposition \( P(k) : T(3/2)^k = c \cdot \log_{3/2}(3/2)^k \)

Proof: basis
\[
(k = 1) : T(3/2)^1 = T(3/2) = T(1) + 1 = c = c \cdot \log_{3/2}(3/2)
\]
P(1) is true. Assume P(k) is true, show for P(k+1)

\[
T(3/2)^{k+1} = T((3/2)^k + 1) = c \cdot \log_{3/2}(3/2)^k + 1 = ck + c = c(k + 1) = c \cdot \log_{3/2}(n)
\]
\[
T(n) \in O(\log_{3/2}n) = O(lg(n))
\]

• (c) The general form of \( T(n) \) for using the Master Theorem is: \( T(n) = aT(n/b) + f(n) \). Therefore for \( T(n) = T(2n/3) + 1 \) we have \( a = 1, b = (3/2) \), and \( f(n) = 1 \)

compare \( f(n) \) with \( n^{\log_{3/2}1} = n^0 = 1 \)

\( f(n) = n^{\log_{3/2}1} = T(n^{\log_{3/2}1}) \), which is case 2

\[
T(n) = T(n^{\log_{3/2}1} \cdot \log(n)) = T(n^0 \cdot \log(n)) \in O(lg(n))
\]

Second relation

• (a) Using a recurrence tree we’ll have:

Level: 0 Size of call: \( T(n) \) Total: \( cn \)
Level: 1 Size of call: \( T(n/4) \) Total: 3 \( cn/4 \)
Level: 2 Size of call: \( T(n/16) \) Total: 9\( cn/16 \)
Level: k (depth of the tree) Size of call: T(1) Total: \((3/4)^k cn\)

The depth of the tree is calculated by: \(T(n) = T(1)\) so \(n/4^k = 1\) or

\[ k = \log_4 n \]

and there are \(3^k\) nodes for kth level of the tree, so the total time would be:

\[
\sum_{k=0}^{\log_4 n} c \cdot n \cdot (3/4)^k = c \cdot n \sum_{k=0}^{\log_4 n} (3/4)^k < c \cdot n \sum_{k=0}^{\infty} (3/4)^k
\]

using geometric series the total time is \(< 4cn \in O(n)\) since \(\log_4 3 < 1\).

- (b) let \(n = 4^k\) and prove by induction on k, that \(T(n) = c \cdot 4^k\) which leads to \(T(n) \in O(n)\). The base case holds for \(k=0\) and \(c=4\) since \(T(1) = 1\). Suppose inductively that \(T(4^k) = c \cdot 4^k\). The recurrence relation gives \(T(4^{k+1}) = 3T(4^k) + 4^{K+1} \leq 3 \cdot 4 \cdot 4^k + 4^{(k+1)} = 4 \cdot 4^{(k+1)}\).

- (c)

\[ a = 3, \ b = 4, \ f(n) = n, \ \text{compare} \ f(n) = n \ \text{with} \ n^{\log_4 3} \]

\[ n = n^{\log_4 3 + \epsilon}, \ \epsilon > 0 \ \text{which is case 3} \]

\[ 3f(n/4) = 3n/4 \leq c^* n, \ c \geq 0 \]

\( T(n) = T(f(n)) \in O(n) \)

Question 3. [Irina]

```java
import java.util.*;

//Question 3: proposed solution
```
```java
/**
 * NOTE: Provided solution does not recognize all invalid strings!
 * for example 1 2 \ passes ok
 * In order to be able to detect all invalid strings and throw an exception
 * we may want to add a parsing method
 */

class Question3 {

    static String delimiters = "\*%()"; // valid non-digit characters
    static String[] binaryOps = {"\", "+", "/"};
    // valid binary operators

    public Question3() {
    }

    /**
     * Returns whether or not our token is a recognized operator
     */
    boolean isBinaryOperator(String op) {

        for (int i = 0; i < binaryOps.length; i++)
        {
            if (binaryOps[i].compareTo(op) == 0)
                return true;
        }

        return false;
    }

```
/**
 * Returns whether or not it is an opening bracket
 */
boolean isOpeningBracket(String op){
    return (op.compareTo("(") == 0);
}

/**
 * Returns whether or not it is a closing bracket
 */
boolean isClosingBracket(String op){
    return (op.compareTo("") == 0);
}

/***
 * Given a recognized token (an operator or bracket) returns its priority
 *
 */
int getPriority(String operator) throws Exception{
    if (operator.compareTo("") == 0) return 4;
    if (operator.compareTo("\^") == 0) return 3;
}
if (operator.compareTo("\n") == 0) return 2;                 // same priority
if (operator.compareTo("\%") == 0) return 2;
if (operator.compareTo("\(\)") == 0) return 1;

throw new Exception(" unidentified operator ");
}

/***
 * Evaluates a binary operator;
 *
*/
Integer evaluate(String operator, Integer num1, Integer num2) throws Exception{

    if (operator.compareTo("\^\") == 0)
        return new Integer( (int) Math.pow((double)num1.intValue(),
                                          (double) num2.intValue()));

    if (operator.compareTo("\*\") == 0)
        return new Integer(num1.intValue() * num2.intValue());

    if (operator.compareTo("\%\") == 0)
        return new Integer(num1.intValue() % num2.intValue());

    throw new Exception(" unidentified operator ");
}

/***
 *
* numbers we always push onto the stack
* operations we pull everything that has the same or bigger priority from the stack and
  * evaluate them;
*/

```java
public int evaluateExpression(String expression) throws Exception{

    arrayStack numStack = new arrayStack(20);
    arrayStack opStack = new arrayStack(20);

    Stringtokenizer tokenizer = new Stringtokenizer(expression, delimeters, true);

    String next, prevOperator;
    Integer num1, num2;

    int next_priority, prev_priority;

    while (tokenizer.hasMoreTokens()) {
        next = tokenizer.nextToken();

        if (isClosingBracket(next))
            {
                // until we pull out an opening bracket
                while (! isOpeningBracket( (String) opStack.top() ) )
                    {
                        prevOperator = (String) opStack.pop();
                        num2 = (Integer) numStack.pop();
                        num1 = (Integer) numStack.pop();
```
numStack.push(evaluate(prevOperator, num1, num2));

}

System.out.println(next + getPriority(next));

opStack.pop(); // remove that opening bracket
}

else if (isOpeningBracket(next))
{
    opStack.push(next);

    System.out.println(next + getPriority(next));
}

else if (isBinaryOperator(next)) // check if it is a binary operator
{
    next_priority = getPriority(next);

    while (!opStack.isEmpty() )
        &&
        ( getPriority((String) opStack.top() ) ≥
        next_priority ) )
    {
        prevOperator = (String) opStack.pop();
        num2 = (Integer) numStack.pop();
        num1 = (Integer) numStack.pop();

        numStack.push(evaluate(prevOperator, num1, num2));
    }
opStack.push(next);

System.out.println(next + getPriority(next));

} else if (next.compareTo(" ") == 0) { // space
else {
    // it's a number
    // push it on the stack
    numStack.push(new Integer(Integer.parseInt(next)));
}
}

// no more tokens; process what's remained on the stack

while (!opStack.isEmpty())
{
    prevOperator = (String) opStack.pop();
    num2 = (Integer) numStack.pop();
    num1 = (Integer) numStack.pop();
    numStack.push(evaluate(prevOperator, num1, num2));
}

// should be exactly one number left in the numStack
int result = ((Integer)numStack.pop()).intValue();

if (!numStack.isEmpty())
    throw new Exception("invalid expression");
return result;
}

public static void main(String[] argv) {

// NOTE: should implement a procedure that enables keyboard entry

String[][] expressions = {
    " 1 \ ( 7 * 5 ) ",
    " 2 * 3 \ 5 ",
    " 2 \ ( 7 * 5 \ 3 )",
    " ( 3 \ ( 35 \ 6 ) ) \ ( 7 * 5 )",
    " ( 3 * 1 ) \ ( 3 * 4 ) \ 21",
    " (3 \ ) \ 21"
};

Question3 Q3 = new Question3();

for (int i = 0; i < expressions.length; i++) {
    try {
        System.out.println("answer " + Q3.evaluateExpression(expressions[i]));
    } catch (Exception e) {
        System.out.println("invalid expression " + e);
    }
}
}