CSCI-GA.1170-003/004 Fundamental Algorithms	April 9, 2020

Solutions to Problem 2 of Homework 8 (16+12 points)

Name: Rahul Zalkikar (rz1567) *Due: 8 pm on Thursday, April 9*

Collaborators: ua388, jni215, nmd353

A boolean expression is one which evaluates to either True or False. It is typically composed of symbols T and F corresponding to True and False respectively along with boolean operators. For this question, we will restrict ourselves to binary boolean operators, such as $\wedge$ corresponding to AND, $\vee$ corresponding to OR, $\oplus$ corresponding to XOR, $\overline{\vee}$ corresponding to NOR, and $\overline{\wedge}$ corresponding to NAND. The evaluations are given below for your recollection:

Operand 1	Operand 2	$\vee$	$\wedge$	$\oplus$	$\overline{\vee}$	$\overline{\wedge}$
True	True	True	True	False	False	False
True	False	True	False	True	False	True
False	True	True	False	True	False	True
False	False	False	False	False	True	True

Note that while $\wedge, \vee, \oplus$ are associative $\overline{\vee}, \overline{\wedge}$ are not. Interestingly, an expression consisting of only associative operators is also not associative. For example, consider the expression

$$\text{False} \wedge \text{True} \vee \text{True}$$

Evaluating this expression from left to right yields the value True. However, if we evaluated from right to left, the expression evaluates to False. In other words, by carefully parenthesizing a given expression, i.e., True $\vee$ (False $\oplus$ True), we have managed to evaluate the expression to True.

Formally, assume that you are given a *valid* boolean expression ex in two arrays:

- $S[1, \ldots, n]$ such that $\forall\ i, S[i] \in \{\text{True}, \text{False}\}$, and
- $Op[1, \ldots, n-1]$ such that $\forall\ i, Op[i]$ is a binary operator.

such that $ex = s[1]||op[1]||s[2]||op[2]|| \ldots ||s[n-1]||op[n-1]||s[n]$ where $||$ denotes concatenation.

(a) (4 points) Let $C[n]$ denote the total number of ways to paranthesize the given expression with n operands and $n-1$ operators, as defined above. Write a recursive formulation for $C[n]$ using dynamic programming and argue its correctness. What would be the runtime of this algorithm?

Solution:

$$C[n] = \begin{cases} \sum_{i=1}^{n-1}(C[i] * C[n-i]) & \textbf{if } n \geq 2 \\ 1 & \textbf{if } n = 1 \end{cases}$$

Correctness:

Base Case: $C[1] = 1$, $C[2] = 1$

Inductive Assumption: Assume that $\forall\ n' > n$, the recursive formulation is correct.

Induction Step: We know the number of ways to parenthesize the first i operands multiplied by the number of ways to parenthesize the next $n-i$ operands equals the number of ways to parenthesize n operands for $1 \leq i < n$. So our formulation $\sum_{i=1}^{n-1}(C[i] * C[n-i])$ for $1 \leq i < n$ is correct.

Runtime:

Since we are doing $O(i)$ work to compute $C[i]$ the runtime is $O(\sum_{i=1}^{n-1} i) = O(n^2)$

$\square$

The goal of this question is to compute the number of ways one can parenthesize the expression to make it evaluate to True. We will use Dynamic Programming to formulate a recursive solution for the same. To this end, we will use one $n \times n$ matrix T defined as:

- $T[i][j]$ is the number of ways in which you can parenthesize the expression $ex' = S[i]||Op[i]||\ldots||Op[j-1]||S[j]$ such that it always returns True.

For the next part, assume that the *only* operator is $\overline{\vee}$.

(b) (6 points) Formulate a recursive equation based on dynamic programming for $T[i][j]$ and argue its correctness. What is the runtime of your algorithm? You may assume that the C array is already filled for you. (**Hint**: Some useful hints. We have True in a boolean world. We can think of its dual which is False. You may find it useful to subtract values from the values computed in part (a). Think about how to do it.)

Solution:

Let $F[i][j]$ be the number of ways in which you can parenthesize the expression ex' such that it always returns False.

We know that $F[i][j] = C[j-i+1] - T[i][j]$ since the total number of ways to parenthesize an expression with n operands is the sum of the number of ways for the expression to return True and False.

For the $\overline{\vee}$ operator we have $T[i][j] = \displaystyle\sum_{k=i}^{j-1} F[i][k] * F[k+1][j]$, since both sides of the expression must return False for the expression to return True.

Now we can see that $F[i][k] = C[k-i+1] - T[i][k]$ and $F[k+1][j] = C[j-k] - T[k+1][j]$

Finally,

$$T[i][j] = \begin{cases} \sum_{k=i}^{j-1}((C[k-i+1] - T[i][k]) * (C[j-k] - T[k+1][j])) & \textbf{if } i \neq j \\ 1 & \textbf{if } i = j \end{cases}$$

Correctness:

Base Case: $T[1][2] = (C[1] - T[1][1]) * (C[1] - T[2][2])$, which returns True only if both sides return False

Inductive Assumption: Assume that $\forall\ j' - i' > j - i$, the recursive formulation is correct.

Induction Step: We know the number of ways to parenthesize the first k operands such that they return False multiplied by the number of ways to parenthesize the next $j-k$ operands such that they return False equals the number of ways to parenthesize n operands such that the expression returns True for $i \leq k < j$. So our formulation $\sum_{k=i}^{j-1}((C[k-i+1] - T[i][k]) * (C[j-k] - T[k+1][j]))$ for $i \leq k < j$ is correct.

Runtime:

There are $O(n^2)$ subproblems, two for each side of the expression. The subproblem $T[i][j]$ must consider $O(j-i) = O(n)$ smaller subproblems. Thus the total runtime is $O(n^3)$.

$\square$

For this part, assume that the *only* operator is $\overline{\wedge}$. You will use a strategy similar to part (b) but have to be a little creative in how you formulate the recurrence relation. We will give you hints.

(c) (6 points) (**Extra Credit:**) Formulate a recursive equation based on dynamic programming for $T[i][j]$ and argue its correctness. What is the runtime of your algorithm? You may assume that the C array is already filled for you. (**Hint**: Some useful hints. You will use part (a) again but the strategy is a bit different from part (b). Note that $\overline{\wedge}$ evaluates to True if at least one of the operand is False. Or, *except when both evaluates to True.*)

Solution:

Let $F[i][j]$ be the number of ways in which you can parenthesize the expression ex' such that it always returns False.

We know that $T[i][j] = C[j-i+1] - F[i][j]$ since the total number of ways to parenthesize an expression with n operands is the sum of the number of ways for the expression to return True and False.

For the $\overline{\wedge}$ operator we have $F[i][j] = \displaystyle\sum_{k=i}^{j-1} T[i][k] * T[k+1][j]$, since both sides of the expression must return True for the expression to return False. In other words, at least one side of the expression must return False for the expression to return True

Finally,

$$T[i][j] = \begin{cases} C[j-i+1] - \sum_{k=i}^{j-1}(T[i][k] * T[k+1][j]) & \textbf{if } i \neq j \\ 1 & \textbf{if } i = j \end{cases}$$

Correctness:

Base Case: $T[1][2] = C[2] - (T[1][1] * T[2][2])$, which returns True only if at least one side returns False

Inductive Assumption: Assume that $\forall\ j' - i' > j - i$, the recursive formulation is correct.

Induction Step: We know the number of ways to parenthesize the first k operands such that they return True multiplied by the number of ways to parenthesize the next $j-k$ operands such that they return True, subtracted from the total number of ways to parenthesize the expression, equals the number of ways to parenthesize n operands such that the expression returns True for $i \leq k < j$. So our formulation $C[j-i+1] - \sum_{k=i}^{j-1}(T[i][k] * T[k+1][j])$ for $i \leq k < j$ is correct.

Runtime:

There are $O(n^2)$ subproblems, two for each side of the expression. The subproblem $T[i][j]$ must consider $O(j-i) = O(n)$ smaller subproblems. Thus the total runtime is $O(n^3)$.

□

In the previous parts we studied how to fill the table T when there existed only one kind of operator. In this question, we will look at the case when more than one type of operator can occur. More specifically, we assume that the expression can only contain operators $\vee, \wedge$.

(d) (6 points) Formulate a recursive equation based on dynamic programming for $T[i][j]$ and argue its correctness. What is the runtime of your algorithm? You may assume that the C array is already filled for you. (**Hint**: Combine strategies from parts (b), (c).)

Solution:

Let $F[i][j]$ be the number of ways in which you can parenthesize the expression ex' such that it always returns False.

We know that $T[i][j] = C[j-i+1] - F[i][j]$ since the total number of ways to parenthesize an expression with n operands is the sum of the number of ways for the expression to return True and False.

For the $\wedge$ operator we have $T[i][j] = \sum_{k=i}^{j-1} T[i][k] * T[k+1][j]$, since both sides of the expression must return True for the expression to return True.

For the $\vee$ operator we have $F[i][j] = \sum_{k=i}^{j-1} F[i][k] * F[k+1][j]$, since both sides of the expression must return False for the expression to return False. In other words, at least one side of the expression must return True for the expression to return True

Now we can see that $F[i][k] = C[k-i+1] - T[i][k]$ and $F[k+1][j] = C[j-k] - T[k+1][j]$.

Finally,

$$T[i][j] = \begin{cases} \sum_{k=i}^{j-1}(T[i][k] * T[k+1][j]) \\ \textbf{if } i \neq j \ \&\& \ Op[k] = \wedge \\ \\ C[j-i+1] - \sum_{k=i}^{j-1}((C[k-i+1] - T[i][k]) * (C[j-k] - T[k+1][j])) \\ \textbf{if } i \neq j \ \&\& \ Op[k] = \vee \\ \\ 1 \textbf{ if } i = j \end{cases}$$

Correctness (if $Op[k] = \wedge$):

Base Case: $T[1][2] = T[1][1] * T[2][2]$, which returns True only if both sides returns True

Inductive Assumption: Assume that $\forall\ j' - i' > j - i$, the recursive formulation is correct.

Induction Step: We know the number of ways to parenthesize the first k operands such that they return True multiplied by the number of ways to parenthesize the next $j-k$ operands such that they return True equals the number of ways to parenthesize n operands such that the expression returns True for $i \leq k < j$. So our formulation $\sum_{k=i}^{j-1}(T[i][k] * T[k+1][j])$ for $i \leq k < j$ is correct.

Correctness (if $Op[k] = \vee$):

Base Case: $T[1][2] = C[2] - (C[1] - T[1][1]) * (C[1] - T[2][2])$, which returns True only if at least one side returns True

Inductive Assumption: Assume that $\forall\ j' - i' > j - i$, the recursive formulation is correct.

Induction Step: We know the number of ways to parenthesize the first k operands such that they return False multiplied by the number of ways to parenthesize the next $j - k$ operands such that they return False, subtracted from the total number of ways to parenthesize the expression, equals the number of ways to parenthesize n operands such that the expression returns True for $i \leq k < j$. So our formulation $C[j-i+1] - \sum_{k=i}^{j-1}((C[k-i+1] - T[i][k]) * (C[j-k] - T[k+1][j]))$ for $i \leq k < j$ is correct.

Runtime:

There are $O(n^2)$ subproblems, two for each side of the expression. The subproblem $T[i][j]$ must consider $O(j - i) = O(n)$ smaller subproblems. Thus the total runtime is $O(n^3)$.

$\square$

(e) (6 points) (**Extra Credit:**) Assume that the expression can only contain operators $\vee, \wedge, \oplus$. Formulate a recursive relation for $T[i][j]$ and argue its correctness. What is the runtime of your algorithm? You may assume that the C array is already filled for you.

Solution:

Let $F[i][j]$ be the number of ways in which you can parenthesize the expression ex' such that it always returns False.

We know that $T[i][j] = C[j-i+1] - F[i][j]$ since the total number of ways to parenthesize an expression with n operands is the sum of the number of ways for the expression to return True and False.

For the $\oplus$ operator we have $T[i][j] = \displaystyle\sum_{k=i}^{j-1}(T[i][k] * F[k+1][j]) + (F[i][k] * T[k+1][j])$, since one side of the expression must return False and the other side of the expression must return True for the expression to return True

Now we can see that $F[i][k] = C[k-i+1] - T[i][k]$ and $F[k+1][j] = C[j-k] - T[k+1][j]$.

Finally,

$$T[i][j] = \begin{cases} \sum_{k=i}^{j-1}(T[i][k] * T[k+1][j]) \\ \textbf{if } i \neq j \text{ \&\& } Op[k] = \wedge \\ \\ C[j-i+1] - \sum_{k=i}^{j-1}((C[k-i+1] - T[i][k]) * (C[j-k] - T[k+1][j])) \\ \textbf{if } i \neq j \text{ \&\& } Op[k] = \vee \\ \\ \sum_{k=i}^{j-1}((T[i][k] * (C[j-k] - T[k+1][j])) + ((C[k-i+1] - T[i][k]) * T[k+1][j])) \\ \textbf{if } i \neq j \text{ \&\& } Op[k] = \oplus \\ \\ 1 \textbf{ if } i = j \end{cases}$$

We have already shown Correctness for our formulations if $Op[k] = \wedge$ or $Op[k] = \vee$

Correctness (if $Op[k] = \oplus$):

Base Case: $T[1][2] = (T[1][1] * (C[1] - T[2][2])) + ((C[1] - T[1][1]) * T[2][2])$, which returns True only if one side returns True and the other returns False.

Inductive Assumption: Assume that $\forall\ j' - i' > j - i$, the recursive formulation is correct.

Induction Step: We know the number of ways to parenthesize the first k operands such that they return True multiplied by the number of ways to parenthesize the next $j-k$ operands such that they return False, plus the number of ways to parenthesize the first k operands such that they return False multiplied by the number of ways to parenthesize the next $j-k$ operands such that they return True, equals the number of ways to parenthesize n operands such that the expression returns True for $i \leq k < j$. So our formulation $\sum_{k=i}^{j-1}((T[i][k]*(C[j-k] - T[k+1][j])) + ((C[k-i+1] - T[i][k]) * T[k+1][j]))$ for $i \leq k < j$ is correct.

Runtime:

There are $O(n^2)$ subproblems, two for each side of the expression. The subproblem $T[i][j]$ must consider $O(j-i) = O(n)$ smaller subproblems. Thus the total runtime is $O(n^3)$.

$\square$