ECE374 SP23 HW6

Contributors

Zhirong Chen (zhirong4)

Ziyuan Chen (ziyuanc3)

Problem 5

Expressions as Graphs.

(a) Suppose an arithmetic expression is given as a tree. Each leaf is an integer and each internal node is one of the standard arithmetical operations (+, -, *, /). Give an $O(n)$ algorithm for evaluating such an expression, where there are n nodes in the tree.

(b) Suppose an arithmetic expression is given as a DAG with common subexpressions removed. Each leaf is an integer and each internal node is one of the standard arithmetical operations (+, -, *, /). Give an $O(n+m)$ algorithm for evaluating such a DAG with n nodes and m edges.

Solution

Example Abstract Syntax Tree (AST) parsing algorithm by Steven S. Lumetta

(a)

$\text{EvalExpTree}(T)$

if T is an operator

$a \leftarrow \text{EvalExpTree}(T.\text{left})$

$b \leftarrow \text{EvalExpTree}(T.\text{right})$

return $a\ T\ b$

else

return T

This pseudocode is equivalent to using a stack to store the intermediate results.

- We initialize the stack by pushing the tree nodes in a **post-order** traversal.
- When we encounter an operator, we pop two operands, perform the operation, and push the result back.
- When we encounter an operand, we push it back.
- When we reach the base, the result is on the stack top.

Each node is visited once and pushed once, yielding time complexity of $O(n)$.

(b)

$\text{EvalExpDAG}(G)$

    if G is an operator
        $a \leftarrow \text{EvalExpDAG}(G.\text{left})$
        $b \leftarrow \text{EvalExpDAG}(G.\text{right})$
        $G \leftarrow a\ G\ b$ $\quad$ // G is now an operand
    return G

We memoize intermediate results in DAG nodes. Each time the "value" of an **operator node** is evaluated, the node is overwritten with the result and becomes an **operand node**.

Each node is visited once, but requesting the memoized values still involves traversing all the edges. Total time complexity of $O(n + m)$.