OPTICAL CODE GENERATION FOR EXPRESSIONS

- We can choose registers optimally when a basic block consists of a single expression evaluation, or if we accept that it is sufficient to generate code for a block one expression at a time.
- In the following algorithm, we introduce a numbering scheme for the nodes of an expression tree (a syntax tree for an expression) that allows us to generate optimal code for an expression tree when there is a fixed number of registers with which to evaluate the expression.

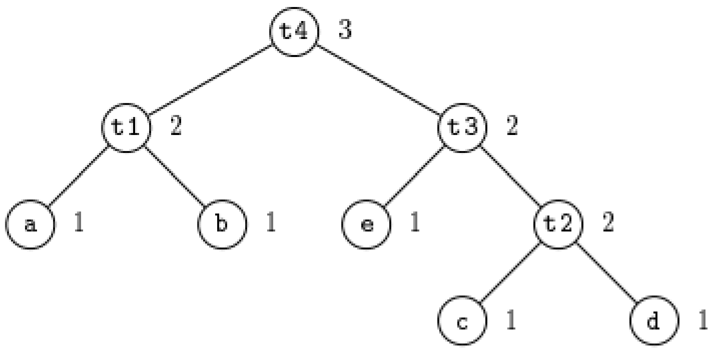
Ershov Numbers:

- We begin by assigning to each node of an expression tree a number that tells how many registers are needed to evaluate that node without storing any temporaries. These numbers are sometimes called Ershov numbers, after A.
- Ershov, who used a similar scheme for machines with a single arithmetic register.
- For our machine model, the rules are:
  1. Label all leaves 1.
  2. The label of an interior node with one child is the label of its child.
  3. The label of an interior node with two children is
     (a) The larger of the labels of its children, if those labels are different.
     (b) One plus the label of its children if the labels are the same.

Example: we see an expression tree (with operators omitted) that might be the tree for expression (a b) + e (c + d) or the three-address code

```
t1 = a - b
t2 = c + d
t3 = e * t2
t4 = t1 + t3
```

Each of the five leaves is labeled 1 by rule (1). Then, we can label the interior node for t1 = a - b, since both of its children are labeled. Rule (3b) applies, so it gets label one more than the labels of its children, that is, 2. The same holds for the interior node for t2 = c + d



Now, we can work on the node for t3 = e * t2. Its children have labels 1 and 2, so the label of the node for t3 is the maximum, 2, by rule (3a). Finally, the root, the node for t4 = t1 + t3, has two children with label 2, and therefore it gets label.

Generating code from labeled expression trees:

- It can be proved that, in our machine model, where all operands must be i registers, and registers can be used by both an operand and the result of an operation, the label of a node is the fewest registers with which the expression can be evaluated using no stores of temporary results.
- Since in this model, we are forced to load each operand, and we are forced to compute the result corresponding to each interior node, the only thing that can make the generate code inferior to the optimal code is if there are unnecessary stores of temporaries.
- The argument for this claim is embedded in the following algorithm for generating code with no stores of temporaries, using a number of registers equal to the label of the root.

Algorithm: Generating code from a labeled expression tree.

INPUT: A labeled tree with each operand appearing once (that is, no common subexpressions).

OUTPUT: An optimal sequence of machine instructions to evaluate the root into a register.

METHOD: The following is a recursive algorithm to generate the machine code. The steps below are applied, starting at the root of the tree. If the algorithm is applied to a node with label k, then only k registers will be used. However, there is a "base" $b \geq 1$ for the registers used so that the actual registers used are $R_b, R_{b+1}, \ldots, R_{b+k-1}$. The result always appears in R_{b+k-1}.

1. To generate machine code for an interior node with label k and two children with equal labels (which must be $k-1$) do the following:
   (a) Recursively generate code for the right child, using base $b+1$. The result of the right child appears in register R_{b+k-1}.
   (b) Recursively generate code for the left child, using base b; the result appears in R_{b+k-2}.
   (c) Generate the instruction OP $R_{b+k-1}, R_{b+k-2}, R_{b+k-1}$, where OP is the appropriate operation for the interior node in question.

2. Suppose we have an interior node with label k and children with unequal labels. Then one of the children, which we'll call the "big" child, has label k, and the other child, the "little" child, has some label $m < k$. Do the following to generate code for this interior node, using base b:
   (a) Recursively generate code for the big child, using base b; the result appears in register R_{b+k-1}.
   (b) Recursively generate code for the little child, using base b; the result appears in register R_{b+m-1}. Note that since $m < k$, neither R_{b+k-1} nor any higher-numbered register is used.
   (c) Generate the instruction OP $R_{b+k-1}, R_{b+m-1}, R_{b+k-1}$ or the instruction OP $R_{b+k-1}, R_{b+k-1}, R_{b+m-1}$, depending on whether the big child is the right or left child, respectively.

3. For a leaf representing operand x, if the base is b generate the instruction LD R_b, x.