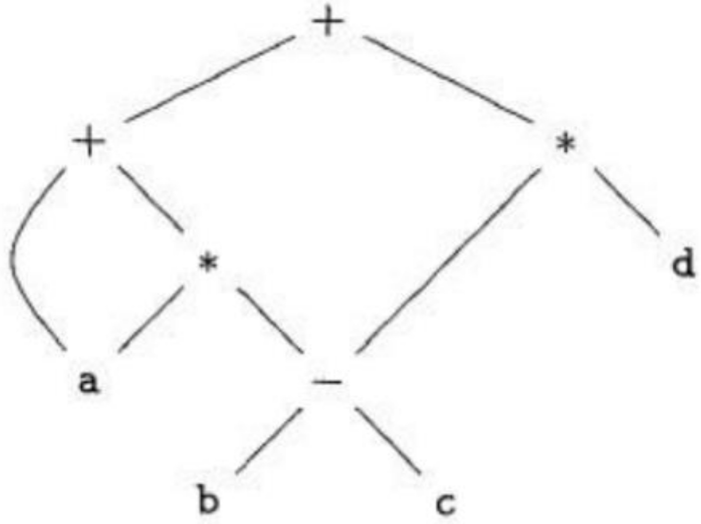
OPTIMIZATION OF BASIC BLOCKS

We can often obtain a substantial improvement in the running time of code merely by performing local optimization within each basic block by itself.

DIRECTED ACYCLIC GRAPHS (DAG)

- Like the syntax tree for an expression, a DAG has leaves corresponding to atomic operands and interior codes corresponding to operators.
- The difference is that a node N in a DAG has more than one parent if N represents a common subexpression; in a syntax tree, the tree for the common subexpression would be replicated as many times as the subexpression appears in the original expression.
- Thus, a DAG not only represents expressions more succinctly, it gives the compiler important clues regarding the generation of efficient code to evaluate the expressions.

Example: The DAG for the expression a+a*(b-c)+(b-c)*d by sequence of steps The leaf for "a" has two parents, because a appears twice in the expression. More interestingly, the two occurrences of the common subexpression b-c are represented by one node, the node labeled "-". That node has two parents, representing its two uses in the subexpressions a*(b-c) and (b-c)*d. Even though b and c appear twice in the complete expression, their nodes each have one parent, since both uses are in the common subexpression b-c.



The DAG Representation of Basic Blocks

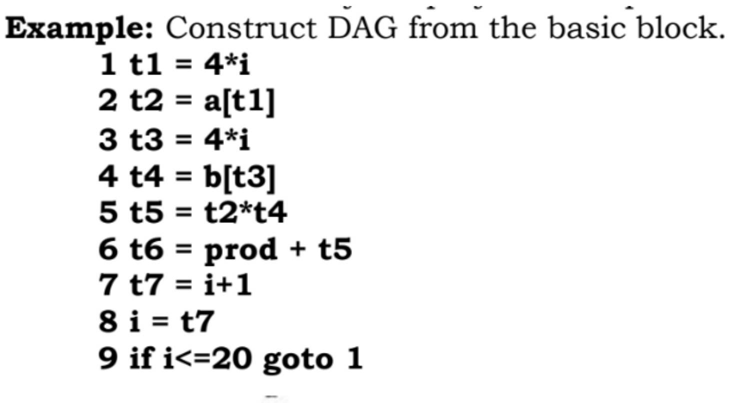
DAG for a basic block as follows:

1. There is a node in the DAG for each of the initial values of the variables appearing in the basic block.

2. There is a node N associated with each statement s within the block. The children of N are those nodes corresponding to statements that are the last definitions, prior to s, of the operands used by s.

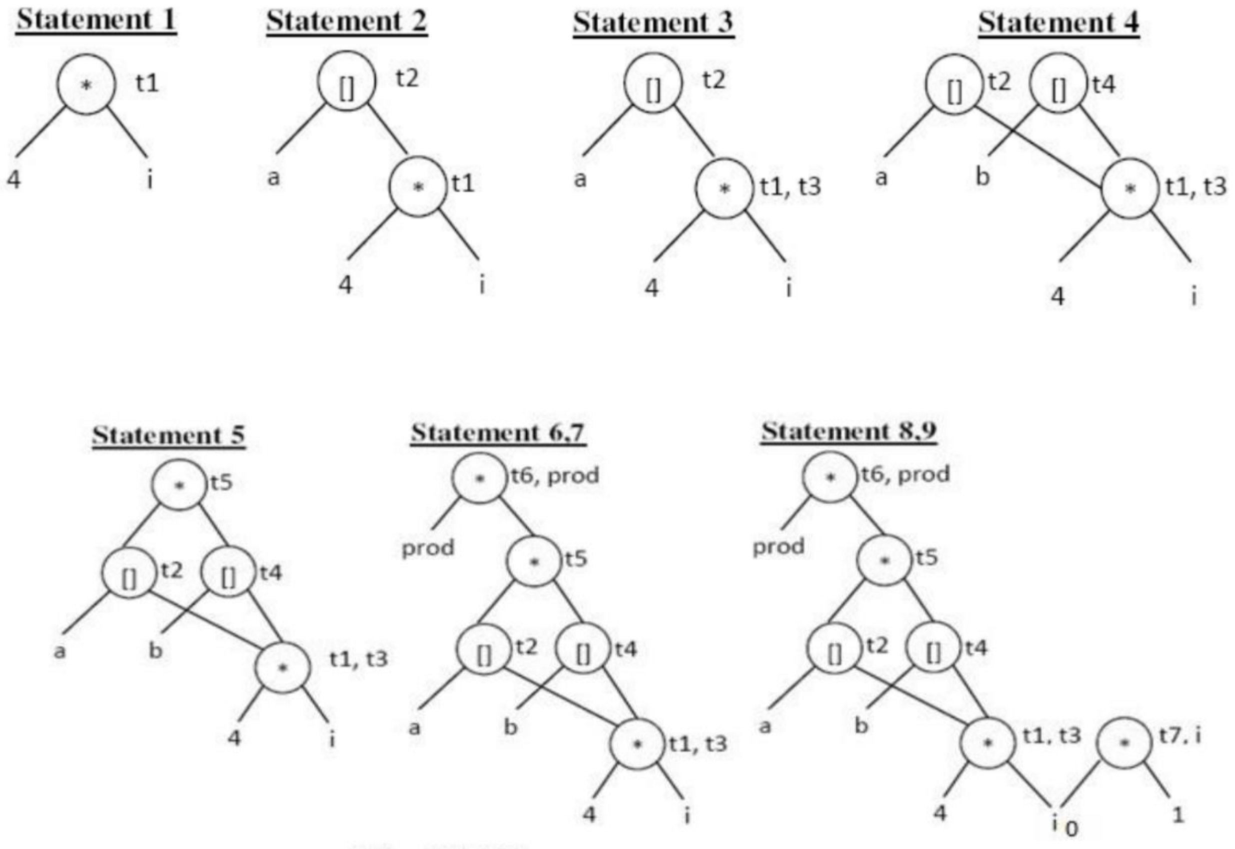
3. Node N is labeled by the operator applied at s, and also attached to N is the list of variables for which it is the last definition within the block.

4. Certain nodes are designated output nodes. These are the nodes whose variables are live on exit from the block; that is, their values may be used later, in another block of the flow graph. Calculation of these "live variables" is a matter for global flow analysis, The DAG representation of a basic block lets us perform several code improving transformations on the code represented by the block.

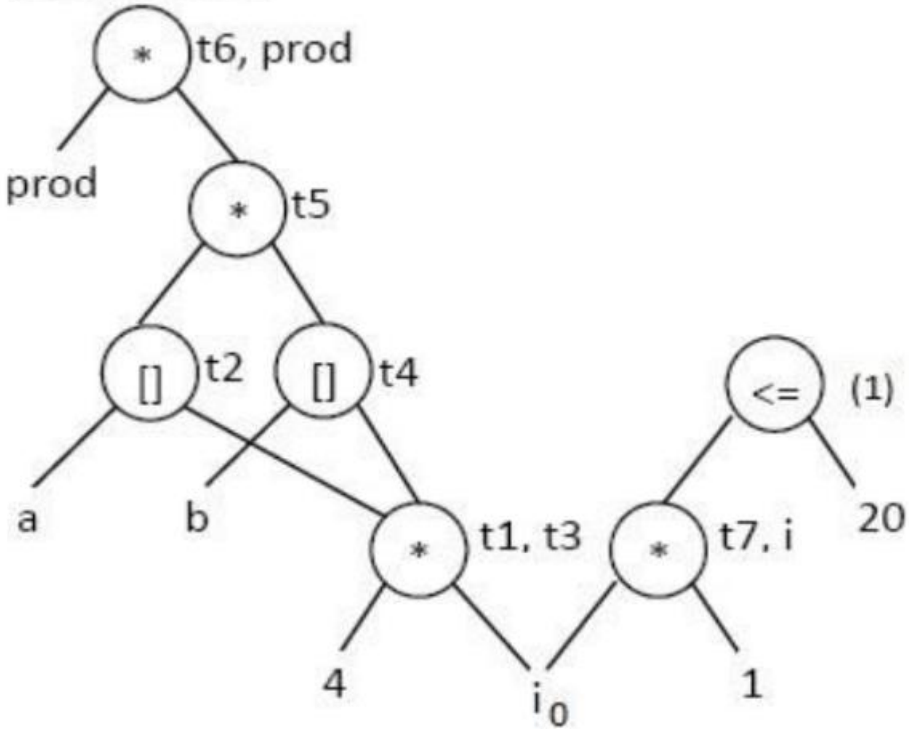


Example: Construct DAG from the basic block.

```
1 t1 = 4*i
2 t2 = a[t1]
3 t3 = 4*i
4 t4 = b[t3]
5 t5 = t2*t4
6 t6 = prod + t5
7 t7 = i+1
8 i = t7
9 if i<=20 goto 1
```



Final DAG


*
t6, prod
prod
*
t5
[]
t2
[]
t4
<=
(1)
a
b
20
*
t1, t3
*
t7, i
4
i0
1