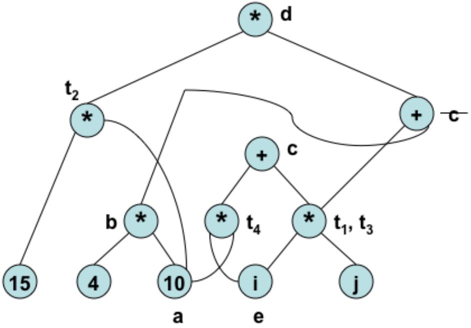
Example of a Directed Acyclic Graph (DAG)

1. $a = 10$
2. $b = 4 * a$
3. $t1 = i * j$
4. $c = t1 + b$
5. $t2 = 15 * a$
6. $d = t2 * c$
7. $e = i$
8. $t3 = e * j$
9. $t4 = i * a$
10. $c = t3 + t4$



Value Numbering in Basic Blocks

- A simple way to represent DAGs is via *value-numbering*
- While searching DAGs represented using pointers etc., is inefficient, *value-numbering* uses hash tables and hence is very efficient
- Central idea is to assign numbers (called value numbers) to expressions in such a way that two expressions receive the same number if the compiler can prove that they are equal for all possible program inputs
- We assume quadruples with binary or unary operators
- The algorithm uses three tables indexed by appropriate hash values:
  HashTable, ValnumTable, and *NameTable*
- Can be used to eliminate common sub-expressions, do constant folding, and constant propagation in basic blocks
- Can take advantage of commutativity of operators, addition of zero, and multiplication by one

Data Structures for Value Numbering

In the field *Namelist*, first name is the defining occurrence and replaces all other names with the same value number with itself (or its constant value)

HashTable entry
(indexed by expression hash value)

Expression	Value number

ValnumTable entry
(indexed by name hash value)

Name	Value number

NameTable entry
(indexed by value number)

Name list	Constant value	Constflag

Example of Value Numbering

HLL Program	Quadruples before Value-Numbering	Quadruples after Value-Numbering
$a = 10$	1. $a = 10$	1. $a = 10$
$b = 4 * a$	2. $b = 4 * a$	2. $b = 40$
$c = i * j + b$	3. $t1 = i * j$	3. $t1 = i * j$
$d = 15 * a * c$	4. $c = t1 + b$	4. $c = t1 + 40$
$e = i$	5. $t2 = 15 * a$	5. $t2 = 150$
$c = e * j + i * a$	6. $d = t2 * c$	6. $d = 150 * c$
	7. $e = i$	7. $e = i$
	8. $t3 = e * j$	8. $t3 = i * j$
	9. $t4 = i * a$	9. $t4 = i * 10$
	10. $c = t3 + t4$	10. $c = t1 + t4$
		(Instructions 5 and 8 can be deleted)

Running the algorithm through the example (1)

1. $a = 10$:
   - a is entered into *ValnumTable* (with a *vn* of 1, say) and into *NameTable* (with a constant value of 10)
2. $b = 4 * a$:
   - a is found in *ValnumTable*, its constant value is 10 in *NameTable*
     - We have performed *constant propagation*
     - $4 * a$ is evaluated to 40, and the quad is rewritten
     - We have now performed *constant folding*
     - b is entered into *ValnumTable* (with a *vn* of 2) and into *NameTable* (with a constant value of 40)
3. $t1 = i * j$:
   - i and j are entered into the two tables with new *vn* (as above), but with no constant value
   - $i * j$ is entered into *HashTable* with a new *vn*
   - $t1$ is entered into *ValnumTable* with the same *vn* as $i * j$

Running the algorithm through the example (2)

4. Similar actions continue till $e = i$
   - e gets the same vn as i
5. $t3 = e * j$:
   - e and i have the same vn
   - hence, $e * j$ is detected to be the same as $i * j$
   - since $i * j$ is already in the HashTable, we have found a *common subexpression*
   - from now on, all uses of $t3$ can be replaced by $t1$
   - quad $t3 = e * j$ can be deleted
6. $c = t3 + t4$:
   - $t3$ and $t4$ already exist and have vn
   - $t3 + t4$ is entered into *HashTable* with a new vn
   - this is a reassignment to c
   - c gets a different vn, same as that of $t3 + t4$
7. Quads are renumbered after deletions

Example: *HashTable* and *ValNumTable*

HashTable

Expression	Value-Number
$i * j$	5
$t1 + 40$	6
$150 * c$	8
$i * 10$	9
$t1 + t4$	11

ValNumTable

Name	Value-Number
a	1
b	2
i	3
j	4
$t1$	5
c	6,11
$t2$	7
d	8
e	3
$t3$	5
$t4$	10