2.DAG

- **What are the advantages of DAG Representation?Give example. [Apr/May 2015]**
- **Describe the algorithm for constructing DAG with an example. [Apr/May 2015]**
- **Generate DAG Representation with an example and list out the applications of DAG Representation. [Nov/Dec 2014]**
- **Construct DAG and three address code for the following C Code [Nov/Dec 2013]**

```
prod=0
i=1
while(i<=20)
{ prod=prod+a[i]*b[i]
i=i+1
}
```

THE DAG REPRESENTATION FOR BASIC BLOCKS

- A DAG for a basic block is a **directed acyclic graph** with the following labels on nodes:
  - Leaves are labeled by unique identifiers, either variable names or constants.
  - Interior nodes are labeled by an operator symbol.
  - Nodes are also optionally given a sequence of identifiers for labels to store the computed values.
- DAGs are useful data structures for implementing transformations on basic blocks.
- It gives a picture of how the value computed by a statement is used in subsequent statements.
- It provides a good way of determining common sub - expressions

Application of DAGs:

1. We can automatically detect common sub expressions.
2. We can determine which identifiers have their values used in the block.
3. We can determine which statements compute values that could be used outside the block.

Algorithm for construction of DAG

Input: A basic block

Output: A DAG for the basic block containing the following information:

1. A label for each node. For leaves, the label is an identifier. For interior nodes, an operator symbol.
2. For each node a list of attached identifiers to hold the computed values.

Case (i)x := y OP z

Case (ii)x := OP y

Case (iii)x := y

Method:

Step 1: If y is undefined then create node(y).
If z is undefined, create node(z) for case(i).

Step 2: For the case(i), create a node(OP) whose left child is node(y) and right child is
node(z) . (Checkingfor common sub expression). Let n be this node.
For case(ii), determine whether there is node(OP) with one child node(y). If not create such a
For case(iii), node n will be node(y).

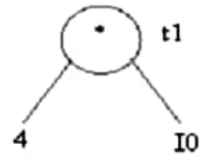
Step 3: Delete x from the list of identifiers for node(x). Append x to the list of attached identifiers for the noden found in step 2 and set node(x) to n.

Example: Consider the block of three- address statements

1. $t_1 := 4* i$
2. $t_2 := a[t_1]$
3. $t_3 := 4* i$
4. $t_4 := b[t_3]$
5. $t_5 := t_2*t_4$
6. $t_6 := prod+t_5$
7. $prod := t_6$
8. $t_7 := i+1$
9. $i := t_7$
10. if i<=20 goto (1)

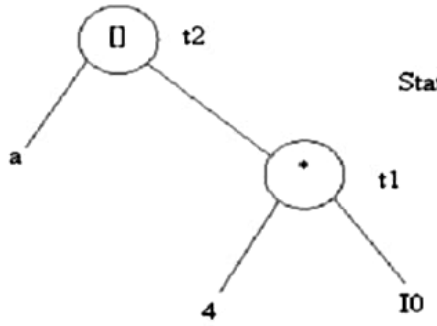
Stages in DAG Construction

(a)



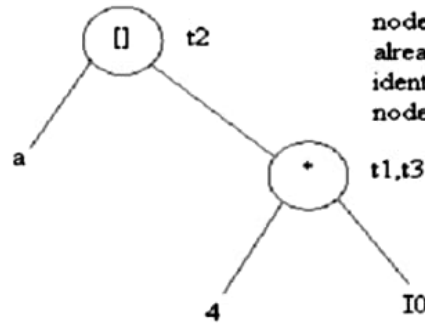
Statement (1)

(b)



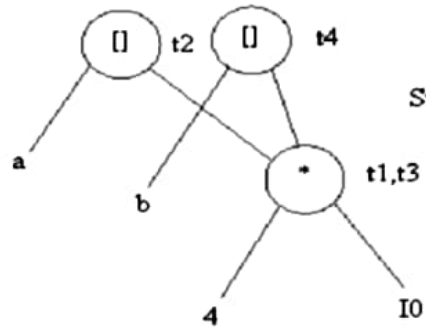
Statement (2)

(c)



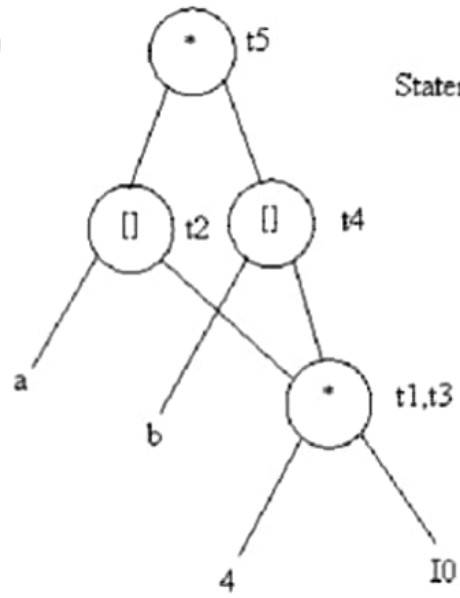
node for 4*I0 exist already, hence attach identifier t3 to the existing node for Statement (3)

(d)



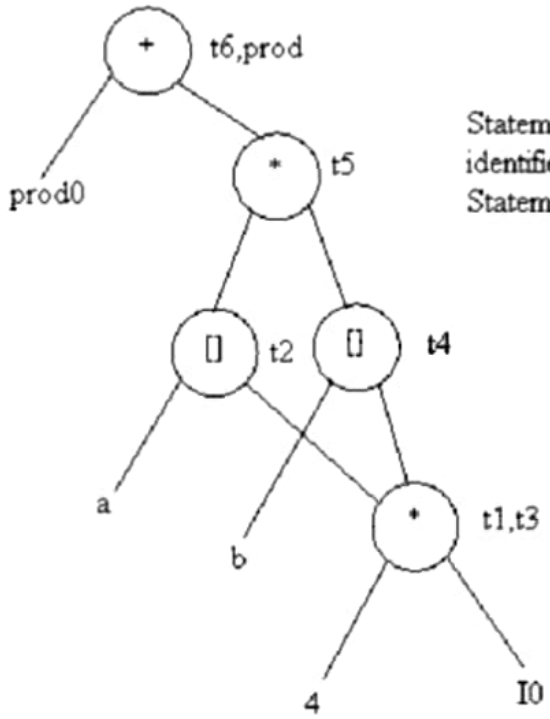
Statement (4)

(e)



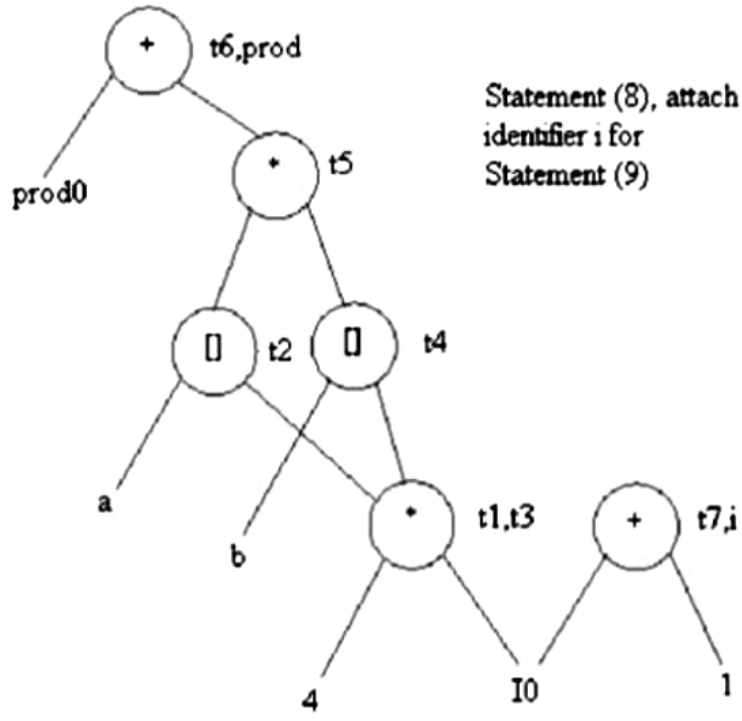
Statement (5)

(f)



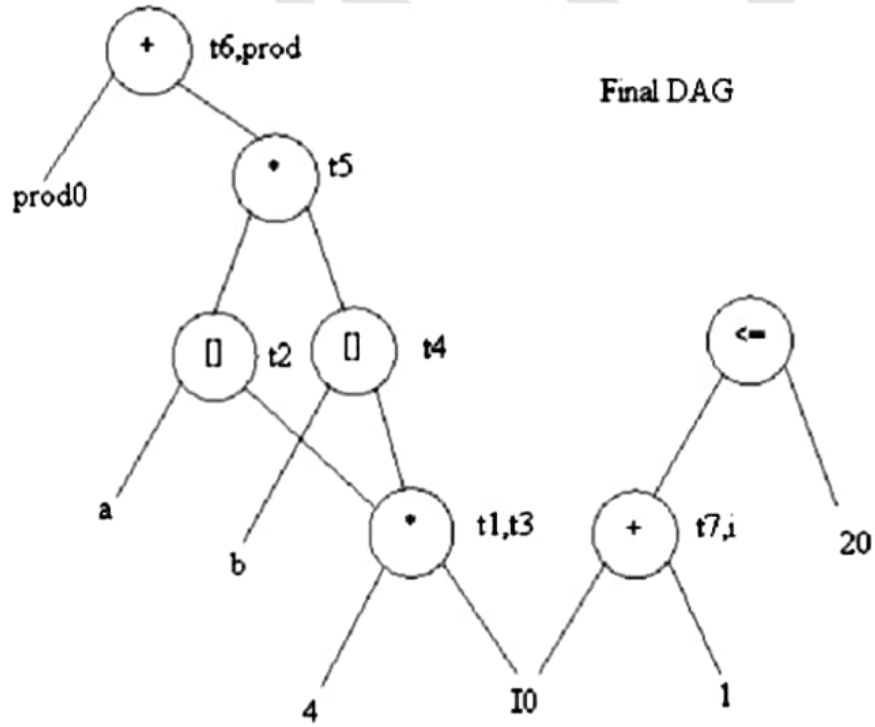
Statement (6), attach identifier prod for Statement (7)

(g)



Statement (8), attach identifier i for Statement (9)

(h)



Final DAG