Directed

Acyclic

Graphs

 amatous

LearnVidFun...
Definitions

In complex design, a directed acyclic graph (DAG) is an abstract syntax tree (AST) with a unique node for each value.

A directed acyclic graph (DAG) is a directed graph that contains no cycles.
Use of DAG for optimizing basic blocks:

- DAG is a useful data structure for implementing transformations on basic blocks.
- A basic block can be optimized by the construction of DAG.
- A DAG can be constructed for a block and certain transformations such as common subexpression elimination and dead code elimination can be applied for performing the local optimization.
- To apply the transformations on basic block, a DAG is constructed from three address statement.
Properties of a DAG

1. The reachability relation in a DAG forms a partial order and any finite partial order may be represented by a DAG using reachability.

2. The transitive reduction and transitive closure are both uniquely defined for DAGs.

3. Every DAG has a topological ordering.
Applications of a DAG

The DAG is used in-

1. determining the common subexpression (expressions computed more than once).
2. determining which names are used in the block and computed outside the block.
3. determining which statements of the block could have their computed value outside the block.
4. simplifying the list of quadruples by eliminating the common subexpressions and not performing the assignment of the form $x := y$ until and unless it is a must.
Rules for the construction of DAG:

Rule-1: In a DAG,
  → Leaf nodes represent identifiers, names or constants.
  → Interior nodes represent operators.

Rule-2: While constructing DAG, there is a check made to find if there is an
  existing node with the same children. A new node is created only when
  such a node does not exist. This action allows us to detect common
  sub-expressions and eliminate the re-computation of the same.

Rule-3: The assignment of the form \( x := y \) must not be performed until
  and unless it is a must.

**********
**Problem 1:** Construct DAG for the given expression:

\[(a+b) \times (a+b+c)\]

**Solu:** Three address code for the given expression is:

\[
\begin{align*}
t1 &= a + b \\
t2 &= t1 + c \\
t3 &= t1 \times t2
\end{align*}
\]

The DAG is:

```
\[\text{+} \quad \text{+} \quad \text{*} \quad \text{+} \quad \text{=} \]
```

```
\[a \quad b \quad t1 \quad t2 \quad t3 \quad c \]
```

```
\[\text{+} \quad \text{=} \quad \text{=} \]
```

```
\[a \quad b \quad c \]
```
Explanation:

From the constructed OAG, we observe that the common subexpression \((a + b)\) is translated into a single node in the OAG. The computation is carried out only once and stored in the identifier \(t_1\) and reused later.

This illustrates how the OAG construction scheme identifies the common sub-expression and helps in eliminating its re-computation later.
Problem-02: Construct DAG for the given expression:

\[ (((a+a) + (a+a)) + ((a+a) + (a+a))) \]

Solt:-

DAG for the given expression is:

[Diagram of DAG]

www.gatevidyalay.com
Problem-03: Construct the DAG for the following block:

\[
\begin{align*}
a &= b \times c \\
d &= b \\
e &= d \times c \\
b &= e \\
f &= b + c \\
g &= f + d
\end{align*}
\]

Soln:

**Step-1:**

\[\begin{array}{c}
a \\
b \\
c
\end{array}\quad \begin{array}{c}
\times \\
\downarrow \\
\downarrow \\
b \\
c
\end{array}\]
Step-2:

Step-3:

Step-4:
Step-5:

\[ a, e, b \]
\[ d \]
\[ c \]

Step-6:

\[ g \]
\[ d \]
\[ c \]

---

**********
Problem-4: Optimize the block given in Problem-3.

Soln:

Step-1: First construct the DAG for the given block.

Step-2: Now, the optimized code can be generated by traversing the DAG.

1. The common subexpression $e = d \times c$ which is actually $b \times c$ (i.e., $d = b$) is eliminated.

2. The dead code $b = e$ is eliminated.

The optimized basic block is:

\[
\begin{align*}
  a &= b \times c \\
  d &= b \\
  f &= a + c \\
  g &= f + d
\end{align*}
\]
Problem-5: Consider the following basic block. Draw the DAG representation of the block and identify local common sub-expressions. Eliminate the common expressions and rewrite the basic block.

L10:

\[ S_1 = 4 \times I \]
\[ S_2 = \text{addn}(A) - 4 \]
\[ S_3 = S_2[I] \]
\[ S_4 = 4 \times I \]
\[ S_5 = \text{addn}(B) - 4 \]
\[ S_6 = S_5[S_4] \]
\[ S_7 = S_3 \times S_6 \]
\[ S_8 = \text{prod} + S_7 \]
\[ \text{prod} = S_8 \]
\[ S_9 = I + 1 \]
\[ I = S_9 \]
\[ \text{If } I \leq 20 \text{ goto L10} \]
Solution:

DAG representation for the block is:

```
s8, prod
+   
|   |
|   |
s7

PROD

*[ ]

s6

s5

adder (B)

- [ ]

adder (A)

s2

s1, s4

[ ]

<

s9, I

20

+ I

1
```
In this code fragment,

- $4 \times I$ is a common subexpression. Hence, we can eliminate $S_4$ because $S_1 = S_4$.

- $S_8 = \text{PROD} + S_7$
  
  $\text{PROD} = S_8$
  
  can be optimized as
  
  $\text{PROD} = \text{PROD} + S_7$

- $S_9 = I + 1$
  
  $I = S_9$
  
  can be optimized as
  
  $I = I + 1$

After eliminating $S_4$, $S_8$, and $S_9$, we get -
L10:

\[ s1 = 4 * I \]
\[ s2 = \text{addr}(A) - 4 \]
\[ s3 = s2[s1] \]
\[ s5 = \text{addr}(B) - 4 \]
\[ s6 = s5[s1] \]
\[ s7 = s3 * s6 \]
\[ \text{PROD} = \text{PROD} + s7 \]

\[ I = I + 1 \]

IF \( I \leq 20 \) GOTO L10