4/2/2021

UNIT-V

Code Optimization:-

* It removes the unwanted or repeated instructions from the intermediate code so that object program runs very fastly.
* The code optimization phase consists of control flow analysis and data flow analysis followed by the application of transformations.
* In code optimizer, programs are represented by flow graphs.

Flow graph:-

* Flow graph is a graph representation of three address code statements in which nodes represents basic blocks and edges represents flow of control.

Basic block:-

* Basic block is a set of statements in which flow of control enters at the beginning and leaves at the end.

Algorithm for constructing Basic blocks:-

1. We first determine the leaders (L) using the following rules.

a) First statement is a leader.

b) The target of conditional or unconditional jump statements is a leader.

c) The statement immediately following the conditional or unconditional jump statement is a leader.

2. For each leader its basic block consists of statements upto but not including the next leader or end of the program.

Eg, construct flow graph for the following statements:

```
begin
    p = 0;
    i = 1;
    do
    {
        p = p + a[i] * b[i];
        i = i + 1;
    } while (i <= 20);
end
```

→ array takes 4 bytes of memory

DD(3-10)

Three address codes

L: 1. $p = 0$

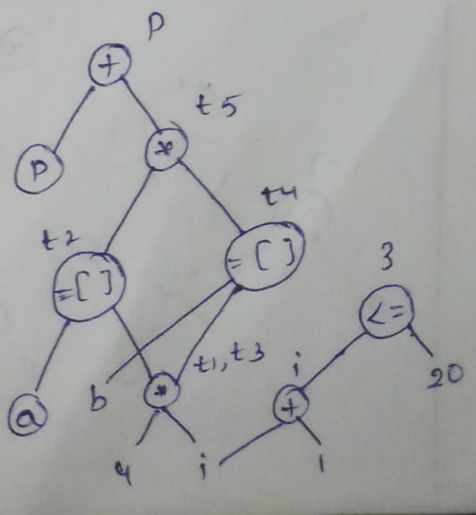
2. $i = 1$

L: 3. $t_1 = 4 * i$

4. $t_2 = a[t_1]$

5. $t_3 = 4 * i$

6. $t_4 = b[t_3]$



7. $t_5 = t_1 * t_4$

8. $P = P + t_5$

9. $i = i + 1$

10. if $i <= 20$ goto 3

L: 11. —

Basic blocks:-

B1 $\boxed{1-2}$

B2 $\boxed{3-10}$

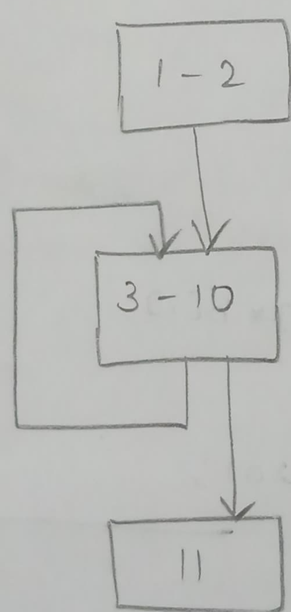
B3 $\boxed{11}$

Flow Graphs:-

B1

B2

B3



Eg: **Quick Sort**

```
i = m-1;
j = n;
v = a[n];
while(1)
{
    do
    {
        i = i+1;
    } while (a[i]<v);
    do
    {
        j = j-1;
    } while (a[j]>v);
    if (i>=j)
        break;
    x = a[i];
    a[i] = a[j];
    a[j] = x;
}
x = a[i];
a[i] = a[n];
a[n] = x;
```

Three address codes :-

L 1. $i = m-1$

2. $j = n$

3. $t_1 = 4*n$

4. $v = a[t_1]$

L 5. $i = i+1$

6. $t_2 = 4*i$

7. $t_3 = a[t_2]$

8. if $t_3 < v$ goto 5

L 9. $j = j - 1$

10. $t_4 = 4 * j$

11. $t_5 = a[t_4]$

12. if $t_5 > v$ goto 9

L 13. if $i >= j$ goto 23

L 14. $t_6 = 4 * i$

15. $x = a[t_6]$

16. $t_7 = 4 * i$

17. $t_8 = 4 * j$

18. $t_9 = a[t_8]$

19. $a[t_7] = t_9$

20. $t_{10} = 4 * j$

21. $a[t_{10}] = x$

22. goto 5

L 23. $t_{11} = 4 * i$

24 $x = a[t_{11}]$

25. $t_{12} = 4 * i$

26. $t_{13} = 4 * n$

27. $t_{14} = a[t_{13}]$

28. $a[t_{12}] = t_{14}$

29. $t_{15} = 4 * n$

30. $a[t_{15}] = x$

Basic blocks:

B_1 1 - 4

B_2 5 - 8

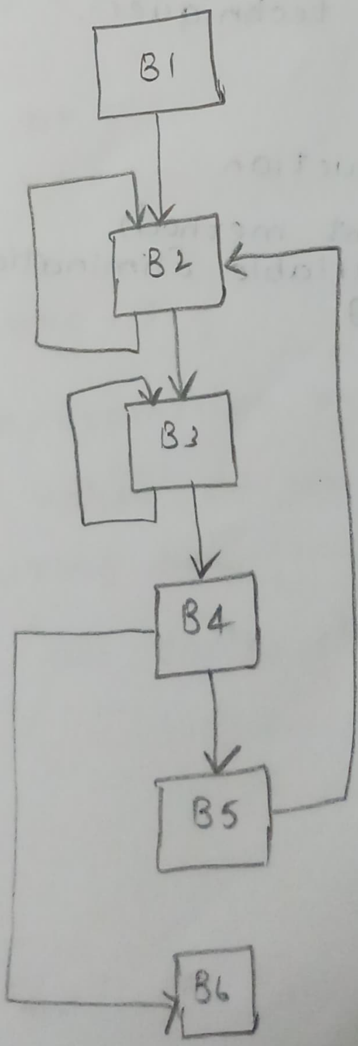
B_3 9 - 12

B_4 13

B_5 14 - 22

B_6 23 - 30

Flow Graph:



Principal Sources of optimization/Machine independent optimization / Transformations :

- Local optimization : within a basic block.
- Global optimization : accross all basic blocks.
- Loop optimization ; within a loop

~~Loop~~

⇒ Local Optimization Techniques (or Function-Preserving Transformations) (or) Semantic preserving transformations

1. Common sub expression Elimination
2. Copy Propagation
3. Dead-~~Co~~ code elimination
4. constant folding.

Loop optimization techniques :.

1. Code motion
2. Strength reduction
3. (Loop invariant method)
3. Induction variable Elimination
4. Loop unrolling
5. Loop fusion,

Local Optimization:-

1. Common Sub expression Elimination:-

- An occurrence of an expression 'E' is called a common sub expression if E was previously computed and the values of variables in E have not changed since the previous computation.

Eg) 1. $a = b * c$

⋮

$z = b * c + d - e$

After elimination,

$t = b * c$

$a = t$

⋮

$z = t + d - e$

2. $a = b * c$

$c = 4 + c$

⋮

$z = b * c + d - e$

Here, the expression $b*c$ is not common because the value of variable c is changed after computing $b*c$.

Therefore, we can't eliminate this expression.

2. Copy Propagation:-

* The statement of the form $f = g$ is a copy statement.
* The idea behind copy propagation is to use 'g' for 'f'.

Eg: 1)

$x = t_3$

$a[t_2] = t_5$

$a[t_4] = x$

goto 5

After applying copy propagation,

$x = t_3$

$a[t_2] = t_5$

$a[t_4] = t_3$

goto 5

2)

count = t5

count = count + 1

C.P

count = t5

$t_5 = t_5 + 1$

3. Dead code Elimination:-

* A variable is said to be live if its value can be used subsequently, otherwise it is dead at that point.

Eg:

a = 10 (dead statement)

a = 20

printf("%d", a) # 20

4. Constant Folding:-

* It is a process of replacing constant expressions by their values at compile time.

Eg: The expression 2 * 3.14 replaced by 6.28 at compile time.

Loop Optimization Techniques:-

* The running time of a program may be improved if we decrease the number of instructions in the inner loop, even if we increase the amount of code outside the loop.

1. Code Motion (Frequency Reduction):-

* It takes the constant expression from the loop and places the expression before the loop.

loop invariant computation

Eg: 1)

```
while (i < limit - 2)          t = limit - 2
{                         =>   while (i < t)
  ---                          {
  ---                            --
}                                --
                               }
```

2)

```
for (i=0; i<10; i++)      =>   t = x*x
  a[i] = 4*i + x*x             for (i=0; i<10; i++)
                                 a[i] = 4*i+t
```

Strength Reduction:-

* Replacing a ~~complex~~ expensive operator by a ~~simple~~ cheaper operator.

Eg:

^ (**) by * => $x**2 = x*x$

* by + => $x*x = x+x$

/ by * => $x/2 = x*0.5$

Induction Variable Elimination:

A variable 'x' is called Induction variable of loop L, if the value of the variable gets changed everytime. It is either decremented or incremented by some constant.

Eg:

$j = j - 1$

$t_4 = 4 * j$

$t_5 = a[t_4]$

if $t_5 > v$ goto 9

$\xRightarrow{A.I.V.E}$

$j = j - 1$

$t_4 = t_4 - 4$

$t_5 = a[t_4]$

if $t_5 > v$ goto 9

Assign $t_4 = 4 * j$ in Basic block 1

$t_4 = 4 * (j - 1)$

$= 4 * j - 4$

$= t_4 - 4$

Loop Unrolling :- / unwinding

In this method, the number of jumps and tests can be reduced by writing the code two times.

Eg:

```
int i = 1
while (i <= 100)
{
    a[i] = b[i]
    i++
}
```

$\Rightarrow$

```
int i = 1
while (i <= 100)
{
    a[i] = b[i]
    i++
    a[i] = b[i]
    i++
}
```

Loop Jaming / Loop Fusion:-

- In this method, several loops are merged to one loop.

Eg: 1)

```
for i = 1 to n do                    for i = 1 to n*m
  for j = 1 to m do         =>       do
    a[i, j] := 10                      a[i, j] = 10
```

2) Before

```
int i, a[100], b[100];
for (i=0; i<100; i++)
a[i] = 1;
for (i=0; i<100; i++)
b[i] = 2;
```

After

```
int i, a[100], b[100];
for (i=0; i<100; i++)
{
  a[i] = 1;
  b[i] = 2;
}
```

Directed Acyclic Graph:-

- DAG is a directed graph with no cycles.
- DAG is a data structure used to implement transformations on basic blocks.
- We can optimize a basic block by constructing a DAG for it.
- In DAG:

  Leaf nodes represent identifiers i.e., names or constants.

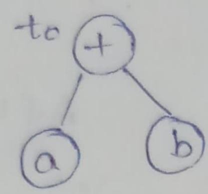
  Interior nodes represent operators.

- DAG is used by identifying the common sub expressions.
- In DAG representation common sub expression has more than one parent.

Construction of DAG for a Three-Address statements:

If the statement is of the form

$t_0 = a + b$

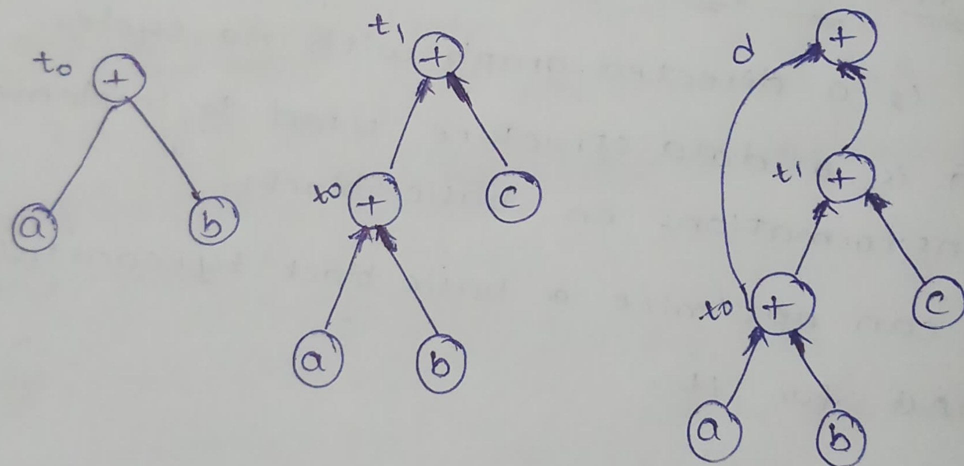


construct DAG for the following expressions

$t_0 = a + b$

$t_1 = t_0 + c$

$d = t_0 + t_1$

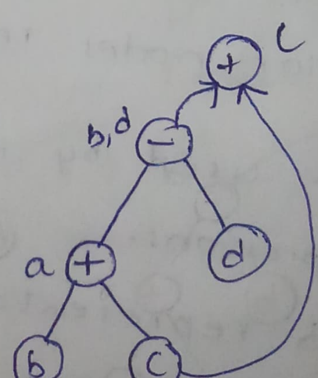


2) $a = b + c$

$b = a - d$

$c = b + c$

$d = a - d$

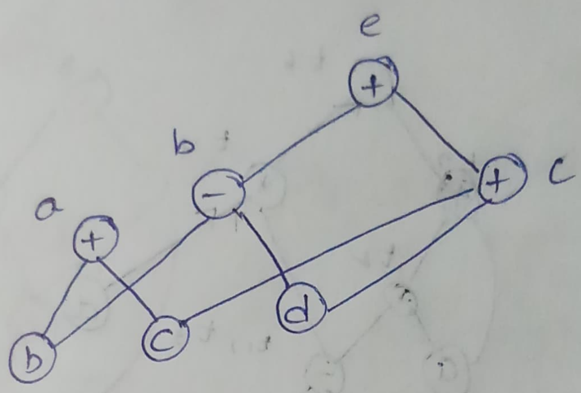


3) $a = b + c$

$b = b - d$

$c = c + d$

$e = b + c$



4) $a := b * - c + b * - c$

$t_1 = -c$

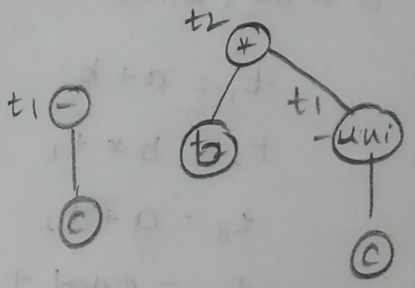
$t_2 = b * t_1$

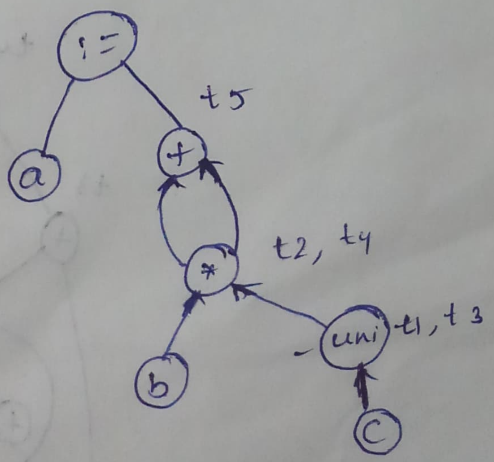
$t_3 = -c$

$t_4 = b * t_3$

$t_5 = t_2 + t_4$

$a = t_5$





5) $a \cancel{+} a*(b-c) + (b-c)*d$

$t_1 = b - c$

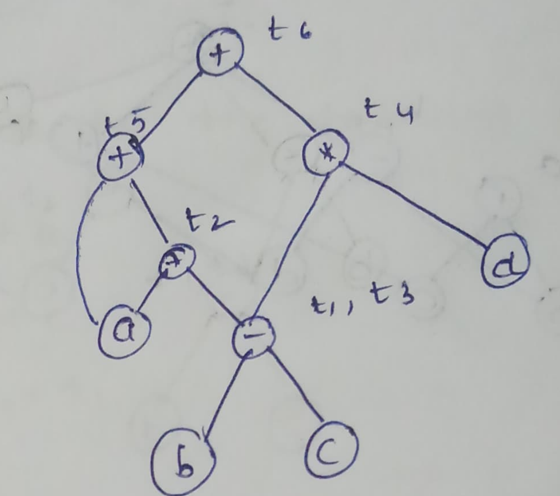
$t_2 = a * t_1$

$t_3 = b - c$

$t_4 = t_3 * d$

$t_5 = \cancel{t_2 + t_4}\ a + t_2$

$\cancel{t_6 = a + t_5}\ t_6 = t_5 + t_4$



6) $a + b*(a+b) + c + d$

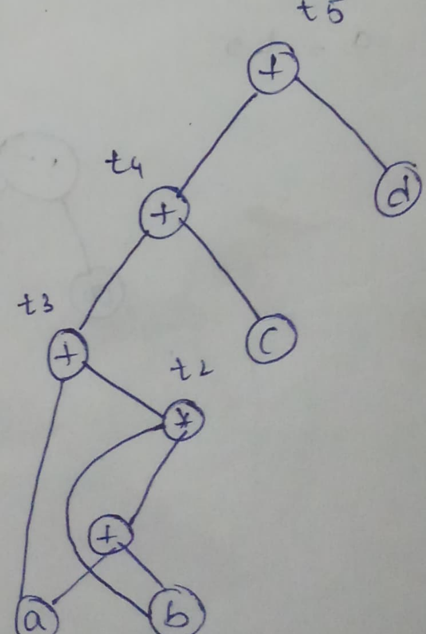
$t_1 = a + b$

$t_2 = b * t_1$

$t_3 = a + t_2$

$t_4 = \cancel{a+d}\ t_3 + c$

$t_5 = t_4 + d$

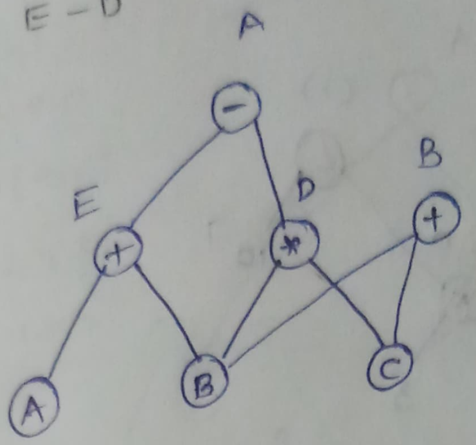


7) $D = B * C$

$E = A + B$

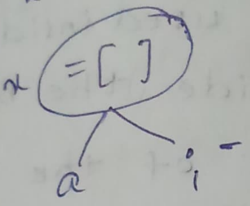
$B = B + C$

$A = E - D$

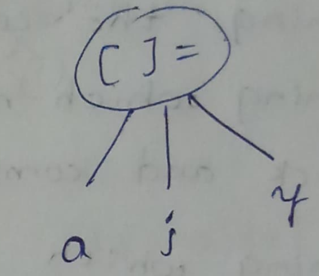


<u>Note:-</u>

$x = a[i]$



$a[i] = y$



construct DAG for the following three address statements.

8) $t_1 = 5 + a$

$a[i]$

$t_2 = x[t_1]$

$t_3 = 5 + a$

$t_4 = y[t_3]$

$a[i]$

$t_5 = t_2 + t_4$

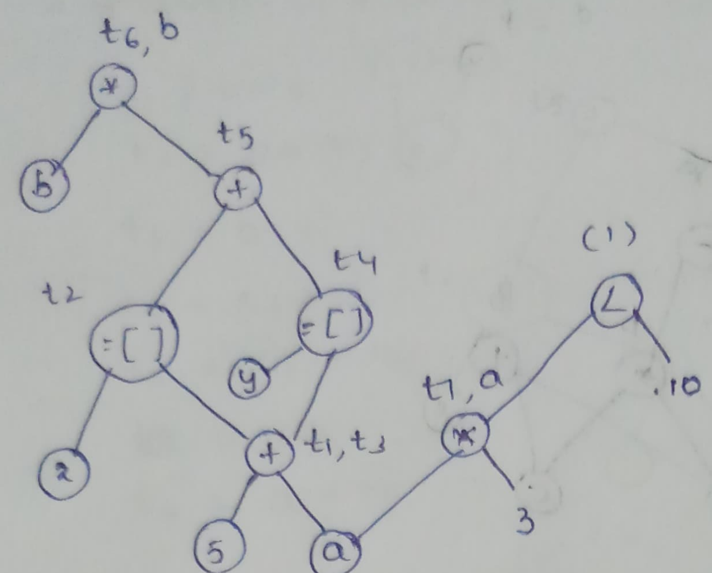
$t_6 = b * t_5$

$b = t_6$

$t_7 = a * 3$

$a = t_7$

if $a < b$ goto 1



Applications of DAG:-

1. Determining the common sub expression.
2. Determining which names are used inside the block and computed outside the block.
3. Determining which statements of the block could have their computed values outside the block.

Optimization of basic blocks:-

Quicksort Example:

Consider blocks:	After eliminating the Block 5 is
$t_6 := 4*i$	$t_6 := 4*i$
$x = a[t_6]$	$x = a[t_6]$
$t_7 = 4*i$	$t_8 = 4*j$
$t_8 = 4*j$	$t_9 = a[t_8]$
$t_9 = a[t_8]$	$a[t_6] = t_9$
$a[t_7] = t_9$	$a[t_8] = x$
$t_{10} = 4*j$	goto 5
$a[t_{10}] = x$	
Goto 5	

Block - 6

$t_{11} := u * i$

$x := a[t_{11}]$

$t_{12} = u * i$

$t_{13} = u * n$

$t_{14} = a[t_{13}]$

$a[t_{12}] = t_{14}$

$t_{15} = u * n$

$t_{15} = x$

After eliminating the Block 6 is

$t_{11} := u * i$

$x := a[t_{11}]$

$t_{13} := u * n$

$t_{14} := a[t_{13}]$

$a[t_{11}] = t_{14}$

$t_{13} := x$

consider B_2 B_5

B_5 :- After eliminating t_6

$x = a[t_2]$

$t_8 = u * j$

$t_9 = a[t_8]$

$a[t_2] = t_9$

$a[t_8] = x$

goto 5

$\Rightarrow$

$x = t_3$

$t_8 = u * j$

$t_9 = a[t_8]$

$a[t_2] = t_9$

$a[t_8] = x$

goto 5

$B_3 B_5$:- After eliminating t_6

$x = t_3$

$t_9 = a[t_4]$

$a[t_2] = t_9$

$a[t_4] = x$

goto 5

$\Rightarrow$

$x = t_3$

$t_9 = t_5$

$a[t_2] = t_9$

$a[t_4] = x$

goto 5

$\Rightarrow$

$x = t_3$

$a[t_2] = t_5$

$a[t_4] = x$

goto 5

B_1, B_6 :-

B_1 :

$i = m-1$

$j = n$

$t_1 = 4*n$

$v = a[t_1]$

B_6 : $t_{11} := 4*i$

$x := a[t_{11}]$

$t_{13} := 4*n$

$t_{14} := a[t_{13}]$

$a[t_{11}] = a[t_{13}]$

$a[t_{13}] := x$

$B_6 \Rightarrow x = t_3$

$a[t_{14}] = a[t_1]$

$a[t_2] = t_{14}$

$a[t_1] = x$

After copy propagation

B_5 is

$x = t_3$

$a[t_2] = t_5$

$a[t_4] = t_3$

goto 5

B_6 is

$a[t_{14}] = a[t_4]$

$a[t_2] = t_{14}$

$a[t_1] = t_3$

Dead code Elimination

B_5

$a[t_2] = t_5$

$a[t_4] = t_3$

goto 5

B_6

$t_{14} = a[t_1]$

$a[t_2] = t_{14}$

$a[t_1] = t_3$