

1.

(a) Compute Gen and Kill sets for each block in the flow graph.

$\text{Gen}(B1) = \{ 1, 2 \}$ $\text{Kill}(B1) = \{ 8, 10, 11 \}$

$\text{Gen}(B2) = \{ 3, 4 \}$ $\text{Kill}(B2) = \{ 5, 6 \}$

$\text{Gen}(B3) = \{ 5 \}$ $\text{Kill}(B3) = \{ 4, 6 \}$

$\text{Gen}(B4) = \{ 6, 7 \}$ $\text{Kill}(B4) = \{ 4, 5, 9 \}$

$\text{Gen}(B5) = \{ 8, 9 \}$ $\text{Kill}(B5) = \{ 2, 7, 11 \}$

$\text{Gen}(B6) = \{ 10, 11 \}$ $\text{Kill}(B6) = \{ 1, 2, 8 \}$

(b) Compute the In and Out sets for each block in the flow graph.

Step 1:

$\text{In}(B1) = \{ \}$ $\text{Out}(B1) = \{ 1, 2 \}$

$\text{In}(B2) = \{ \}$ $\text{Out}(B2) = \{ 3, 4 \}$

$\text{In}(B3) = \{ \}$ $\text{Out}(B3) = \{ 5 \}$

$\text{In}(B4) = \{ \}$ $\text{Out}(B4) = \{ 6, 7 \}$

$\text{In}(B5) = \{ \}$ $\text{Out}(B5) = \{ 8, 9 \}$

$\text{In}(B6) = \{ \}$ $\text{Out}(B6) = \{ 10, 11 \}$

Step 2:

$\text{In}(B1) = \{ \}$ $\text{Out}(B1) = \{ 1, 2 \}$

$\text{In}(B2) = \{ 1, 2, 8, 9 \}$ $\text{Out}(B2) = \{ 1, 2, 3, 4, 8, 9 \}$

$\text{In}(B3) = \{ 1, 2, 3, 4, 6, 7, 8, 9 \}$ $\text{Out}(B3) = \{ 1, 2, 3, 5, 7, 8, 9 \}$

$\text{In}(B4) = \{ 1, 2, 3, 5, 7, 8, 9 \}$ $\text{Out}(B4) = \{ 1, 2, 3, 6, 7, 8 \}$

$\text{In}(B5) = \{ 1, 2, 3, 5, 7, 8, 9 \}$ $\text{Out}(B5) = \{ 1, 3, 5, 8, 9 \}$

$\text{In}(B6) = \{ 1, 3, 5, 8, 9 \}$ $\text{Out}(B6) = \{ 3, 5, 9, 10, 11 \}$

Step 3:

$\text{In}(B1) = \{ \}$ $\text{Out}(B1) = \{ 1, 2 \}$

$\text{In}(B2) = \{ 1, 2, 3, 5, 8, 9 \}$ $\text{Out}(B2) = \{ 1, 2, 3, 4, 8, 9 \}$

$\text{In}(B3) = \{ 1, 2, 3, 4, 6, 7, 8, 9 \}$ $\text{Out}(B3) = \{ 1, 2, 3, 5, 7, 8, 9 \}$

$\text{In}(B4) = \{ 1, 2, 3, 5, 7, 8, 9 \}$ $\text{Out}(B4) = \{ 1, 2, 3, 6, 7, 8 \}$

$\text{In}(B5) = \{ 1, 2, 3, 5, 7, 8, 9 \}$ $\text{Out}(B5) = \{ 1, 3, 5, 8, 9 \}$

$\text{In}(B6) = \{ 1, 3, 5, 8, 9 \}$ $\text{Out}(B6) = \{ 3, 5, 9, 10, 11 \}$

(c) Perform constant propagation and constant folding.

Cannot folder any constant

B1:

In: no def of a and b

Out: a=1, b=2

B2:

In a=1 b=* c=* d=* e=*

Out: a=1 b=* c=* d=* e=*

B3:

In a=1 b=* c=* d=* e=*

Out: a=1 b=* c=* d=* e=*

B4:

In a=1 b=* c=* d=*, e=*

Out: a=1, b=* c=* d=*, e=*

B5:

In =1 b=* c=* d=* d=*

Out a=1 b=*,c=*, d=*,e=*

B6:

In a=1, b=* c=*, d=* e=*

Out a=*, b=* c=*, d=* e=*

(d) Perform common sub expression elimination

GEN(B1)={}

Kill(B1) = { }

Gen(B2) = {c=a+b, d=c-a}

Kill(B2) = {b=a-d, b=a+b, a=b*d, }

Gen(B3) = {d=b+d}

Kill(B3) = { d=a+b, d=c-a, b=a+b, b=a-d }

Gen(B4) = { d=a+b, e=e+1 }

Kill(B4) = { b=a-d, b=a+b, a=b*d, e=c-a }

Gen(B5) = { b=a+b, e=c-a }

Kill(B5) = { b=a-d, b=a+b, a=b*d, , c=a+b }

Gen(B6) = { a=b*d, b=a-d }

Kill(B6) = { b=a+b, d=a+b, d=c-a ,d=b+d }

In(B1) = { }

Out(B1) = { }

In(B2) = { b=a+b, e=c-a, d= c-a }

Out(B2) = { c=a+b, d=c-a, b=a+b }

T1= a+b, c=T1

T2= c-a, d =T2

In(B3) = { c=a+b, d=c-a,d=a+b, e=e+1,b=a+b}

Out(B3) = { d=b+d, d=c-a, c=a+b, d=a+b, b=a+b, e=e+1 }

In(B4) = { d=b+d, d=c-a, c=a+b , d=a+b, e=e+1,b=a+b}

Out(B4) = {d=c-a, c=a+b, d=a+b,e=e+1 ,b=a+b}

d=T1

In(B5)={ d=b+d, d=c-a, c=a+b ,b=a+b}

Out(B5) = {b=a+b, e=c-a, d= c-a }

e=T2

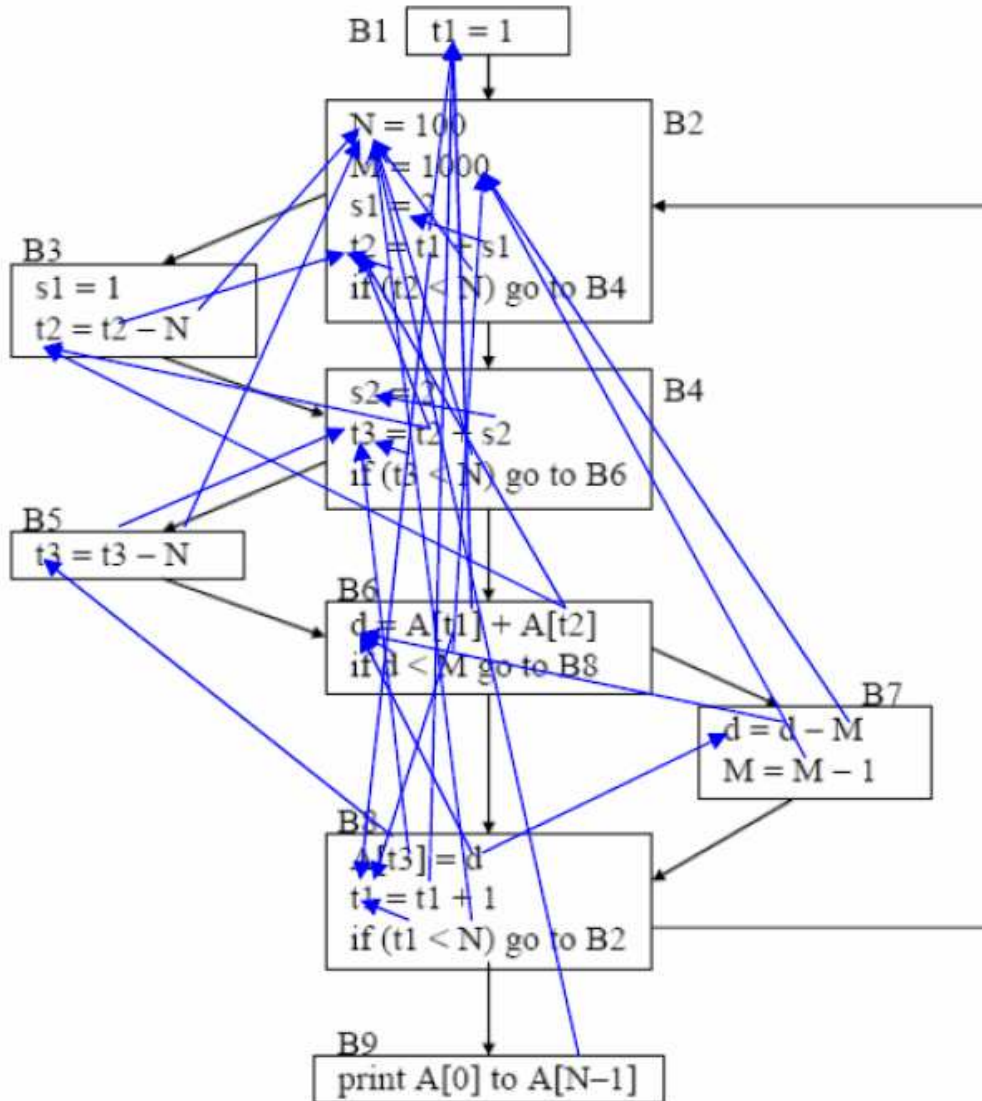
b=T1

In(B6) = { d=c-a, b=a+b, e=c-a }

Out(B6) = { a =b*d, b=a-d }

2

(a) Identify and mark define-use links within the loop based on data flow analysis results (based on the In and Out sets). You need to consider the block level define-use relations as well as define-use relations within the block.



(b) Identify all the loop invariants based on the define-use links computed in (a). Assume that all the loop invariants can be moved out of the loop. So you need to find loop invariants repetitively till a fixed point is reached. In each round, you need to pretend to move out those loop invariants you found in the previous rounds.

The loop invariants are listed as follows,

B2:

$N = 100$

$M = 1000$

$S1 = 2$

B3:

$S1 = 1$

B4:

$S2 = 2$

(c) For each loop invariant, determine whether it can actually be moved out of the loop. If so, move it out of the loop. If not, state the reason why it cannot be moved out. Generate the new code after code motion.

The following loop invariants can be moved out of the loop,

B2:

$N = 100$

B4:

$S2 = 2$

The following loop invariants cannot be moved out of the loop,

B2:

$M = 1000$

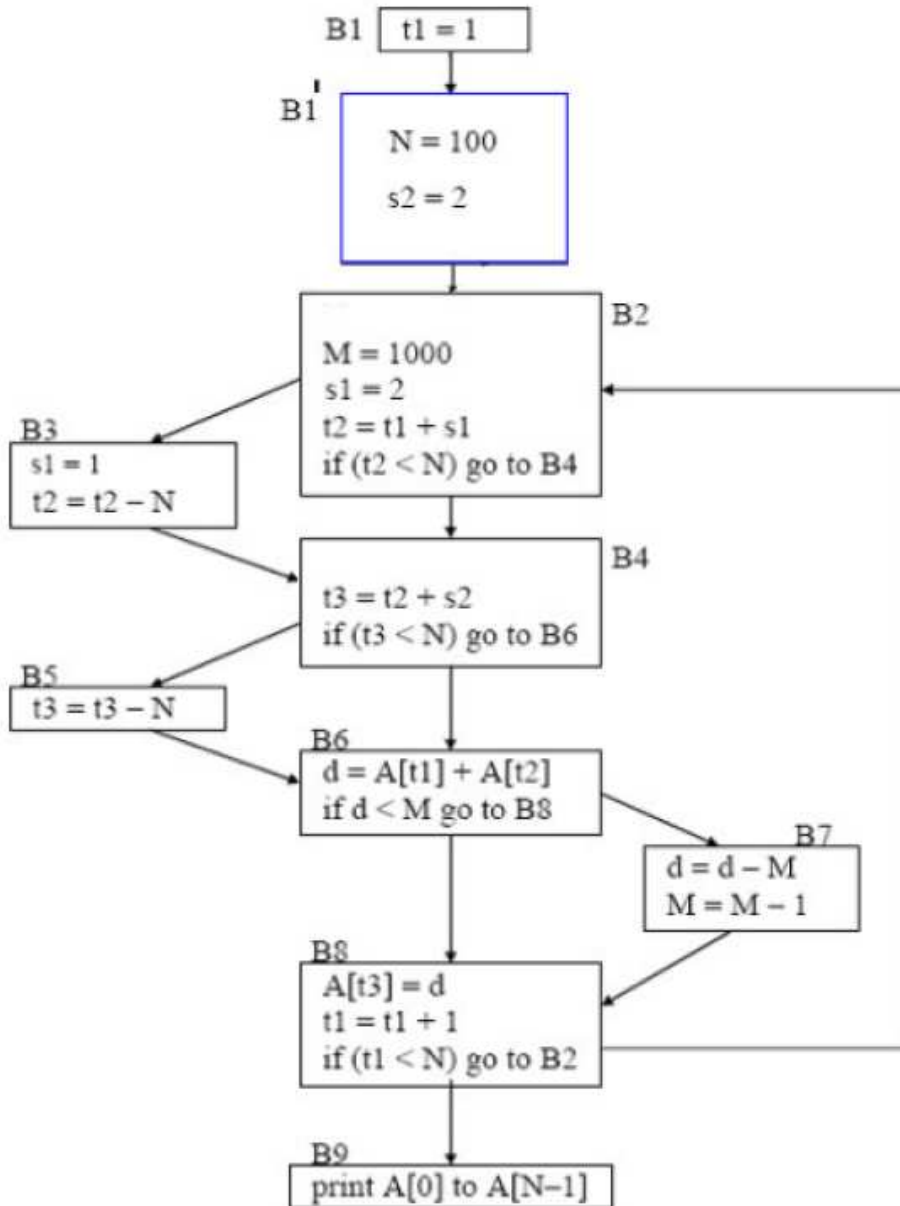
$S1 = 2$

B3:

$S1 = 1$

The reason is that there are other definitions in the loop.

The new code after code motion is shown below.



3.

(b) Perform copy propagation. You need to do necessary data flow analysis to make sure that the propagation can be done correctly.

In (6), the definition of $t3$ is only from (4), $t3$ and j have not redefined on the path, therefore, (6) becomes " $t5 = 4 * j$ ".

In (13), the definition of $t9$ is only from (11), $t9$ and j have not redefined on the path, therefore, (13) becomes " $t11 = 4 * j$ ".

(c) Perform dead code elimination. You need to do necessary analysis to make sure that the elimination can be done correctly. Also, assume that after the basic block, $A[i]$, for all i , are alive and no other variables are alive

After performing copy propagation in (a), the three address code in a basic block is listed as follows,

- (1) $t1 = j - 1$
- (2) $t2 = 4 * t1$

(3) temp = A[t2]
(4) t3 = j
(5) t4 = j + 1
(6) t5 = 4 * j
(7) t6 = A[t5]
(8) t7 = j - 1
(9) t8 = 4 * t7
(10) A[t8] = t6
(11) t9 = j
(12) t10 = j + 1
(13) t11 = 4 * j
(14) A[t11] = temp

Then perform liveness analysis as below,

P(14) = { }
P(13) = { temp, t11 }
P(12) = { temp, j }
P(11) = { temp, j }
P(10) = { temp, j }
P(9) = { temp, j, t6, t8 }
P(8) = { temp, j, t6, t7 }
P(7) = { temp, j, t6 }
P(6) = { temp, j, t5 }
P(5) = { temp, j }
P(4) = { temp, j }
P(3) = { temp, j }
P(2) = { j, t2 }
P(1) = { j, t1 }
P(0) = { j }

In (4), t3 is defined but not in P(4), thus (4) can be eliminated.

Similarly,

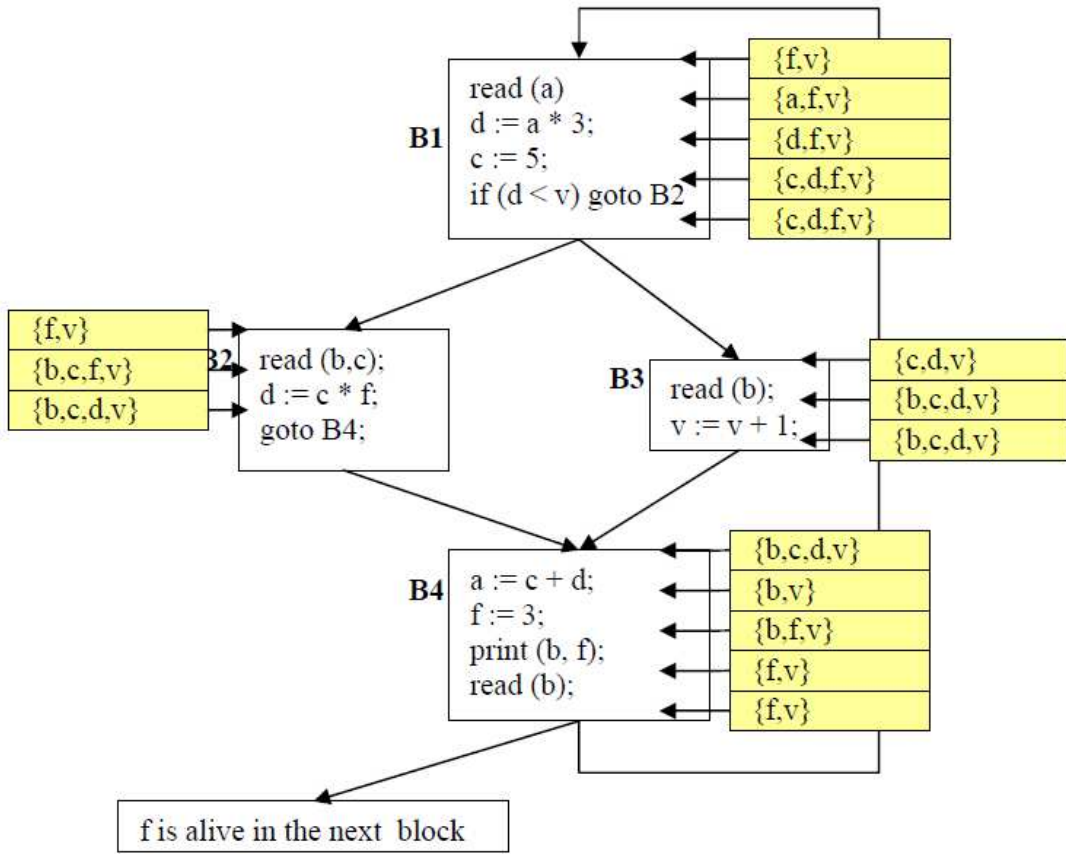
In (5), t4 is defined but not in P(5), thus (5) can be eliminated.

In (11), t9 is defined but not in P(4), thus (11) can be eliminated.

In (12), t10 is defined but not in P(4), thus (12) can be eliminated.

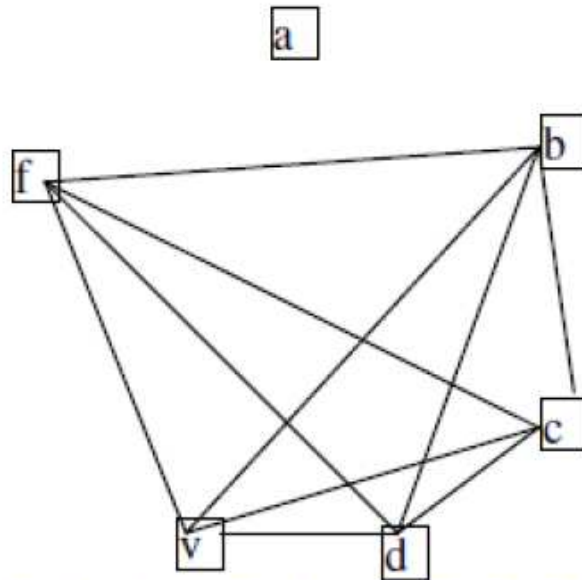
4.

(a) Perform liveness analysis. Show the Live sets at each point of the CFG.
Construct the interference graph using the global register allocation algorithm.



(b) Assume that the system has five registers. Perform color assignment. Determine the register for each variable.

5-color: Remove "a" first



Then all nodes have < 5 edges. It is 5-colorable.

So the registers for each variable are listed as follows,

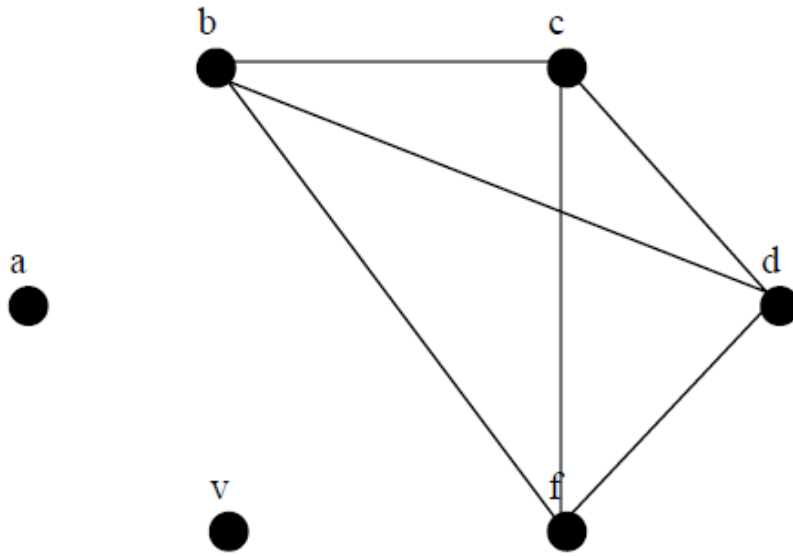
a: R4	b: R4	c: R3
d: R2	f: R1	v: R0

(c) Assume that the system has only three registers. Perform color assignment and determine the register for each variable. Note that you need to spill and rewrite the code and redo the register assignment.

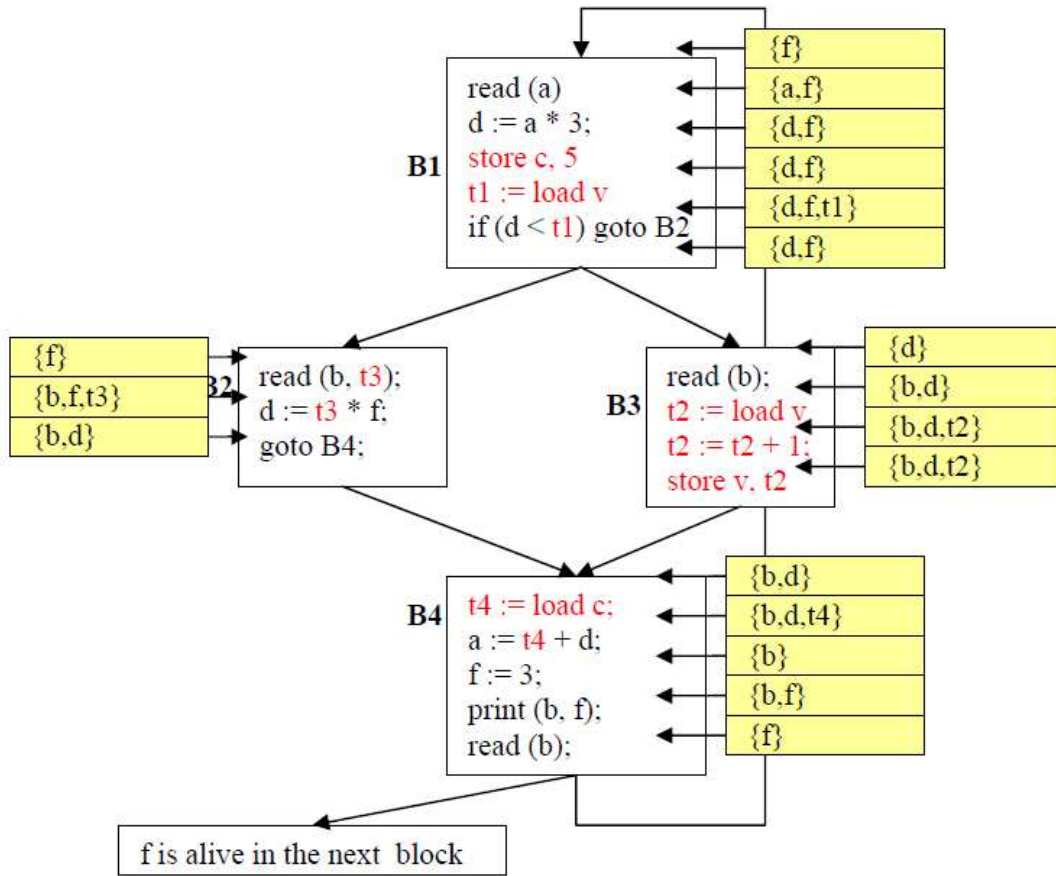
3-color: Remove "a" first

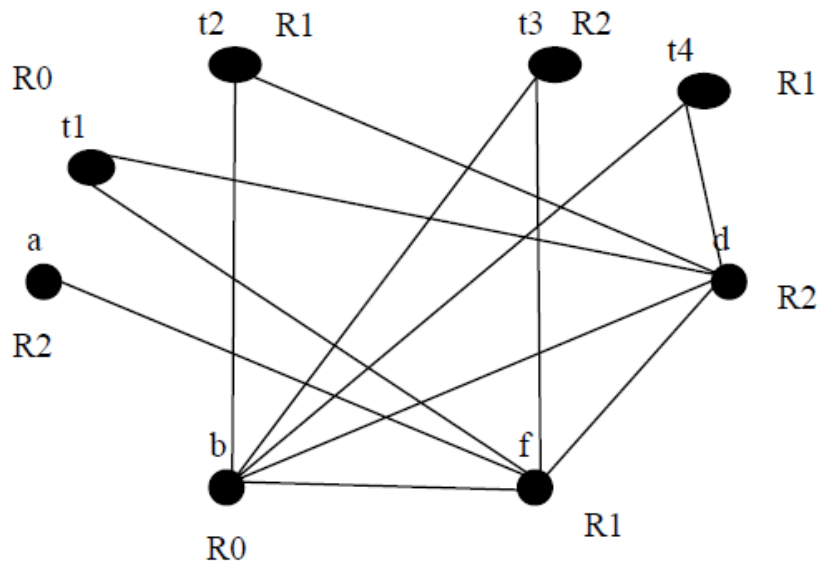
Since the node with 2 edges can't be found \Rightarrow need to spill

Choose to spill v



Since the node with 2 edges still can't be found \Rightarrow need to spill.
 Choose to spill c. Now it is 3-colorable
 Next we check whether the spilled nodes can have their registers when loaded back





The above interference graph is colorable by 3 colors.

So, spilling c and v will work.

So the registers for each variable are listed as follows

a: R2

b: R0

c: spill

d: R2

f: R1

v: spill

t1: R0: is used for temporary loading and using v

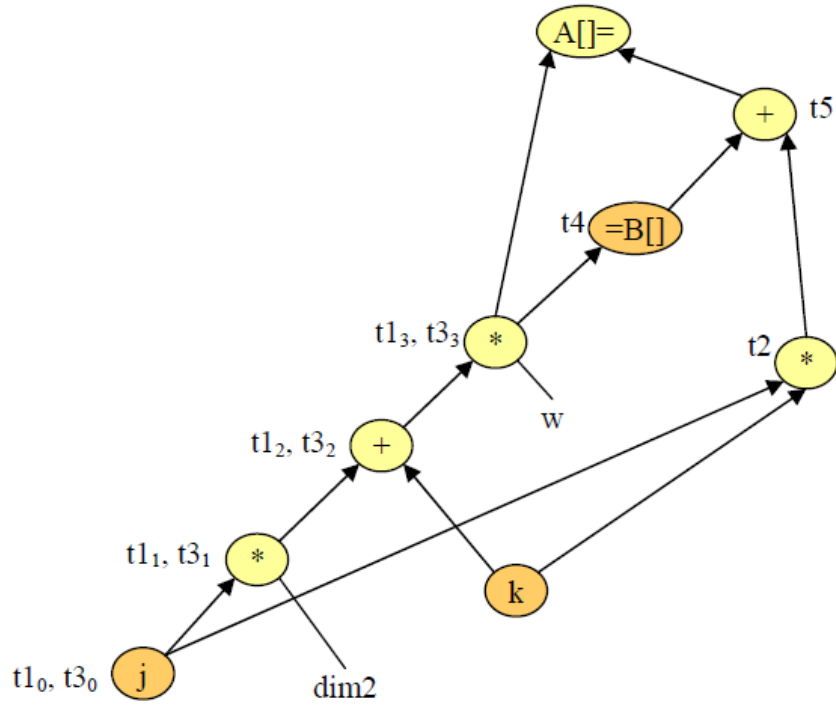
t2: R1: is used for temporary loading and using v

t3: R2: are used for temporary loading and using c

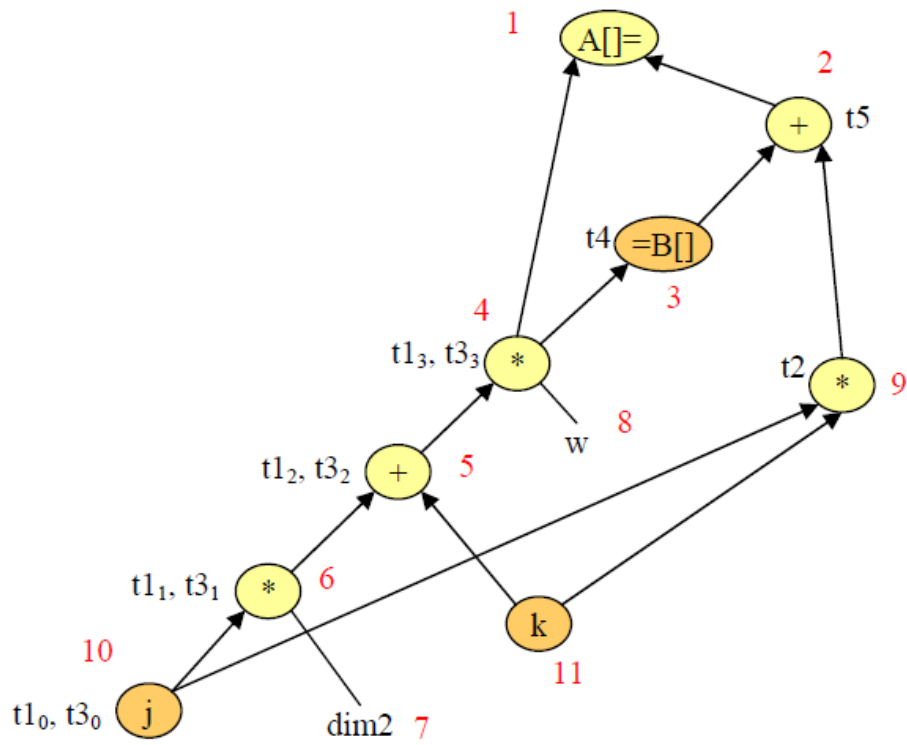
t4: R1: are used for temporary loading and using c

5

(a) Construct a dag from the three address code such that subexpressions and redundant variables are eliminated.



(b) Schedule the dag based on the numbering scheme. Run a global register allocation algorithm. Generate code based on the register allocation and instruction schedule you derived. Your code should make sure that the content of arrays A and B are permanently modified.



$t_1 := j$	\leftarrow	$\{j, k\}$
$t_1 := t_1 * dim2$	\leftarrow	$\{t_1, j, k\}$
$t_1 := t_1 + k$	\leftarrow	$\{t_1, j, k\}$
$t_1 := t_1 * w$	\leftarrow	$\{t_1, j, k\}$
$t_2 := j * k$	\leftarrow	$\{t_1, j, k\}$
$t_3 := j$	\leftarrow	$\{t_1, t_2, j, k\}$
$t_3 := t_3 * dim2$	\leftarrow	$\{t_1, t_2, t_3, k\}$
$t_3 := t_3 + k$	\leftarrow	$\{t_1, t_2, t_3, k\}$
$t_3 := t_3 * w$	\leftarrow	$\{t_1, t_2, t_3\}$
$t_4 := B[t_3]$	\leftarrow	$\{t_1, t_2, t_3\}$
$t_5 := t_2 + t_4$	\leftarrow	$\{t_1, t_2, t_4\}$
$A[t_1] := t_5$	\leftarrow	$\{t_1, t_5\}$
	\leftarrow	$\{\}$

Or you can do eliminate duplicate variables before running the global register allocation algorithm

Need 4 registers

r4 := load k

r3 := load j

r1 := r3

r2 := r3 * r4

r1 := r1 * dim2

r1 := r4 + r1

r1 := r1 * w

r3 := B[r1]

r2 := r3 + r2

A[r1] := r2