```
function f (n : integer) : integer;
begin
   if n <= 0 then
      f := 1
   else
      f := n * f (n - 1)
end;
```

```
function f(n);
begin
  f := 0;
  if n <= 0 then
    f := 1
  else
    f := n * f((-1 +)n)
end;
```

```
function f(%0) : %1
var %0, %1, %2, %3
entry f6
exit f0
f6: li    %1, 0      →f5
f5: blez  %0         →f4, f3
f3: addiu %3, %0, -1 →f2
f2: call  %2, f(%3)  →f1
f1: mul   %1, %0, %2 →f0
f4: li    %1, 1      →f0
```

```
procedure f(1)
var %0, %1, %2, %3, %4, %5, %6
entry f11
f11: newframe          →f10
f10: move  %6, $ra     →f9
f9 : move  %5, $s1     →f8
f8 : move  %4, $s0     →f7
f7 : move  %0, $a0     →f6
f6 : li    %1, 0       →f5
f5 : blez  %0          →f4, f3
f3 : addiu %3, %0, -1  →f2
f2 : j                 →f20
f20: move  $a0, %3     →f19
f19: call  f(1)        →f18
f18: move  %2, $v0     →f1
f1 : mul   %1, %0, %2  →f0
f0 : j                 →f17
f17: move  $v0, %1     →f16
f16: move  $ra, %6     →f15
f15: move  $s1, %5     →f14
f14: move  $s0, %4     →f13
f13: delframe          →f12
f12: jr    $ra         (xmits $v0)
f4 : li    %1, 1       →f0
```

```
procedure f(1)
var 8
entry f11
f11: newframe              →f10
f10: sets  local(0), $ra   →f9
f9 : j                     →f8
f8 : sets  local(4), $s0   →f7
f7 : move  $s0, $a0        →f6
f6 : j                     →f5
f5 : blez  $s0             →f4, f3
f3 : addiu $a0, $s0, -1    →f2
f2 : j                     →f20
f20: j                     →f19
f19: call  f               →f18
f18: j                     →f1
f1 : mul   $v0, $s0, $v0   →f0
f0 : j                     →f17
f17: j                     →f16
f16: gets  $ra, local(0)   →f15
f15: j                     →f14
f14: gets  $s0, local(4)   →f13
f13: delframe              →f12
f12: jr    $ra
f4 : li    $v0, 1          →f0
```

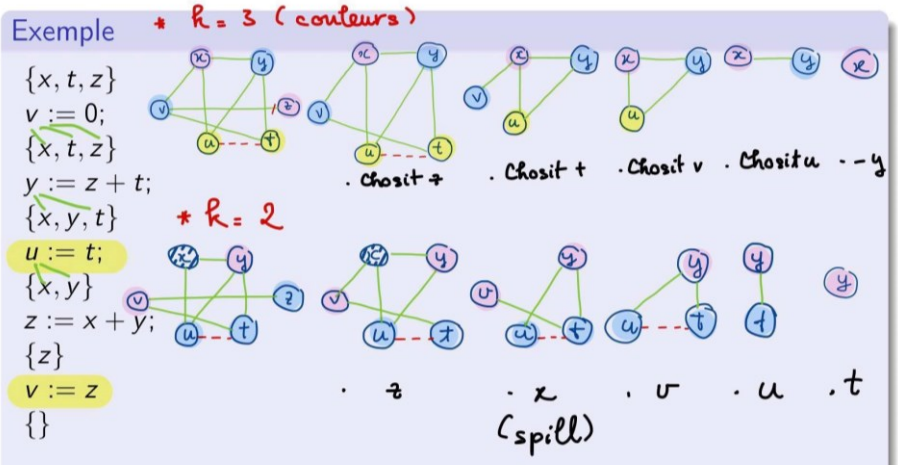
```
procedure f(1)
var 8
f11:
newframe
sets  local(0), $ra
sets  local(4), $s0
move  $s0, $a0
blez  $s0, f4
addiu $a0, $s0, -1
call  f
mul   $v0, $s0, $v0
f16:
gets  $ra, local(0)
gets  $s0, local(4)
delframe
jr    $ra
f4:
li    $v0, 1
j     f16
```

```
f17:
addiu $sp, $sp, -8
sw    $ra, 4($sp)
sw    $s0, 0($sp)
move  $s0, $a0
blez  $s0, f4
addiu $a0, $s0, -1
jal   f17
mul   $v0, $s0, $v0
```

```
f28:
lw    $ra, 4($sp)
lw    $s0, 0($sp)
addiu $sp, $sp, 8
jr    $ra
f4:
li    $v0, 1
j     f28
```

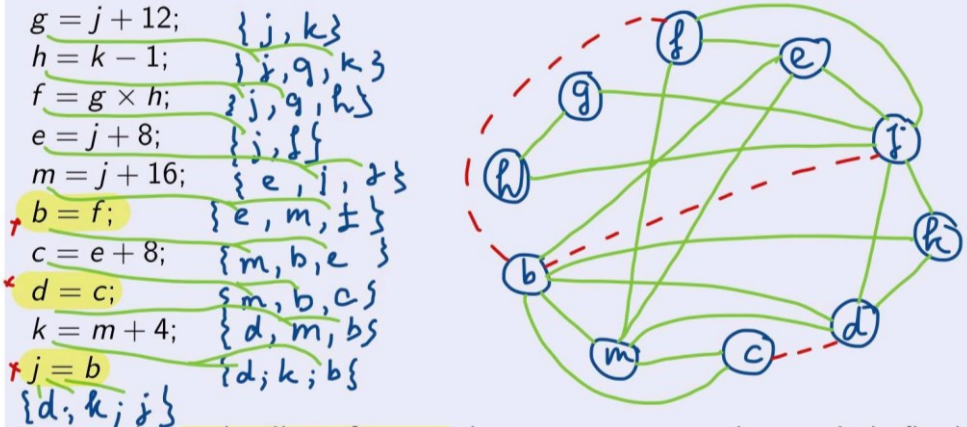
Graphe d'interférences

Exemple



Interférences : $(v,x),(v,t),(v,z),(y,x),(y,t),(u,x),(u,y)$.
Préférences : (u,t).

Soit le programme suivant



Dessiner le graphe d'interférences de ce programme sachant qu'à la fin du programme, les variables vivantes sont d, k, et j ?

* arête de préférence : utilise le même registre

lw dest, offset(base) -> đọc

sw source, offset(base) -> copy nội dung của source và cho vào adresse offset(base)

li dest, constant

addi dest, source, constant

move dest, source -> cho source vào dest

neg dest, source -> vd: 10 sẽ thành -10

add dest, source1, source2

de même pour sub, mul, div

slt dest, source1, source2 -> set on less than if s1 < s2 then dest = 1; else dest = 0

bgtz source, address -> si contenu du source > 0, saute à address(vd : **bgtz** %1, f10)

bgez source, address (branch on greater than or equal to zero)

blez source, address (branch on less than or equal to 0)

bltz source, address (branch on less than zero)

beq source1, source2, address -> branch on equal (vd: **beq** %0, %1, f10)

blt source1, source2, address -> branch on less than

bne source1, source2, address -> branch on not equal to

j address -> vd: j f10

jal address -> lưu adresse của instruction sau vào $ra và nhảy đến adresse “address”

jr target -> jump register (vd: **jr** $ra)

syscall -> appel du système pour afficher les arguments du register $v0 và $a0 - $a3

vd : MIPS

```
.data
positive: .asciiz "positive\n"
negative: .asciiz "negative\n"

.text
main:   li $v0, 5
        syscall
        move $a0, $v0
        jal pos
        li $v0, 10
        syscall
pos:    li $t0, 0
        blt $a0, $t0, neg
        li $v0, 4
        la $a0, positive
        syscall
        j end
neg:    li $v0, 4
        la $a0, negative
        syscall
end:    jr $ra
```

Ici **jal pos** lưu instruction « li $v0, 10 » vào $ra và nhảy đến **pos :** và sau khi **pos :** xong thì nhảy đến **end :** và end sẽ **jr $ra** tức là nhảy về lại instruction « li $v0, 10 »