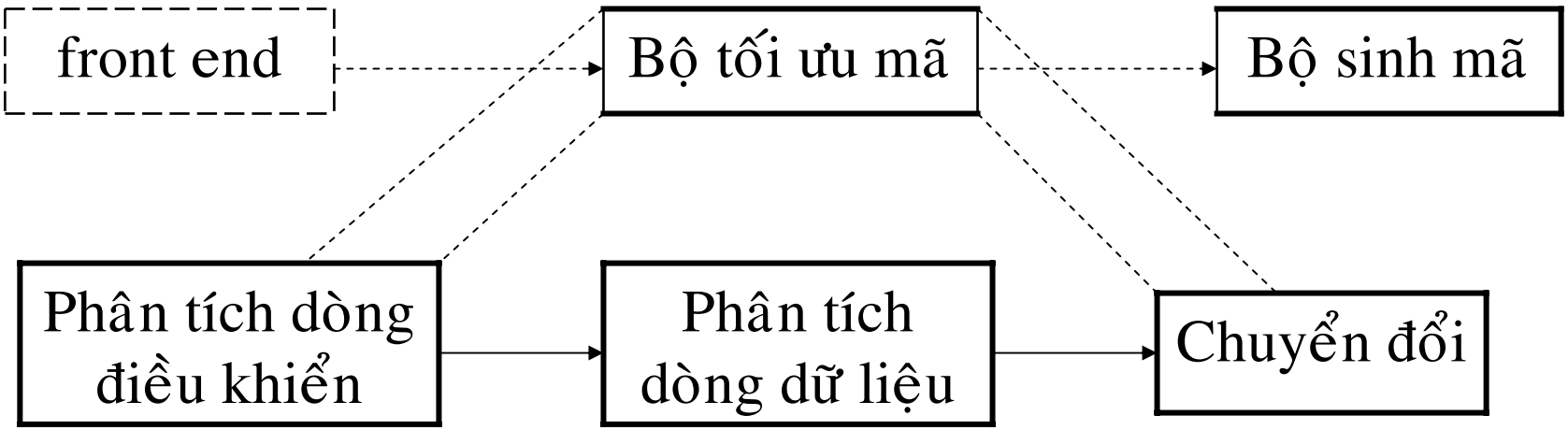
CHƯƠNG 10

TỐI ƯU MÃ

10.1. Giới thiệu

- *Tiêu chuẩn chuyển mã tốt*

- *Tổ chức của trình biên dịch tối ưu*



Hình 10.1. Tổ chức của bộ tối ưu mã

Mã trung gian

Thí dụ 10.1. Chuyển đổi sang mã trung gian ba địa chỉ cho đoạn chương trình trong ngôn ngữ Pascal

```
for i := n – 1   down to 1 do
        for j:= 1 to i do
            if A [j] > A [j + 1] then
            begin
                temp := A [j];
                A [j] := A [j + 1];
                A [j + 1] := temp;
            end;
```

Giả sử mỗi ô nhớ là 4 byte. Địa chỉ nền của dãy A vậy địa chỉ phần tử thứ j của dãy A là: *addr(A[j]) = addr(A) + (j – 1) * 4*.

(1) i = n - 1

(2) ij i < 1 goto (31)

(3) j = 1

(4) if j > i goto (29)

(5) t_1 = j - 1

(6) t_2 = 4 * t_1

(7) t_3 = A [t_2]

(8) t_4 = j + 1

(9) t_5 = t_4 - 1

(10) t_6 = 4 * t_5

(11) t_7 = A [t_6]

(12) ij t_3 < t_7 goto (27)

(13) t_8 = j - 1

(14) t_9 = 4 * t_8

(15) temp = A [t_9]

(16) t_{10} = j + 1

(17) t_{11} = t_{10} - 1

(18) t_{12} = 4 * t_{11}

(19) t_{13} = A [t_{12}]

(20) t_4 = j - 1

(21) t_{15} = 4 * t_{14}

(22) A [t_5] = t_{13}

(23) t_{16} = j + 1

(24) t_{17} = t_{16} - 1

(25) t_{18} = 4 * t_{17}

(26) A [t_{18}] = temp

(27) j = j + 1

(28) goto (4)

(29) i = i - 1

(30) goto 2

**** Khối cơ bản***

Thí dụ 10.2. Đoạn mã trung gian sau được xác định 4 khối cơ bản

(1)	read L	BB_1
(2)	n := 0	
(3)	k := 0	
(4)	m := 1	
(5)	k := k + m	BB_2
(6)	c := k > L	
(7)	if (c) goto (11)	
(8)	n := n + 1	BB_3
(9)	m := m + 2	
(10)	goto (5)	
(11)	write n	BB_4

10.2. Phân tích dòng dữ liệu

Các cấu trúc điều khiển như if, while, for gây ra sự rẽ nhánh của chương trình. Xác định được sự rẽ nhánh sẽ xác định được sự thay đổi trị của biến trong chương trình, từ đó sử dụng các biến này trong quá trình tối ưu hóa.

10.2.1. Mục đích

Xác định cấu trúc điều khiển của chương trình là:

- mô tả các con đường thực hiện chương trình

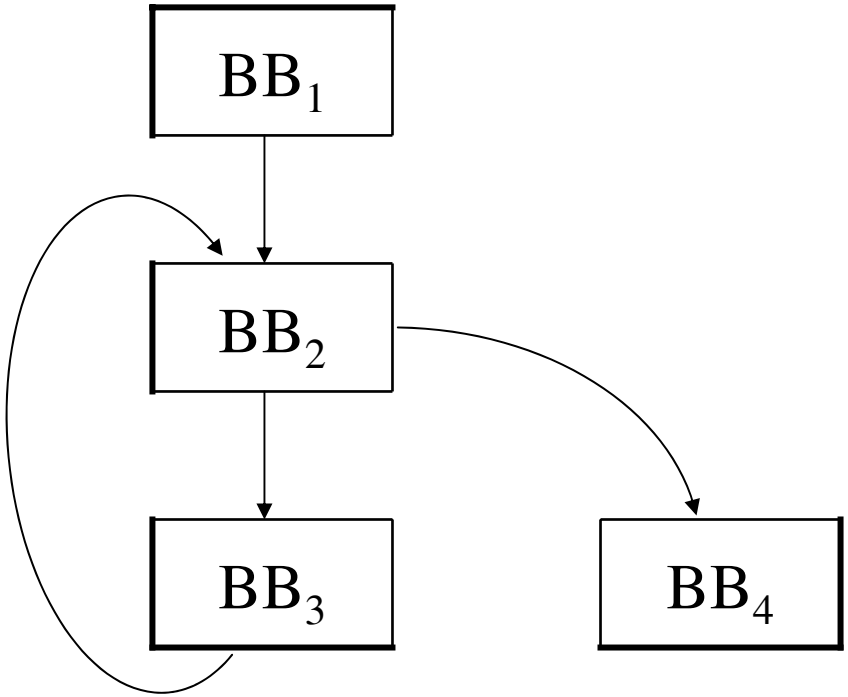
- xác định các vòng lặp

10.2.2. Đồ thị dòng điều khiển (Control Flow Graphs)

Định nghĩa:

Đồ thị dòng điều khiển (CFG) của một chương trình là một đồ thị có hướng, được ký hiệu G = (N, E) mà trong đó N là các khối cơ bản, E là tập cạnh thể hiện cho dòng điều khiển giữa các khối cơ bản.

Thí dụ 10.3. Đoạn mã trung gian (gồm 4 khối cơ bản) ở thí dụ 10.2 được biểu diễn thành đồ thị dòng dữ liệu.



Hình 10.2. Đồ thị dòng điều khiển

10.2.3. Successor, predcessor của một khối cơ bản

Cho một đồ thị dòng điều khiển G = (N, E) và một khối cơ bản b ∈ N, xác định successor và predcessor cho khối cơ bản b như sau:

* ***Successor của b***, ký hiệu succ (b) là tập các khối cơ bản n, mà có thể đạt đến từ b trên 1 cạnh succ (b) = $\{n \in N \mid (b, n) \in E\}$

như ở thí dụ 10.3: succ (BB_1) = $\{BB_2\}$;

succ (BB_2) = $\{BB_3, BB_4\}$, succ (BB_3) = $\{BB_2\}$

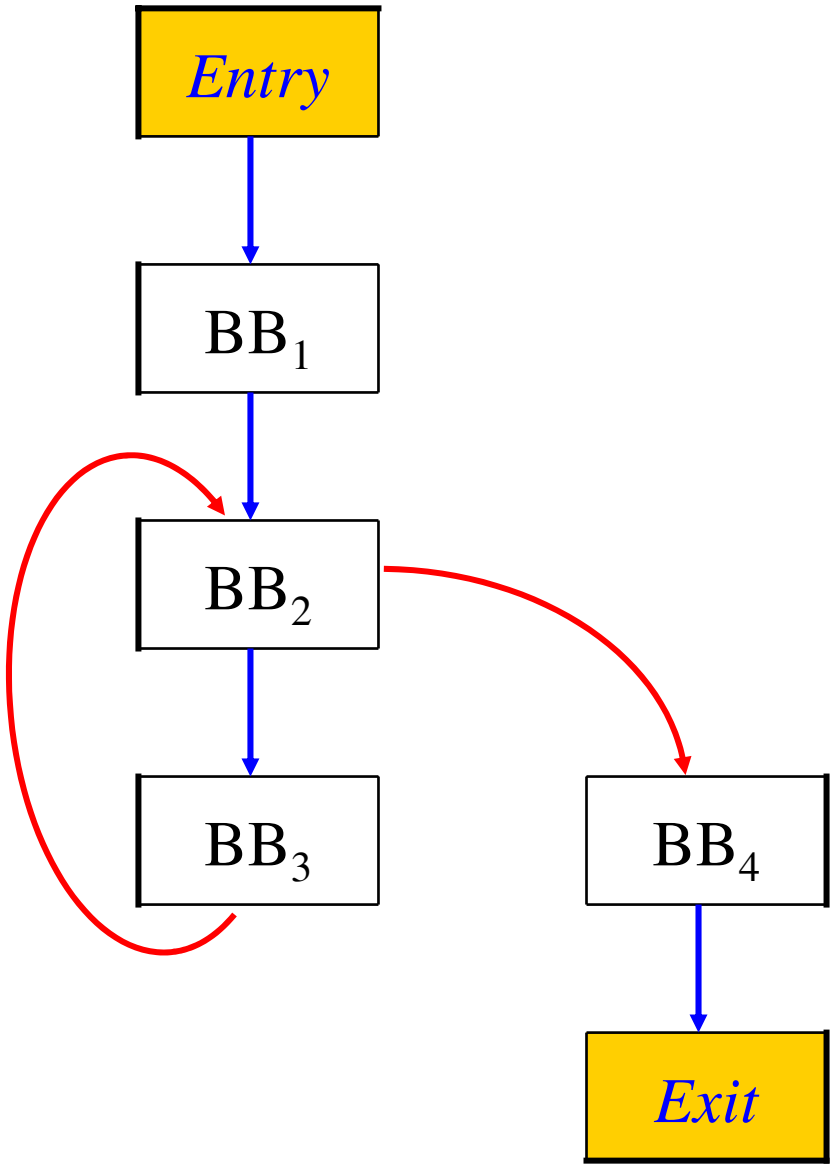
* ***Predcessor của b***, ký hiệu pred (b) là tập các khối cơ bản m, mà có thể đạt đến b trên 1 cạnh pred (b) = $\{m \in N \mid (m, b) \in E\}$

như ở thí dụ 10.3: pred (BB_2) = $\{BB_1 , BB_3\}$

pred (BB_3) = $\{BB_2\}$, pred (BB_4) = $\{BB_2\}$

- ***Entry của G***: là một nút không có predcessor
- ***Exit của G***: là một nút không có successor
- ***nút rẽ*** (branch node) trong G là nút có nhiều hơn một successor
- ***nút hợp*** (join node) trong G là nút có nhiều hơn một predcessor

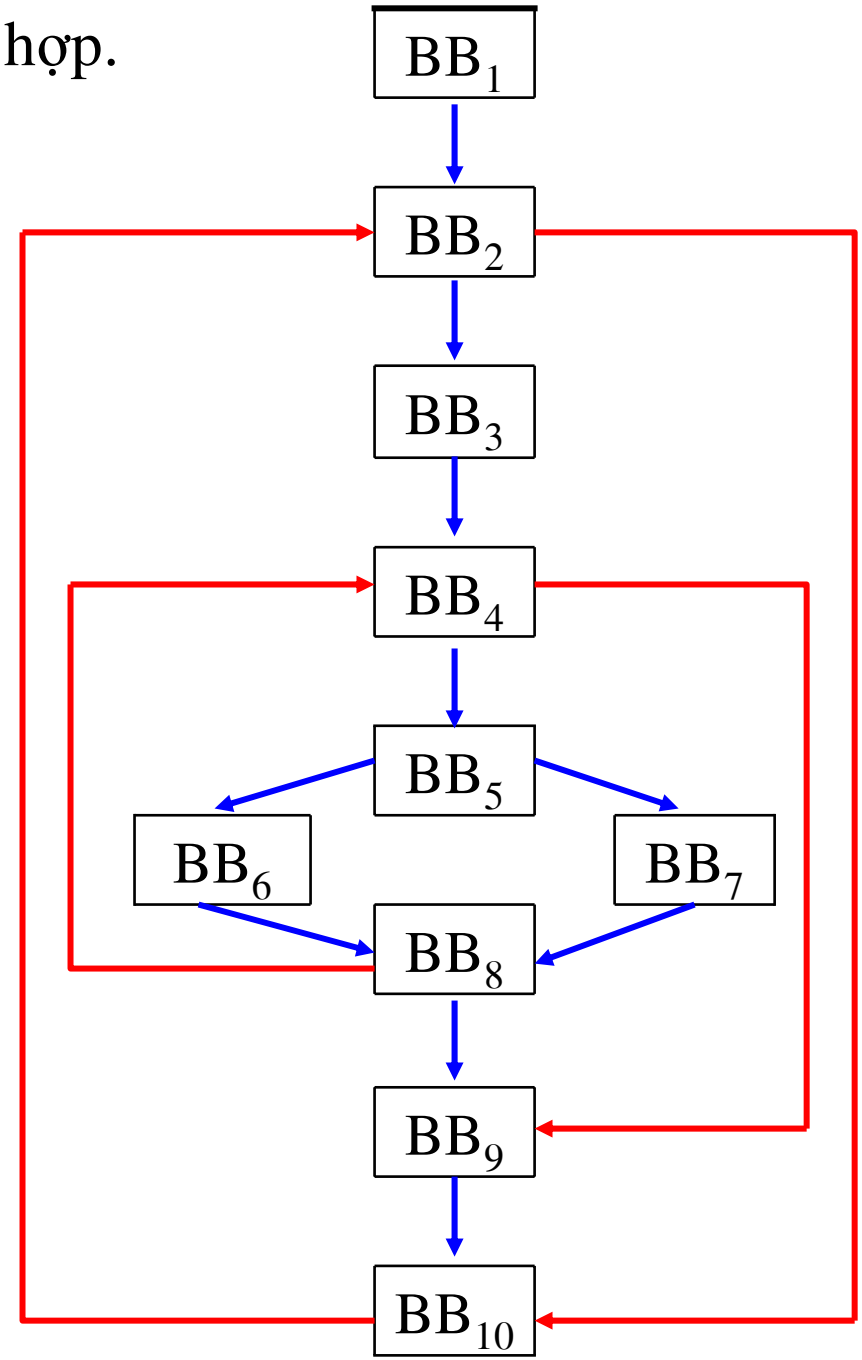
Đồ thị dòng điều khiển ở thí dụ 10.3 được thêm 2 nút Entry, Exit.

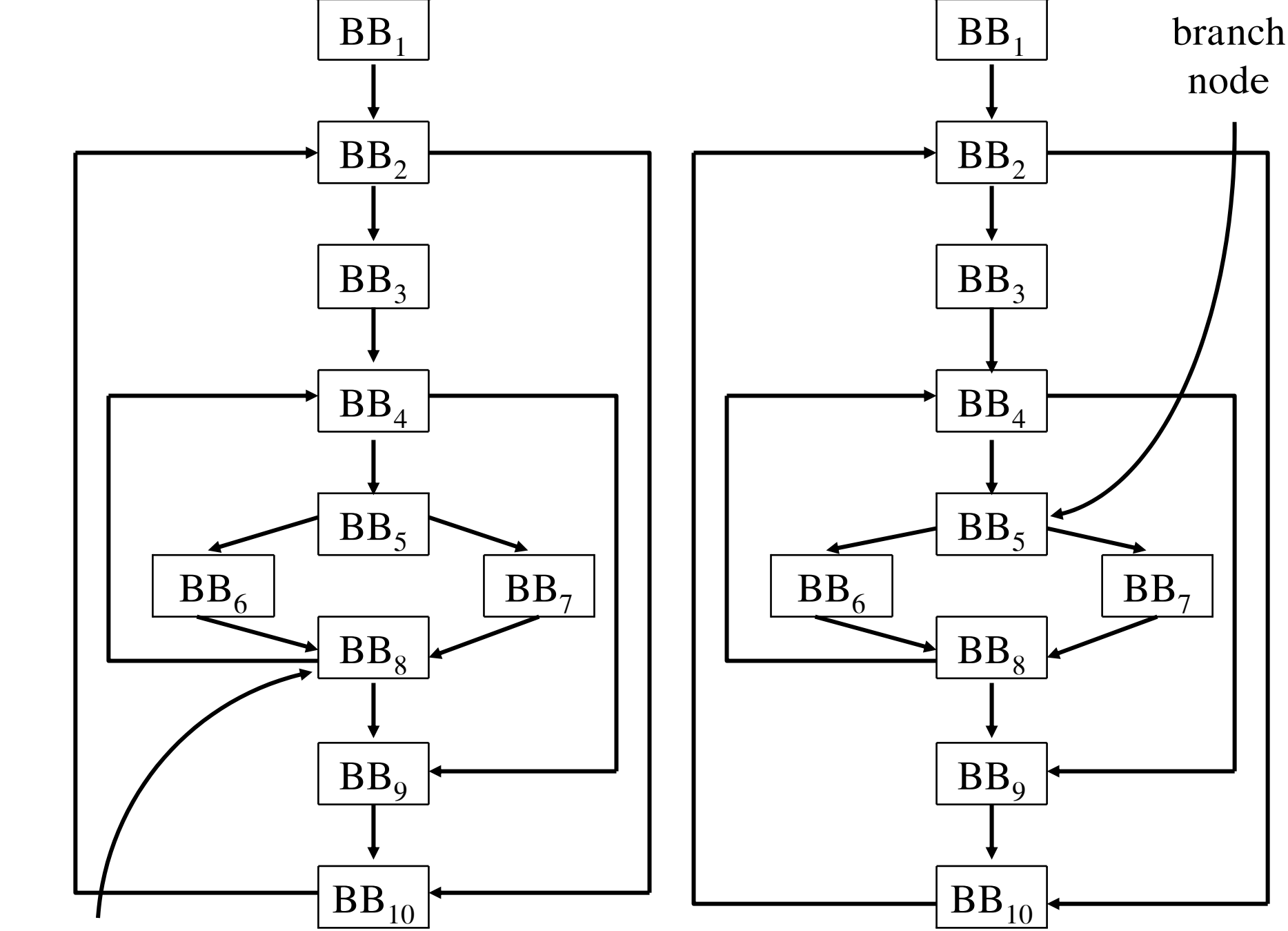


Hình 10.3. Nút Entry và Exit trong G

Thí dụ minh họa các nút rẽ nhánh và hợp.

(1) ⟶ i := 1
(2) ⟶ if(I>n) goto (15)
(3) ⟶ t := 0
(4) j := 1
(5) ⟶ if(j>n) goto (13)
(6) ⟶ tmp := te + ts
(7) if(tmp < 0) goto (10)
(8) ⟶ $t_1 := t_1 + t_s$
(9) goto (11)
(10) ⟶ $t_1 := 4*j$
(11) ⟶ j := j+1
(12) goto (5)
(13) ⟶ i := I+1
(14) goto (2)
(15) ⟶ $t_1 := 0$



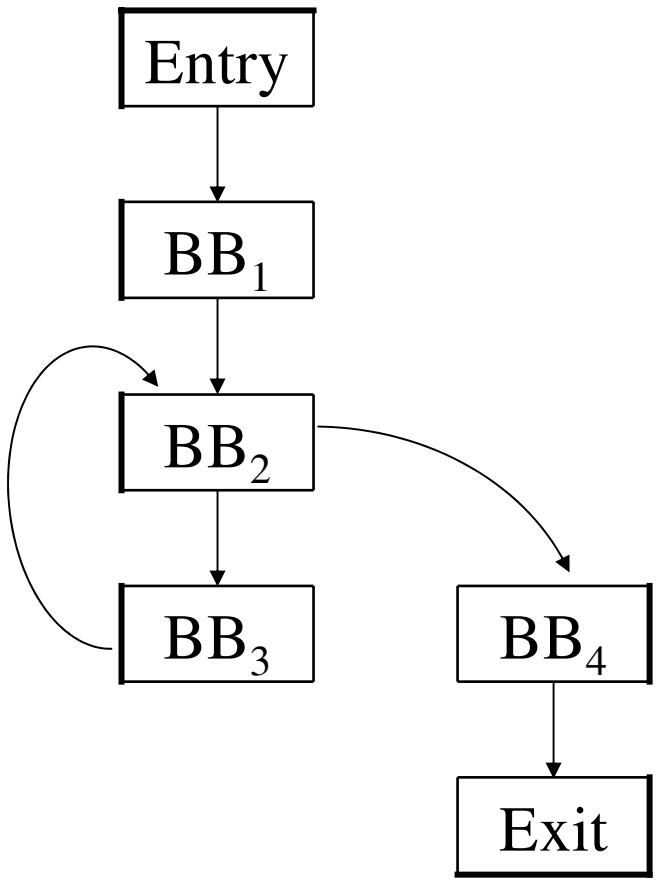


Hình 10.4. Các nút rẽ nhánh và hợp

10.2.4. Chi phối (dominater)

▪ *Định nghĩa dominater*: nút n của G chi phối nút n', ký hiệu n dom n' (hay n → n'), nếu mọi con đường từ nút Entry của G đến n' đều phải đi qua n. Với định nghĩa này thì mọi nút n chi phối chính nó. Nút là điểm vào vòng lặp sẽ chi phối các nút trong vòng lặp.

$BB_1 \rightarrow BB_1$;
$BB_1 \rightarrow BB_2$;
$BB_1 \rightarrow BB_3$;
$BB_1 \rightarrow BB_4$
$BB_2 \rightarrow BB_2$;
$BB_2 \rightarrow BB_3$;
$BB_2 \rightarrow BB_4$
$BB_3 \rightarrow BB_3$
$BB_4 \rightarrow BB_4$



- *properdominate*

N là proper dominate n’ , ký hiệu $n \rightarrow_p n'$, nếu $n \rightarrow n'$ và $n \neq n'$ như thí dụ ở trên thì $BB_1 \rightarrow_p BB_2$; $BB_1 \rightarrow_p BB_3$; $BB_1 \rightarrow_p BB_4$; $BB_2 \rightarrow_p BB_3$; $BB_2 \rightarrow_p BB_4$

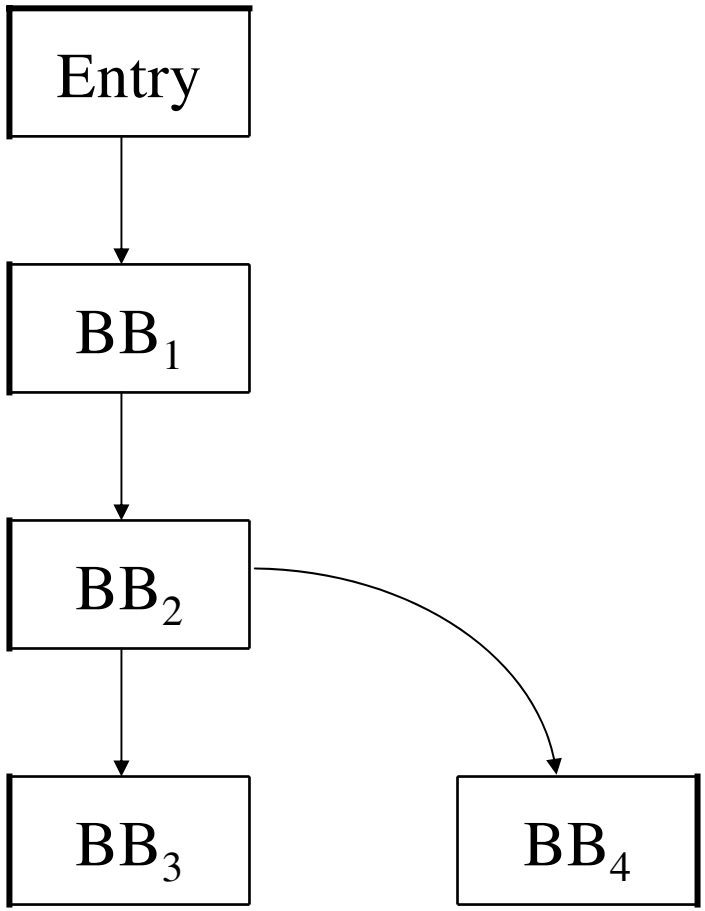
- *direct dominate*

Nút n được gọi là direct dominate n’ , ký hiệu $n \rightarrow_d n'$, nếu $n \rightarrow_p n'$ và không tồn tại $n'' \in N$ mà $n \rightarrow_p n''$.

$BB_1 \rightarrow_d BB_2$; $BB_2 \rightarrow_d BB_3$; $BB_2 \rightarrow_d BB_4$

- *cây dominate (dominate tree)*; ký hiệu viết tắt DT, là cây mà nút gốc là Entry; nút thuộc G và cạnh là quan hệ direct dominator.

Ở thí dụ trên DT có dạng:



Hình 10.4. Cây dominate

Giải thuật 2.2: tìm các dominator.

Nhập: đồ thị dòng điều khiển G.

Xuất: tập dominator của mỗi nút thuộc G.

Giải thuật:

DOM (Entry) = Entry

DOM (ν) = N ∀ ν ∈ N - {Entry, Exit}

change = true

while (change) {change = false

for each ν ∈ N - {Entry, Exit} {

old DOM = DOM (ν)'

DOM (ν) = {ν} ∪ ∩ OM (p)

pepred (ν)

if (DOM (v) = old DOM) change = true

}

}

10.2.5. Direct dominator

Direct dominator của một nút n.

Giải thuật tìm direct dominator của một nút:

- Khởi động tập proper dominator của nút n

- Loại bỏ những nút mà nó là proper dominator những nút khác trong tập

Giải thuật 10.3. Tìm direct dominator

Nhập: đồ thị dòng G.

Xuất: direct dominator của mỗi nút của G.

Giải thuật:

for với mỗi nút n $\in$ N - {Entry} **do**

DOM_d (n) = DOM (n) - {n}

for với mỗi nút n $\in$ N - {Entry} **do**

for với mỗi nút s $\in$ DOM_d (n) - {s} **do**

if t $\in$ DOM_d (s) **then**

DOM_d (n) = DOM_d (n) - {t};

Ở thí dụ ở hình 10.1, ta có:

DOM_d (1) = {Entry}; DOM_d (2) = {1};

DOM_d (3) = {2}; DOM_d (4) = {2}

10.2.6. Post – dominator

Định nghĩa:

Cho đồ thị dòng điều khiển G = (N, E) và n, n' $\in$ N.

- Nút n được gọi là post – dominator của nút n', ký hiệu n $\rightarrow$ n' nếu mọi con đường từ n đến nút Exit chứa n'.

Ở thí dụ hình 10.3, ta có các post – dominator:

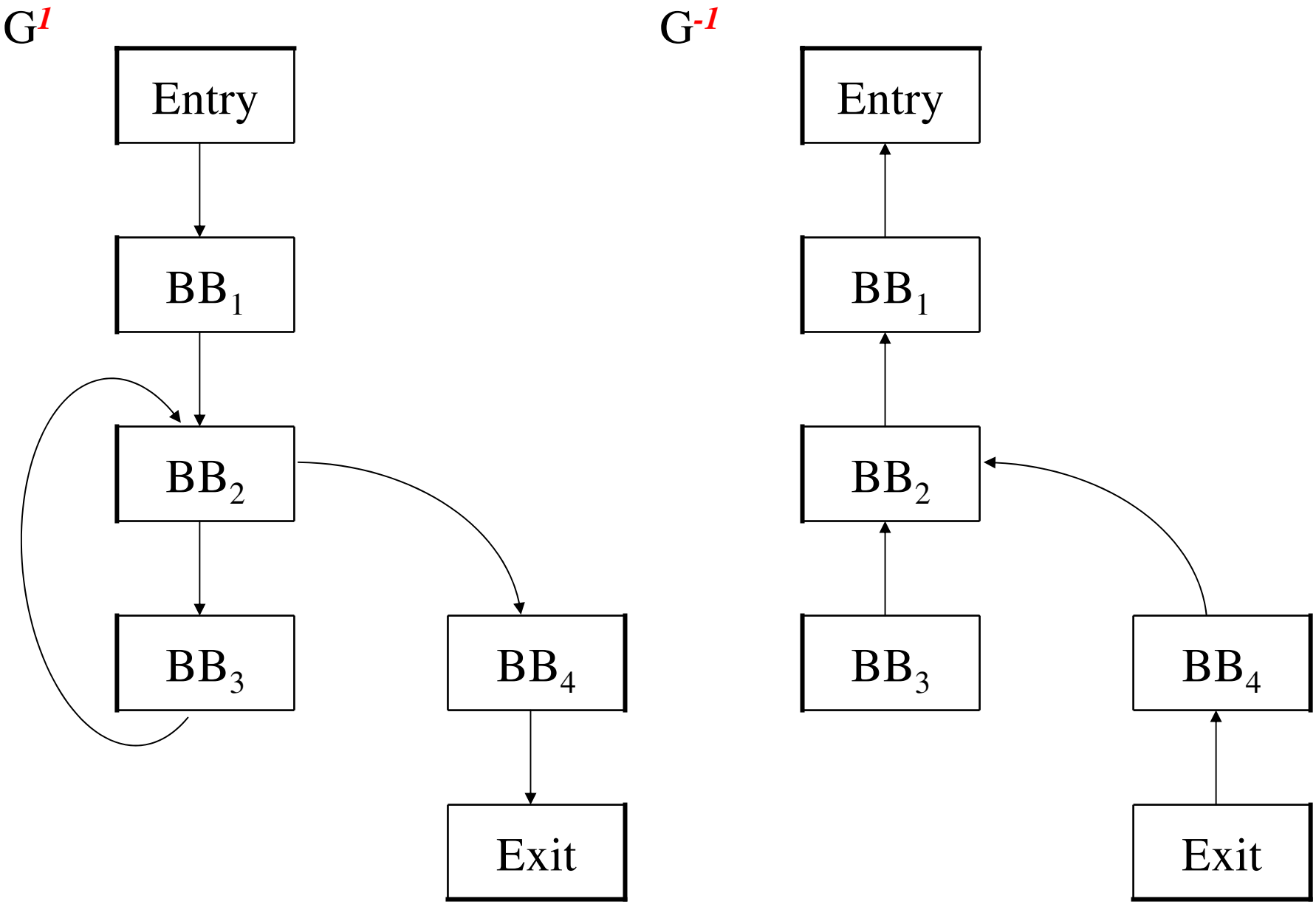
$$1 \leftarrow 1;\ 1 \leftarrow 2;\ 1 \leftarrow 3;\ 1 \leftarrow 4;\ 2 \leftarrow 2;\ 2 \leftarrow 3;\ 2 \leftarrow 4;\ 3 \leftarrow 3;\ 4 \leftarrow 4$$

- Nút n được gọi là direct post – dominator của nút n', ký hiệu n $\leftarrow$ n', nếu n $\leftarrow_{\mathbf{p}}$ n' và không tồn tại n" $\in$ N mà n $\leftarrow_{\mathbf{p}}$ n" $\leftarrow_{\mathbf{p}}$ n'.

- Nút n được gọi là proper post – dominator của nút n', ký hiệu là

n $\leftarrow_{p}$ n', nếu n $\leftarrow$ n' và n $\neq$ n'.

Thí dụ về post – dominator trong G là dominator trong G-1



10.2.7. Vòng lặp

Các yếu tố xác định vòng lặp tự nhiên:

- Một vòng lặp phải có 1 điểm vào đơn, gọi là header.

- Điểm vào header dominate tất cả các nút còn lại trong vòng lặp.

- Phải có ít nhất một cách lặp, nghĩa là phải có ít nhất một cạnh quay về header.

Giải thuật 10.3. Tìm vòng lặp

Nhập: đồ thị dòng G và một cạnh về t → h.

Xuất: vòng lặp bao gồm tất cả các nút trong vòng lặp tự nhiên t → h.

Phương pháp:

- Tìm dominator của mỗi nút trong CFG.

- Xác định cạnh vẽ.

- Tìm tất cả những nút liên quan đến cạnh vẽ.

- Để tìm cạnh vẽ: thực hiện duyệt cây CFG theo chiều sâu trước. Một cạnh lùi e = (t, h) ∈ E là cạnh lùi nếu h → t. Một cạnh lùi luôn là cạnh vẽ trong vòng lặp bằng cách sử dụng điều khiển có cấu trúc.

Giải thuật:

stack s = empty

set loop = {h}

insert – on – stack (t); /* stack s*

while S is not empty **do beg**

m = pop (s);

for each pred (m) **do**

insert – on – stack (p); **end**

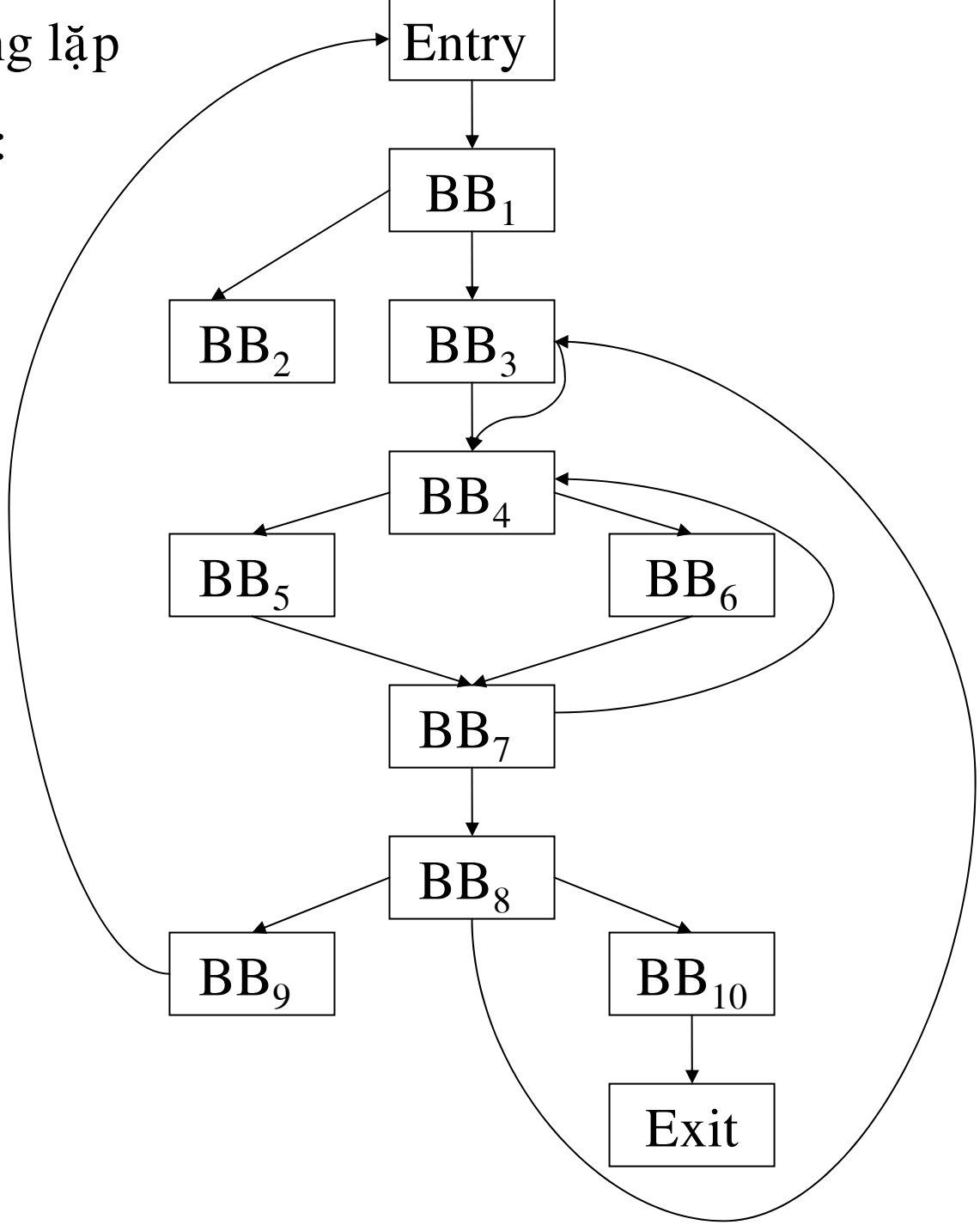
insert – on – stack (a) **begin**

if (a ∈ loop) **then begin** loop = loop ∪ {a} push – on – stack (a);

end

Thí dụ về tìm vòng lặp

Cho grap như sau:



Đầu tiên xác định cạnh về h → t.

Loop = {BB_3 , BB_8}, stack = {BB_8}

Loop = {BB_3 , BB_8}, stack = {}

Loop = {BB_3 , BB_8 , BB_7}, stack = {BB_7}

Loop = {BB_3 , BB_8 , BB_7}, stack = {}

Loop = {BB_3 , BB_8 , BB_7 , BB_5}, stack = {BB_5}

Loop = {BB_3 , BB_8 , BB_7 , BB_5 , BB_6}, stack = {BB_5 , BB_6}

Loop = {BB_3 , BB_8 , BB_7 , BB_5 , BB_6}, stack = {BB_5}

Loop = {BB_3 , BB_8 , BB_7 , BB_5 , BB_6 , BB_4}, stack = {BB_5 , BB_4}

Loop = {BB_3 , BB_8 , BB_7 , BB_5 , BB_6 , BB_4}, stack = {BB_5}

Loop = {BB_3 , BB_8 , BB_7 , BB_5 , BB_6 , BB_4}, stack = {}

Vòng lặp tìm được là {BB_3 , BB_4 , BB_5 , BB_6 , BB_7 , BB_8}, BB_3 là header, BB_8 là node kết thúc.

10.3. Phân tích dòng dữ liệu (Data Flow Analyst) – DFA

10.3.1. Mục đích của phân tích dòng dữ liệu

- Xác định dữ liệu được dùng trong chương trình.

- Sử dụng dữ liệu để trình biên dịch quyết định tối ưu mã.

- Trong một khối cơ bản: xác định tính hiệu quả trong câu lệnh, từ đầu đến cuối khối cơ bản.

10.3.2. Điểm và đường

Điểm là vị trí giữa hai phát biểu liền nhau.

Tồn tại điểm trước và sau phát biểu.

Thí dụ đoạn chương trình:

p_0

d_1 i := m - 1

p_1

d_2 j := n

p_2

d_3 a := u_1

p_3

Ở đây có 4 điểm p_0 trước d_1 , p_1 trước d_2 , trước d_3 , p_3 sau d_3

Đường: từ p_1 đến p_n là con đường đi từ p_1 đến điểm p_n trong chương trình.

10.3.3. Đạt đến sự định nghĩa (Reaching definition)

Định nghĩa của một biến x là tác vụ gán trị cho biến x.

Định nghĩa d cho một biến x được gọi là đạt đến một điểm p trong chương trình nếu tồn tại một con đường từ điểm ngay sau d đến p mà x không bị thay đổi trị bởi một định nghĩa của x dọc theo con đường này.

10.3.3.1. Tập Gen

Gen (b) là tập các định nghĩa ở trong b và đạt đến điểm kết thúc của b.

10.3.3.2. Tập Kill

Kill (b) là tập định nghĩa ở một khối cơ bản khác b nhưng bị thay đổi trong b (bởi một tác vụ trong b), v là biến được định nghĩa trong b, tập kill chứa tất cả các định nghĩa của v trong các khối cơ bản khác.

10.3.3.3. Sự cân bằng dòng dữ liệu

RDin (b): tập các định nghĩa mà đạt đến sự bắt đầu của b

RDout (b): tập các định nghĩa mà đạt đến sự kết thúc của b.

Công thức xác định:

$$RDin(b) = \bigcup_{i \in pred(b)} RDout(i)$$

$$RDout(b) = Gen(b) \cup [RDin(b) - Kill(b)]$$

Giải thuật: xác định việc đạt đến sự định nghĩa.

Nhập: đồ thị dòng G với tập Gen (b) và kill (b) đã được tính toán trước cho mỗi khối cơ bản b.

Xuất: RDin (b) và Rdout (b) cho mỗi khối cơ bản b

Giải thuật:

RDout (Entry) = ∅

RDout (b) = ∅ ∀ b ∈ N - {Entry, Exit}

/* Gen (b) thì tốt hơn */

change = true

while (change) {

change = false

for each b ∈ N - {Entry, Exit} {

oldout = RDout (b)

RDin (b) = $\cup$ RDout (i)

$i \in pred(b)$

RDout (b) = Gen (b) $\cup$ [RDin (b) – kill (b)]

if (RDout (b) $\neq$ oldout) change = true

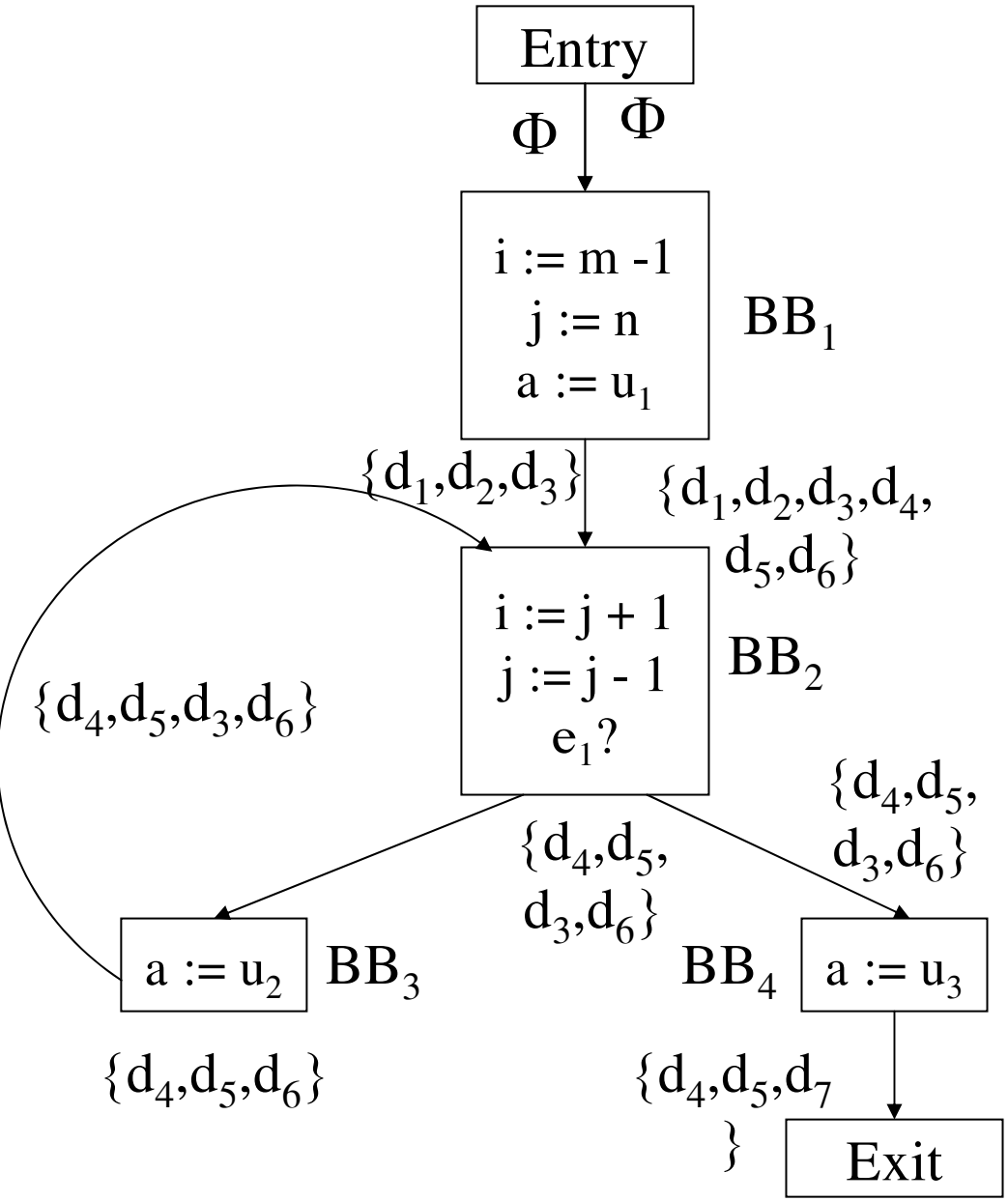
}

}

RDin (Exit) = $\cup$ RDout (i)

$i \in pred(Exit)$

Thí dụ:



$d_1 : i := m - 1$ $\quad d_5 : j := j - 1$

$d_2 : j := n$ $\quad d_6 : a := u_2$

$d_3 : a := u_1$ $\quad d_7 : a := u_3$

$d_4 : i := i + 1$

BB	Gen (BB)	Kill(BB)
1	$\{d_1,d_2,d_3\}$	$\{d_4,d_5,d_6,d_7\}$
2	$\{d_4,d_5\}$	$\{d_1,d_2\}$
3	$\{d_6\}$	$\{d_3,d_7\}$
4	$\{d_7\}$	$\{d_3,d_6\}$

$$RDin(b) = \bigcup_{i \in pred(b)} RDout(i)$$

$$RDout(b) = Gen(b) \cup [RDin(b) - kill(b)]$$

10.3.4. Biến sống

Biến v được gọi là sống tại điểm p trong chương trình nếu giá trị hiện tại của v được dùng trước khi v được gán giá trị mới hoặc trước khi chương trình kết thúc, ngược lại gọi là biến chết.

- Ứng dụng biến sống là xác định xem có cần lưu giữ trị của nó khi ra khỏi khối cơ bản, trong thanh ghi.

- Cần xác định biến sống ở điểm vào và ra của mỗi khối cơ bản.

10.3.4.1. Tập Use

Use (b) là tập các biến được sử dụng trước khi (hoặc có thể) được định nghĩa trong b.

10.3.4.2. Tập Def

Def (b) là tập các biến được định nghĩa trong b.

10.3.4.3. Sự cân bằng dòng dữ liệu

LVin (b): tập các biến sống tại điểm vào của b

LVout (b): tập các biến sống tại điểm ra của b

Công thức:

$$LVout(b) = \bigcup_{i \in succ(b)} LVin(i)$$

$$LVin(b) = Use(b) \cup [LVout(b) - Def(b)]$$

Giải thuật 3.2. Giải thuật tìm biến sống

Nhập: đồ thị dòng G với Def (b) và Use (b) được xác định trước.

Xuất: LVout (b) là tập biến sống tại điểm ra của khối cơ bản b.

Giải thuật:

LVin (Entry) = $\varnothing$

Lvin (b) = $\varnothing \; \forall$ b $\in$ N - {Entry, Exit}

change = true

while change {change = false}

for each b $\in$ N - {Entry, Exit} {

oldin = LVin (b)

LVout (b) = $\cup$ LVin (i)

$i \in succ(b)$

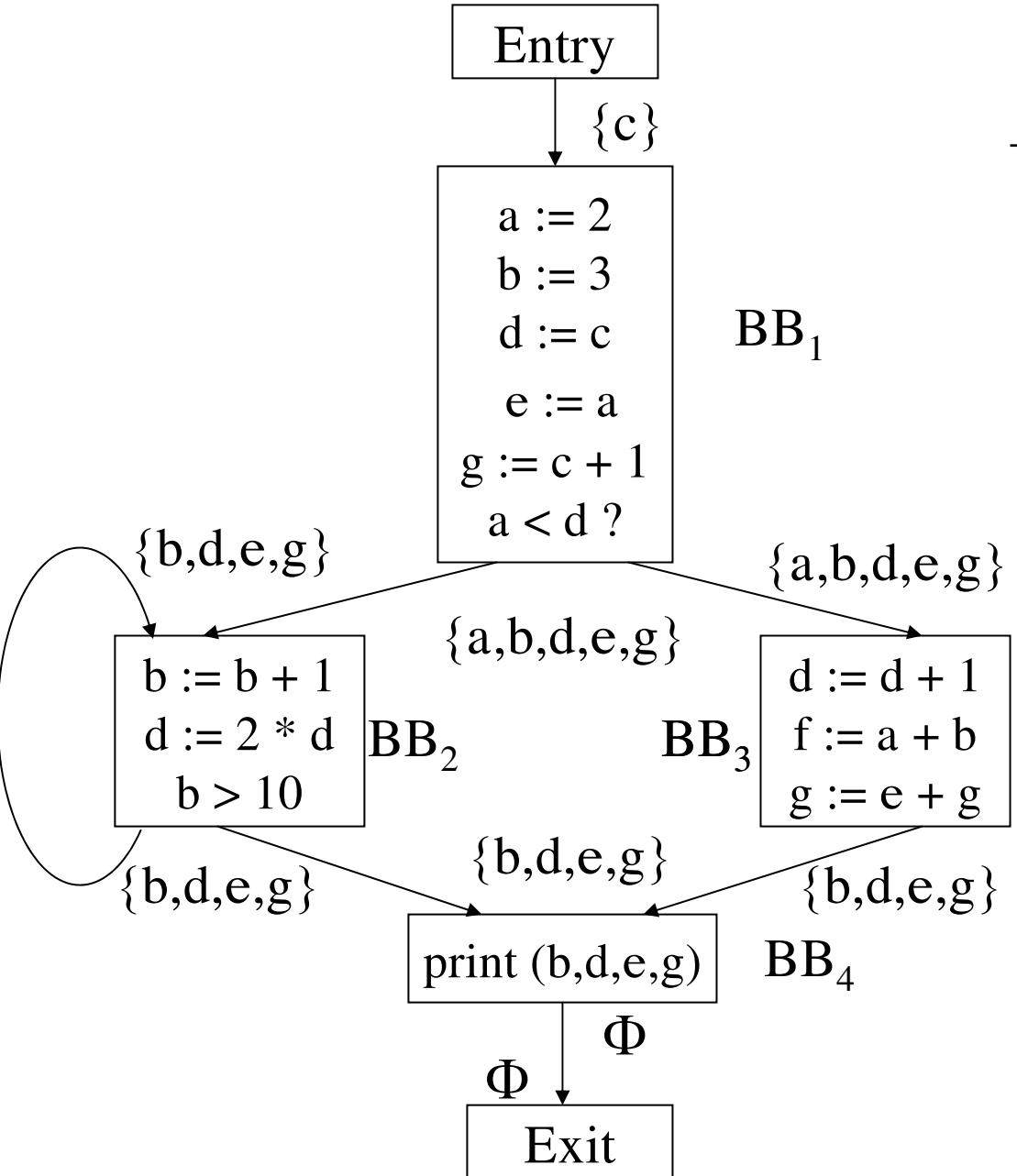
LVin (b) = Use (b) $\cup$ [LVout (b) - Def (b)]

if (LVin (b) $\neq$ oldin) change = true

}

}

Thí dụ: cho dòng điều khiển như sau, tìm tập các biến sống khi ra khỏi các khối cơ bản.



BB	Use (BB)	Def (BB)
1	{c}	{a,b,d,e,g}
2	{b,d}	{b,d}
3	{a,b,d,e,g}	{d,f,g}
4	{b,d,e,g}	Φ

$$LVout(b) = \bigcup_{i \in succ(b)} LVin(i)$$

$$LVin(b) = Use(b) \cup [LVout(b) - Def(b)]$$

10.3.5. Biểu thức có sẵn (Avaible expression)

▪ Một biểu thức x op y được gọi biểu thức có sẵn tại điểm p nếu mọi con đường từ nút khởi đầu đến p hoặc sau lần tính toán trước khi đạt đến p không có tác vụ gán cho x và y.

▪ Ứng dụng của biểu thức có sẵn là để loại bỏ biểu thức con dùng chung.

▪ Ta phải tìm tất cả biểu thức có sẵn tại điểm vào và ra của mỗi khối cơ bản.

10.3.5.1. Tập Eval

Eval (b) là tập các biểu thức có sẵn được thực hiện trong b mà vẫn có sẵn tại điểm ra của b.

10.3.5.2. Tập Kill

Kill (b) là tập các biểu thức bị thay đổi trong b.

10.3.5.3. Sự cân bằng dòng dữ liệu

AEin (b): tập các biểu thức có sẵn tại điểm bắt đầu của b.

AEout (b); tập biểu thức có sẵn chạm đến điểm kết thúc của b.

Công thức:

$$AEin(b) = \bigcap_{i \in pred(b)} AEout(i)$$

$$AEout(b) = Eval(b) \cup [AEin(b) - Kill(b)]$$

Giải thuật: tìm tập các biểu thức có sẵn tại điểm vào và ra của mỗi khối cơ bản.

Nhập: đồ thị dòng G với Eval (b) và Kill (b) được tính toán trước cho khối cơ bản b.

Xuất: AEin (b) cho khối cơ bản b.

Giải thuật:

$\cup$: tập các biểu thức trong đồ thị dòng điều khiển

AEout (Entry) = $\varnothing$

AEout (b) = Eval (b) $\cup$ [U – Kill (b)] $\forall$ b $\in$ N - {Entry, Exit}

change = true

while (change) {

change = false

for each b $\in$ N - {Entry, Exit} {

oldout = AEout (b)

AEin (b) = $\cap$ AEout (i)

i $\in$ pred (b)

10.4. Loại bỏ dư thừa

Quá trình loại bỏ dư thừa bao gồm loại bỏ những biểu thức con chung, lan truyền những bản copy, di chuyển mã không đổi trong vòng lặp ra ngoài vòng lặp.

10.4.1. Loại bỏ biểu thức con chung

Giải thuật: loại bỏ biểu thức con chung

Nhập: mã ba địa chỉ của đồ thị dòng điều khiển với các AEin và AEout cho từng khối cơ bản.

Xuất: đoạn mã ba địa chỉ đã loại bỏ biểu thức con chung.

Giải thuật:

for mỗi khối $b \in N$

for mỗi lệnh $\in$ b có dạng y = x op y

mà x op y là có sẵn tại điểm vào của b {

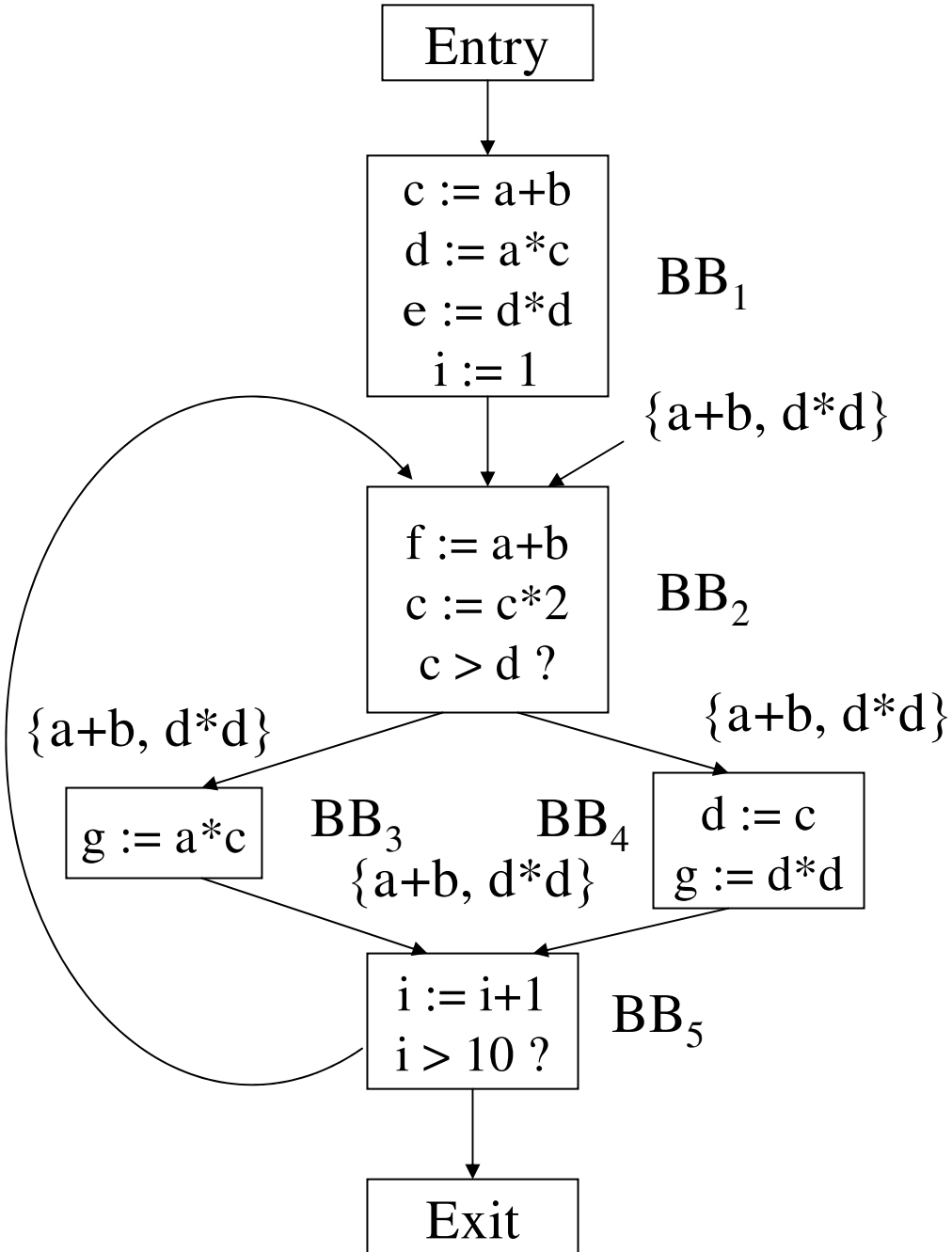
1. Xác định nếu x op y có sẵn tại mỗi câu lệnh
2. Xác định việc tính toán x op y mà đạt đến z
3. Tạo một biến mới t
4. Thay thế sự định nghĩa w = x op y tìm thấy ở bước 2 bằng

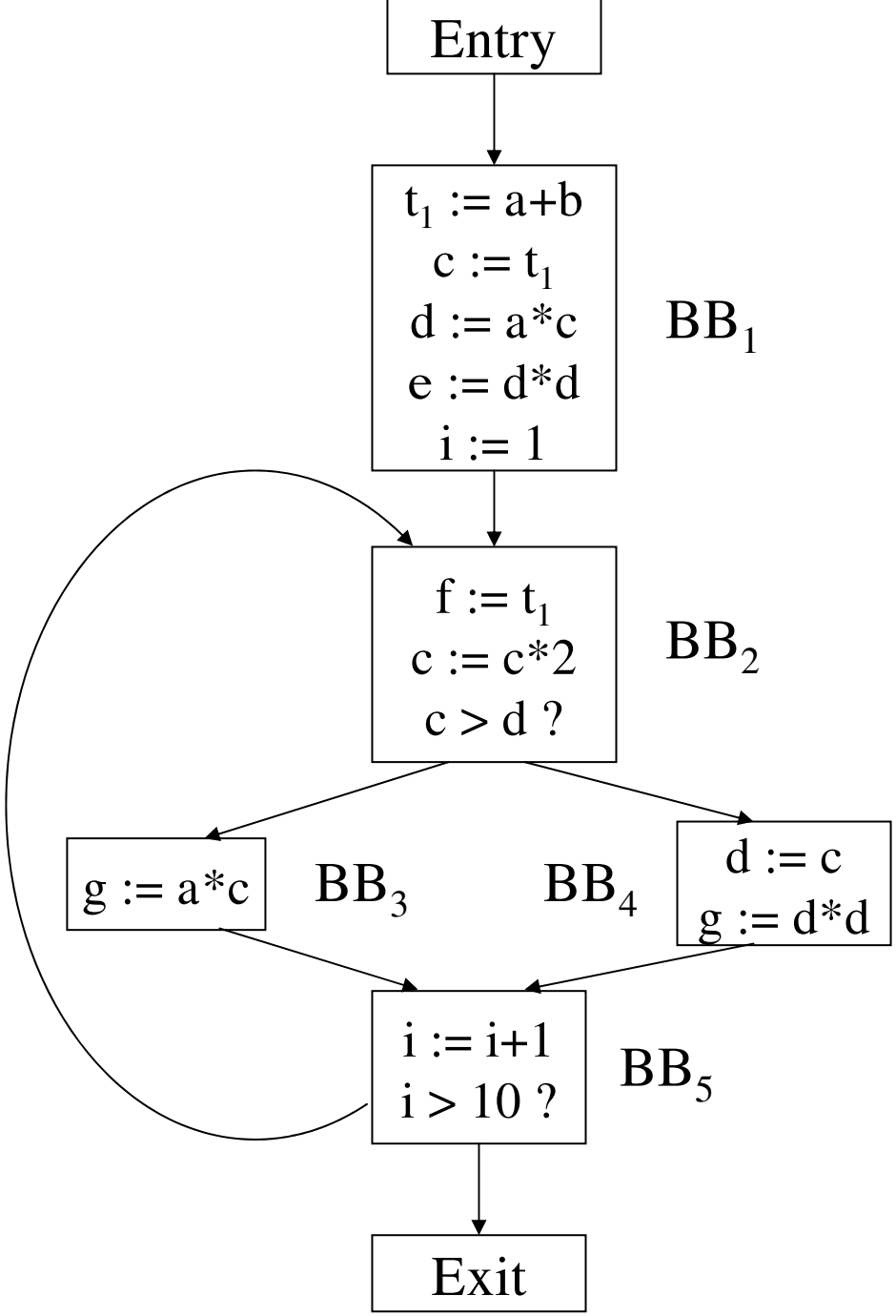
   t = x op y; w = t
5. Thay z = x op y bằng z = t

    }

}

Thí dụ về loại bỏ biểu thức con dùng chung



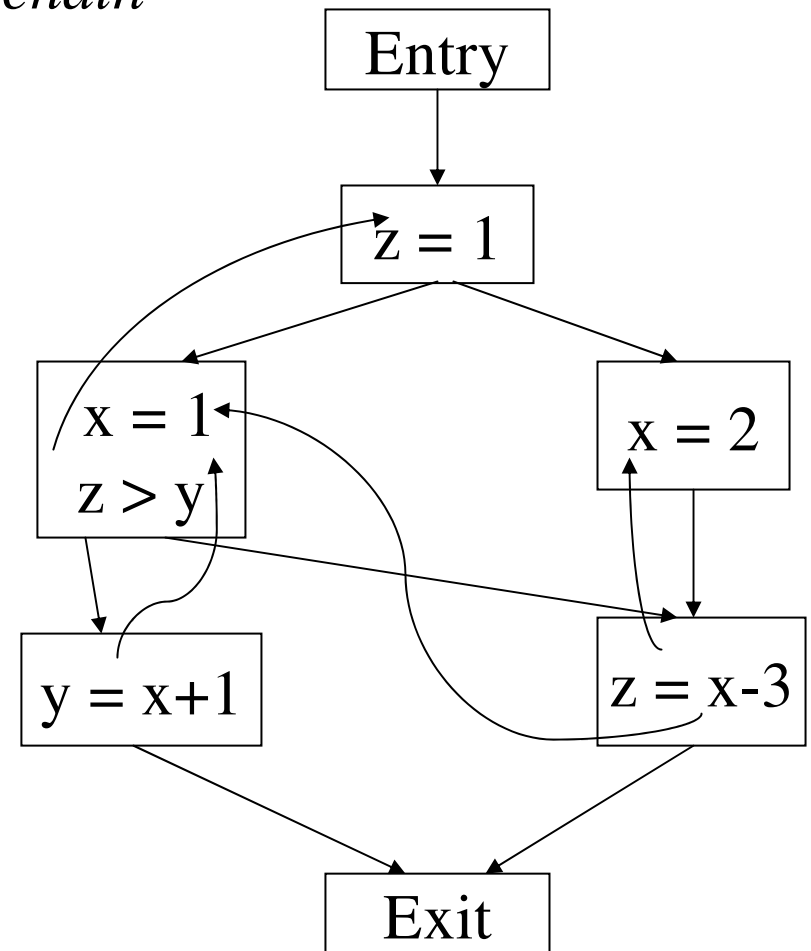

Entry
t1 := a+b
c := t1
d := a*c
e := d*d
i := 1
BB1
f := t1
c := c*2
c > d ?
BB2
g := a*c
BB3
BB4
d := c
g := d*d
i := i+1
i > 10 ?
BB5
Exit

10.4.2. Lan truyền bản copy

10.4.2.1. Định nghĩa sử dụng (use definition)

Tập các định nghĩa đạt đến việc sử dụng của a như là một biến được gọi là dây xích sử dụng định nghĩa (ud - chain) cho biến đó.

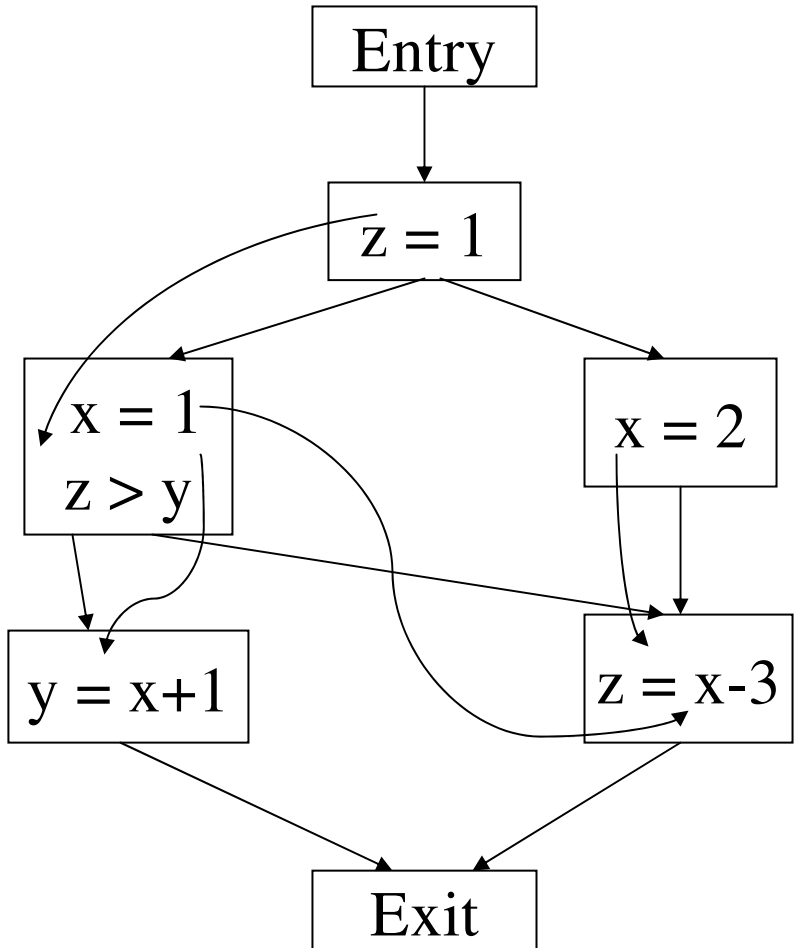
Thí dụ về ud – chain



10.4.2.2. Dây xích định nghĩa sử dụng

Tập tất cả các lần sử dụng mà đạt đến bởi một định nghĩa được gọi là dây xích định nghĩa sử dụng (du – chain).

Thí dụ về du – chain và ud – chain



10.4.2.3. Biểu thức copy có sẵn (Available Copy Expression)

- ***Tập copy***

Tập những câu lệnh copy u := v trong b mà u và v không được gán sau đó trong b, nghĩa là câu lệnh có sẵn tại điểm kết thúc của b.

- ***Tập kill***

Tập các câu lệnh copy bị thay đổi trong b, nghĩa là tập câu lệnh copy trong khối cơ bản khác mà có toán hạng của nó được gán cho b.

- ***Sự cân bằng dòng dữ liệu***

copyin: là tập lệnh copy có sẵn tại điểm vào b

copyout: là tập lệnh copy có sẵn tại điểm kết thúc b

công thức: $\text{copyin}(b) = \bigcap_{i \in \text{pred}(b)} \text{copyout}(i)$

$$\text{copyout}(b) = \text{copy}(b) \cup [\text{copyin}(b) - \text{kill}(b)]$$

Giải thuật: tìm bản copy có sẵn

Nhập: đồ thị dòng điều khiển G với kill (b) và copy (b) được tính sẵn cho mỗi khối cơ bản b.

Xuất: copyin (b) cho mỗi khối cơ bản.

Giải thuật:

U : Tập tất cả các copy trong đồ thị dòng điều khiển

copyout (Entry) = Φ

copyout (b) = copy (b) $\cup$ [U - kill (b)] $\forall$ b $\in$ N - {Entry, Exit}

changed = true

while (changed) {

changed = false

for each b $\in$ N - {Entry, Exit} {

oldout = copyout (b)

copyin (b) = $\cap$ copyout (i)

$i \in pred(b)$

copyout (b) = copy (b) $\cup$ [copyin (b) – kill (b)]

if (copyout (b) $\neq$ oldout) changed = true

}

}

Aein (Exit) = $\cap$ Aeout (i)

$i \in pred(Exit)$

10.4.2.4. Lan truyền bản copy

Câu lệnh copy là câu lệnh có dạng x = y.

Sự lan truyền bản copy là thay thế x bằng y mà không làm thay đổi trị của x hoặc y.

Giải thuật: lan truyền bản copy

Nhập: đồ thị dòng điều khiển G với du – chain.

Xuất: đồ thị dòng điều khiển có sử dụng lan truyền bản copy.

Giải thuật:

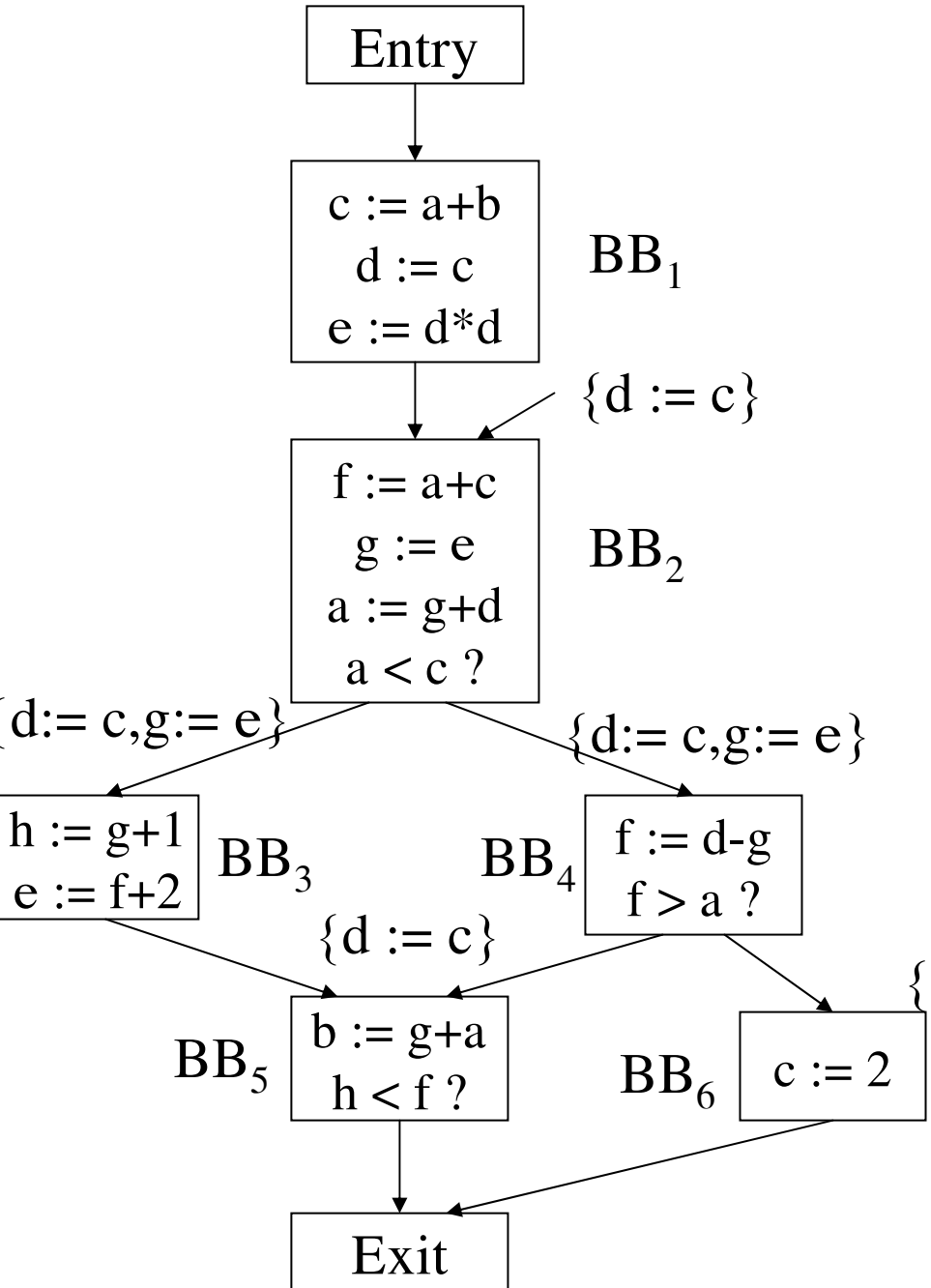
for mỗi bản copy s: x := y thực hiện các bước sau:

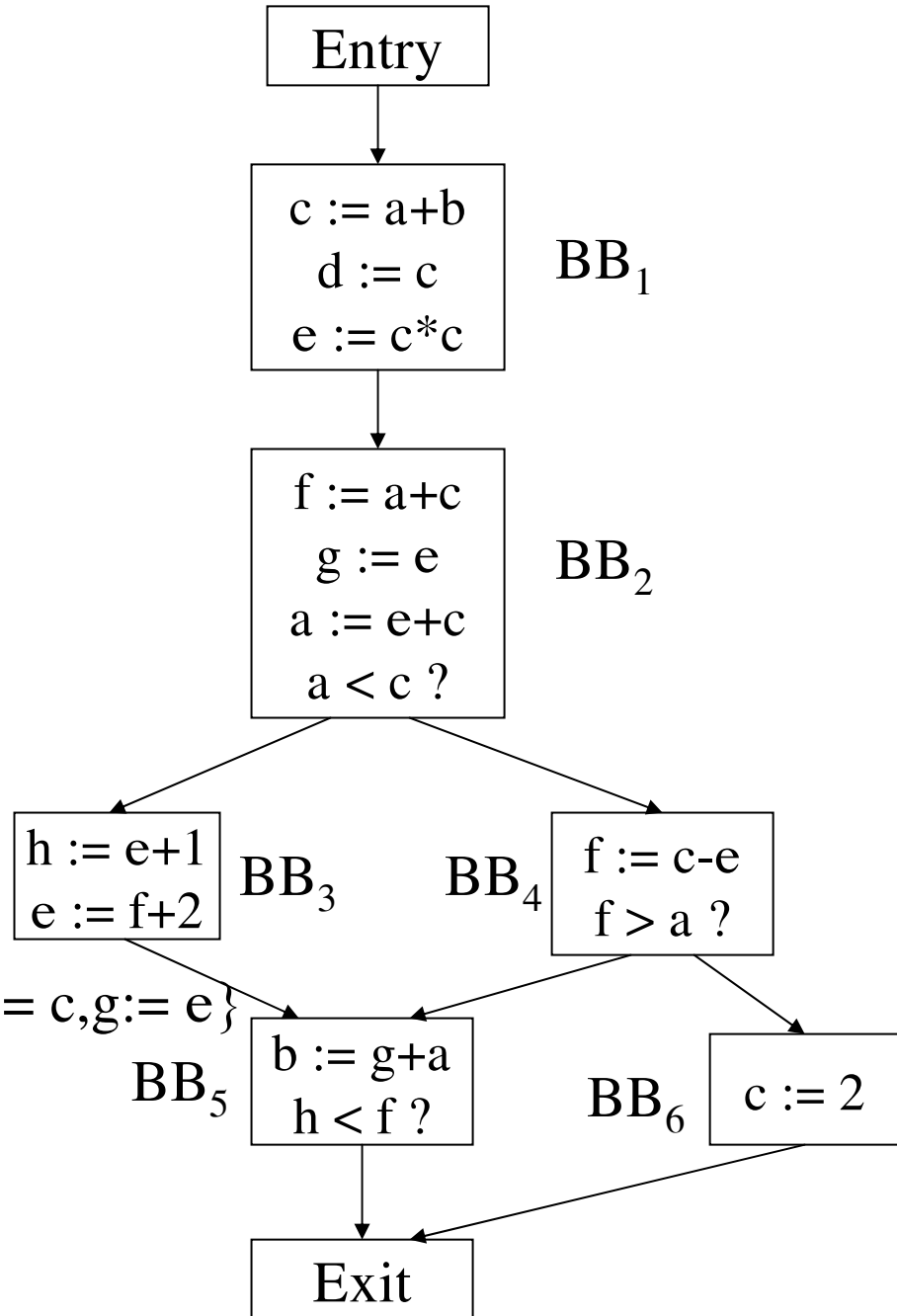
1. Xác định tất cả các nơi mà sự định nghĩa của x được sử dụng.
2. for mỗi lần sử dụng u:

   a. s phải có một sự định nghĩa của x chạm đạt đến u và

   b. mọi con đường từ s đến u, không có tác vụ gán đến y.

Thí dụ về sự lan truyền bản copy





10.4.3. Di chuyển mã không đổi của vòng lặp (loop – invariant code motion)

Một sự tính toán trong vòng lặp được gọi là loop – invariant nếu sự tính toán của nó luôn luôn tạo ra cùng một giá trị.

Di chuyển code không đổi là di chuyển các loop – invariant ra bên ngoài vòng lặp.

10.4.3.1. Loop – Invariant

Một tác vụ trong vòng lặp là loop – invariant nếu mỗi toán hạng trong tác vụ là:

- hằng số hoặc

- tất cả các định nghĩa của toán hạng đều ở bên ngoài vòng lặp hoặc

- chỉ duy nhất có một định nghĩa trong vòng lặp cho toán hạng mà sự định nghĩa là loop – invariant.

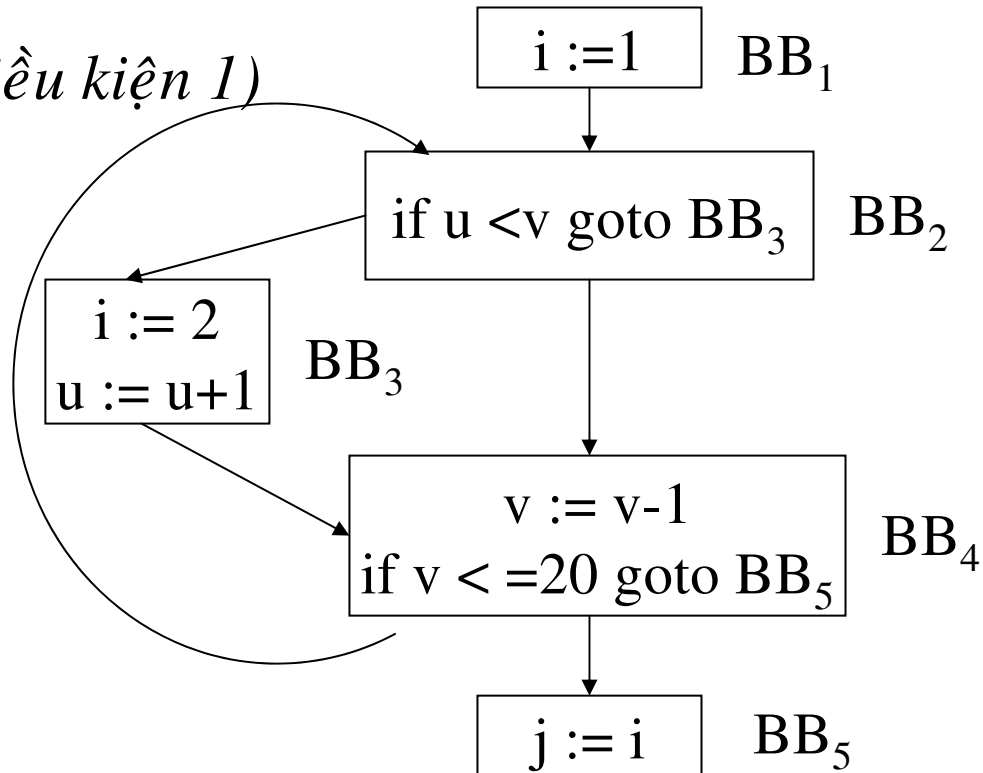
10.4.3.2. Thực hiện di chuyển code

Sự di chuyển code phải thỏa mãn 3 điều kiện ví dụ cho phát biểu

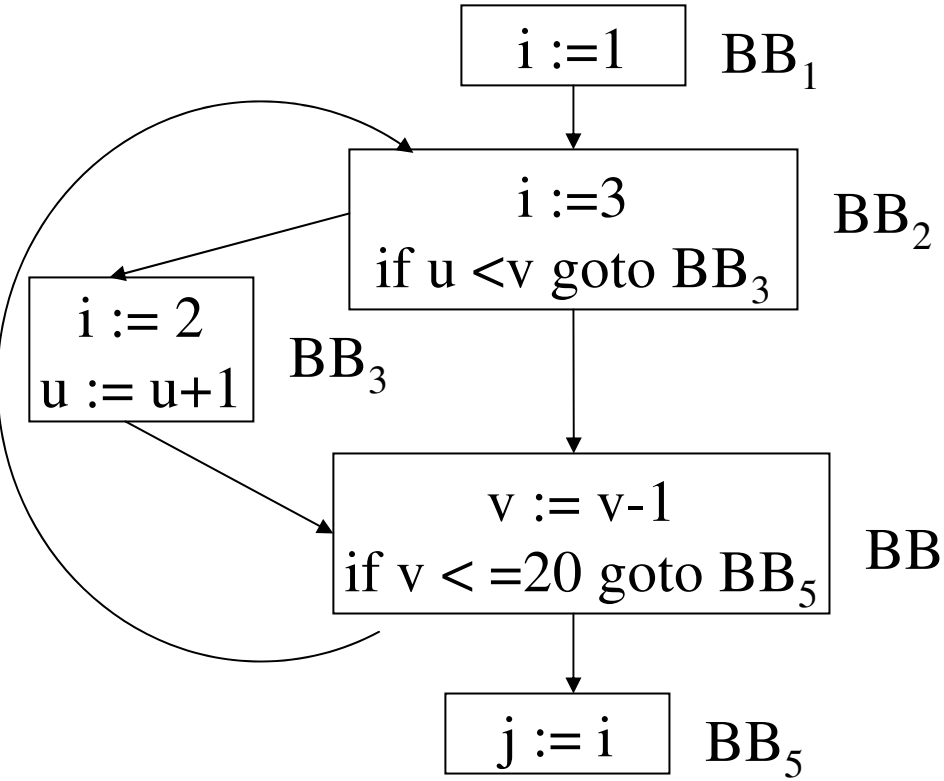
s : x = y + z.

1. Khối cơ bản chứa s phải dominate tất cả các lối ra của vòng lặp.
2. Không có phát biểu nào khác gán cho x.
3. Tất cả các lần sử dụng x chỉ đạt đến sự định nghĩa x trong s.

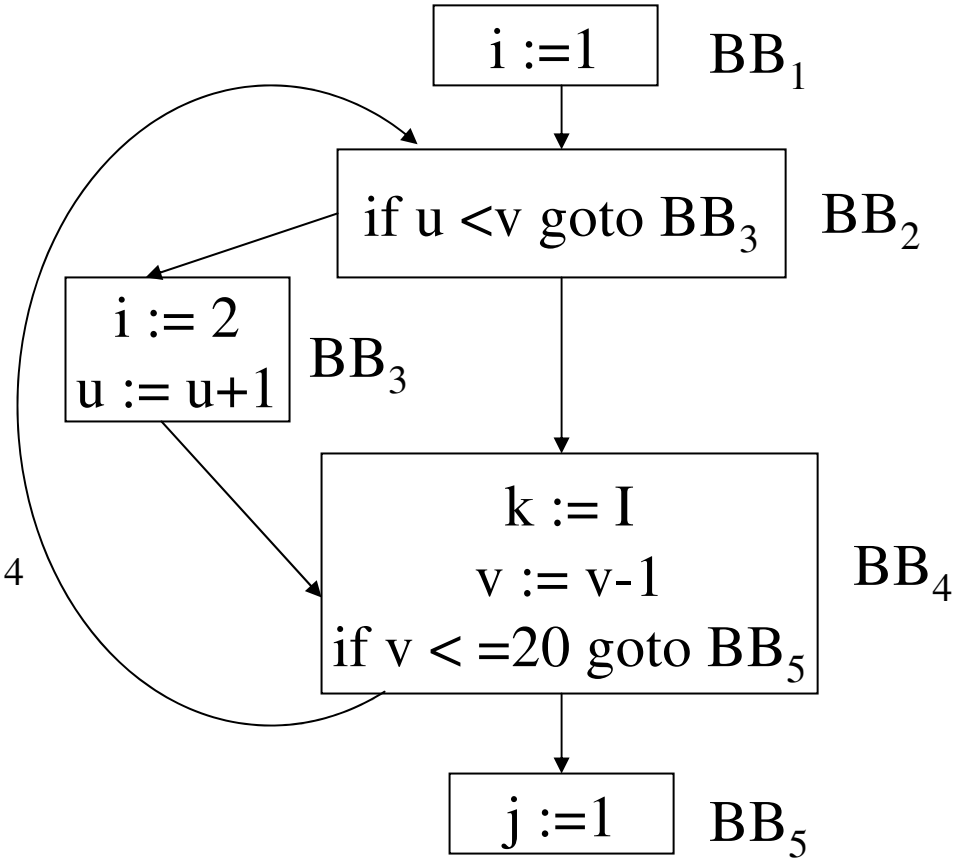
Thí dụ (điều kiện 1)



Thí dụ (điều kiện 2)



Thí dụ (điều kiện 3)



10.5. Tối ưu vòng lặp

Trong phần này chúng ta sẽ trình bày giải thuật tối ưu vòng lặp là strength reduction. Mục đích của giải thuật này là thay thế các câu lệnh đắt tiền bằng câu lệnh rẻ tiền hơn.

10.5.1. Biến thay đổi (Induction variable)

Biến thay đổi trong vòng lặp L là x nếu mỗi lần thay đổi nó tăng hoặc giảm một hằng số nhất định.

10.5.2. Biến thay đổi cơ bản (Basic Induction Variable – BIV)

Biến v được gọi là BIV trong L nếu nó có dạng

$v := v \pm c$ với c là hằng số

10.5.3. Biến thay đổi dẫn xuất (Derived Induction Variable – DIV)

Biến j được gọi là biến thay đổi dẫn xuất nếu nó có một trong các dạng sau xuất hiện trong vòng lặp:

j := a * i hoặc j := i * a

j := b + i hoặc j := a * i

j := b – i hoặc j := i – a

j := i / a

trong đó i là biến thay đổi cơ bản (BIV)

Giải thuật 6.1. Tìm biến thay đổi trong vòng lặp L.

Nhập: vòng lặp L.

Xuất: tất cả các BIV và DIV trong L.

Giải thuật:

1. Tìm tất cả BIV trong L.
2. Tìm các DIV.
3. Lặp lại bước (2) cho đến khi nào không tìm thấy DIV khác.

10.5.4. Giải thuật strength Reduction

Nhập: vòng lặp L và tập họ các biến thay đổi.

Xuất: đoạn mã đã thực hiện thay thế các câu lệnh phức tạp bằng những câu lệnh ít phức tạp hơn từ vòng lặp L.

Giải thuật:

```
for mỗi biến BIV i
{
    for mỗi biến j trong họ i → (I, a, b)
    {
        tạo ra một biến mới sj
        khởi động sj := a * i + b và đặt vào preheader của L
        thay thế câu lệnh gán j bằng j := sj
        sau mỗi phát biểu i := i ± c trong L
```

{

chèn thêm sj := sj ± c * a

}

}

thêm sj vào họ của i