# KIỂM THỬ VÀ ĐẢM BẢO CHẤT LƯỢNG PHẦN MỀM

**INT3117**

**Bài giảng 05: White-box testing (II)**

# Hôm trước…

- Các kỹ thuật sinh test white-box (white-box testing)
  - Dòng điều khiển (Control-flow)
    - Phủ dòng lệnh (Statement coverage)
    - Phủ nhánh (Branch coverage)
    - Phủ đường (Path coverage)

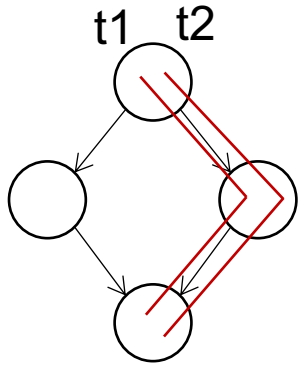
# Hôm nay…

- Các kỹ thuật sinh test white-box (white-box testing)
  - Dòng dữ liệu (Data-flow)
    - All-Defs
    - All-Uses
    - All-DU-Path
    - All-P-Uses/Some-C-Uses
    - All-C-Uses/Some-P-Uses
    - All-P-Uses
    - All-C-Uses

# Có độ phủ CFG rồi tại sao lại cần thêm?

- Ý tưởng chính:
  - Sinh tests dựa trên độ phủ CFG chỉ xem xét các đường thực thi (**cấu trúc**)
  - Trên các đường đi của chương trình, biến nào được định nghĩa (defined) sau đó dùng (used) cũng nên được xét tới
- Từ ý tưởng trên, một "hội" các tiêu trí dựa trên dòng dữ liệu (dataflow) nhằm cung cấp các cấp độ phủ dữ liệu khác nhau

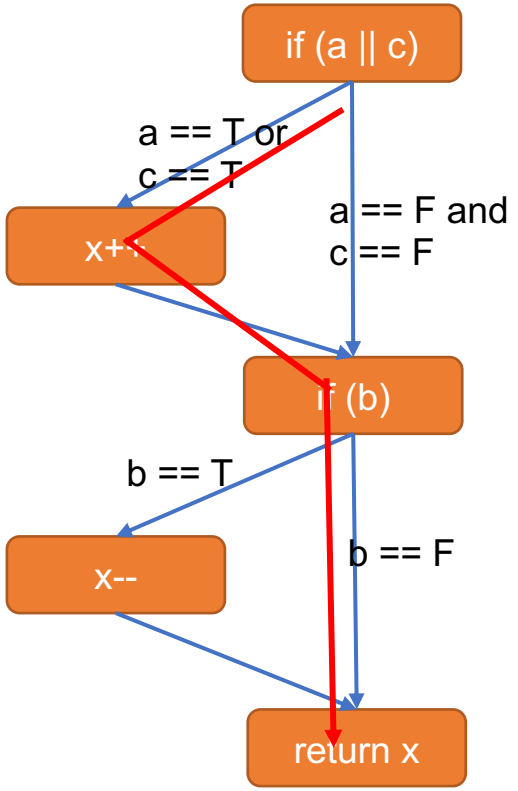
t1 và t2 có phủ cùng một thứ không?



# Có độ phủ CFG rồi tại sao lại cần thêm?

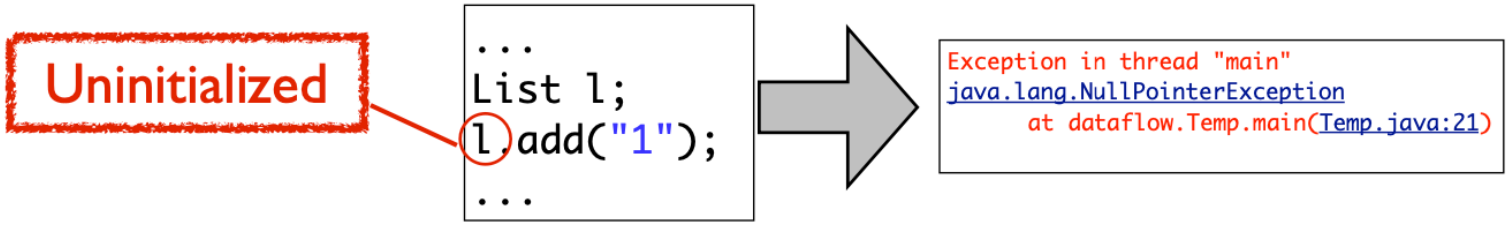
```
public int testMe(int x, boolean a,

boolean b, boolean c){
    if(a || c)
        x++;
    if(b)
        x--;
    return x;
}
```



# Phủ dòng dữ liệu/Dataflow coverage

- Xét tới cách dữ liệu được truy cập và sửa đổi trong hệ thống và cách dữ liệu bị xử lý sai
- Các lỗi phổ biến liên quan tới truy cập:
  - Sử dung biến chưa được khai báo hoặc chưa được khởi tạo
  - Giải phóng bộ hoặc khởi tạo lại biến trước khi biến được tạo hoặc khởi tạo
  - Xoá đối tượng collection (List/Set) và để các thành phần không thể truy cập được



# ĐỊNH NGHĨA biến

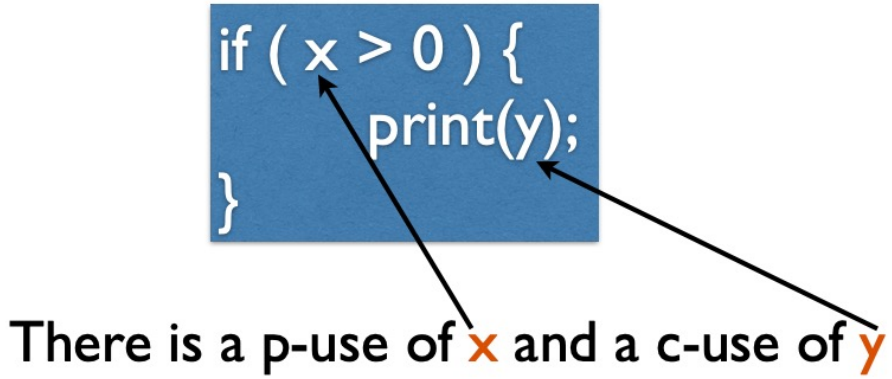
- Một biến (variable) được ĐỊNH NGHĨA (DEFINED) bất khì khi nào giá trị của biến đó bị thay đổi:
  - Nằm ở vế trái của một phép gán: x = 10
  - Nằm trong một lệnh đầu vào: input(y)
  - Được dùng như đối số tham chiếu: update(x, &y)

# DÙNG biến

- Một biến (variable) được DÙNG (USED) bất khì khi nào giá trị của biến đó đọc:
  - Nằm ở vế phải của một phép gán: x = **y**+1
  - Là đối số của một lời gọi hàm bình thường: read(**x**)
  - Nằm trong điều kiện của các câu lệnh rẽ nhánh: if (**x** > 0) {….}

# DÙNG biến: p-use và c-use

- Việc dùng của một biến trong câu lệnh điều kiện được gọi là **p**redicate-use hay p-use
- Bất kỳ việc dùng nào khác được gọi là **c**omputation-use hay c-use



# DÙNG biến

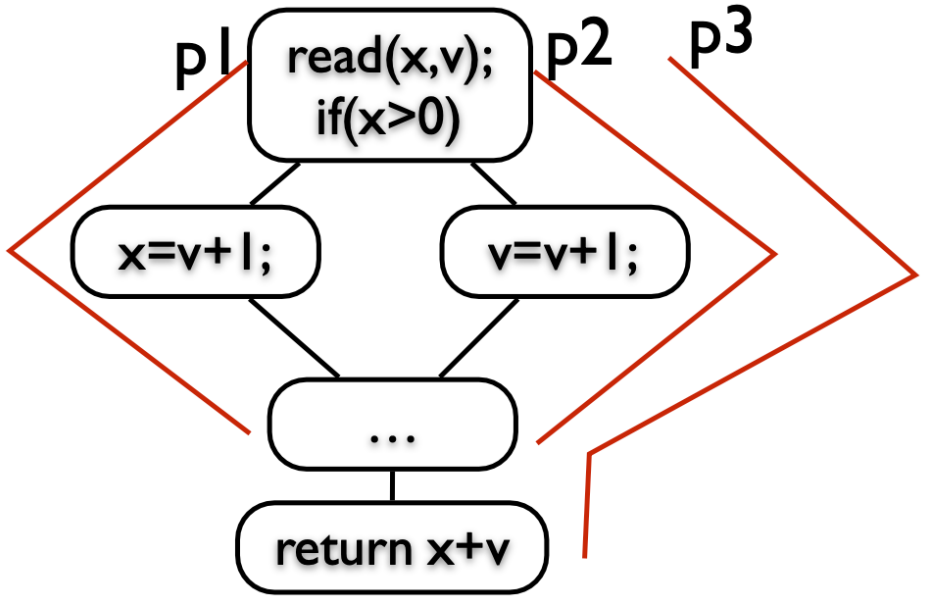
- Một biến (variable) có thể vừa được dùng vừa được định nghĩa (định nghĩa lại trong duy nhất một câu lệnh:
  - Nằm ở cả 2 bên của câu lệnh gán: **x** = **x** + 1
  - Gọi thông qua tham chiếu: increment(&**x**)

# Các thuật ngữ và định nghĩa khác

Với một đồ thị CFG:

- Một đường (path) là **def-clear (definition clear)** với một biến **v** nếu không có bất kỳ phép gán lại giá trị của **v** trên đường đi
- Một **đường hoàn chỉnh (complete path)** là một đường đi từ node entry tới một node exit

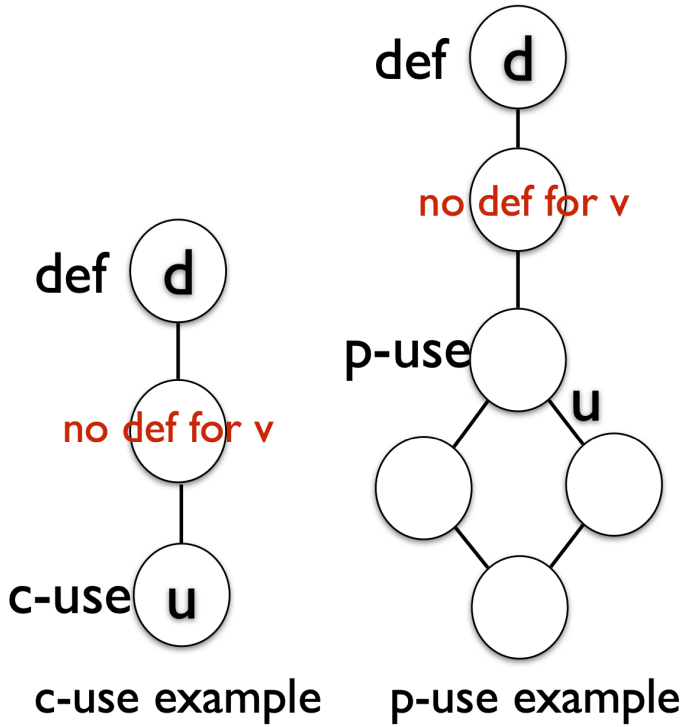
Def-clear path?
Complete path?




Control-flow graph

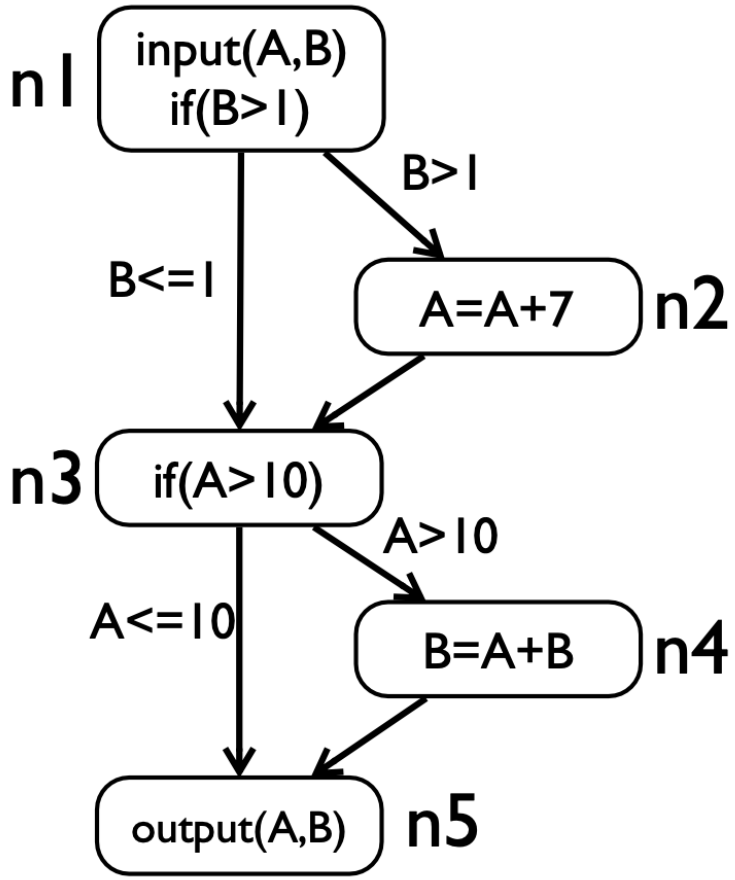
# Definition-use pair (DU pair)

- Definition-use pair (**DU pair**) của một biến **v** là một cặp (d, u):
  - **d** là node định nghĩa **v**
  - Với c-use của **v,** thì **u** là node sử dụng **v.**
  - Với p-use của **v**, thì **u** là cạnh đi ra của dòng lệnh điều kiện dùng **v**
  - Tồn tại một def-clear path của **v** giữa **d** và **u**



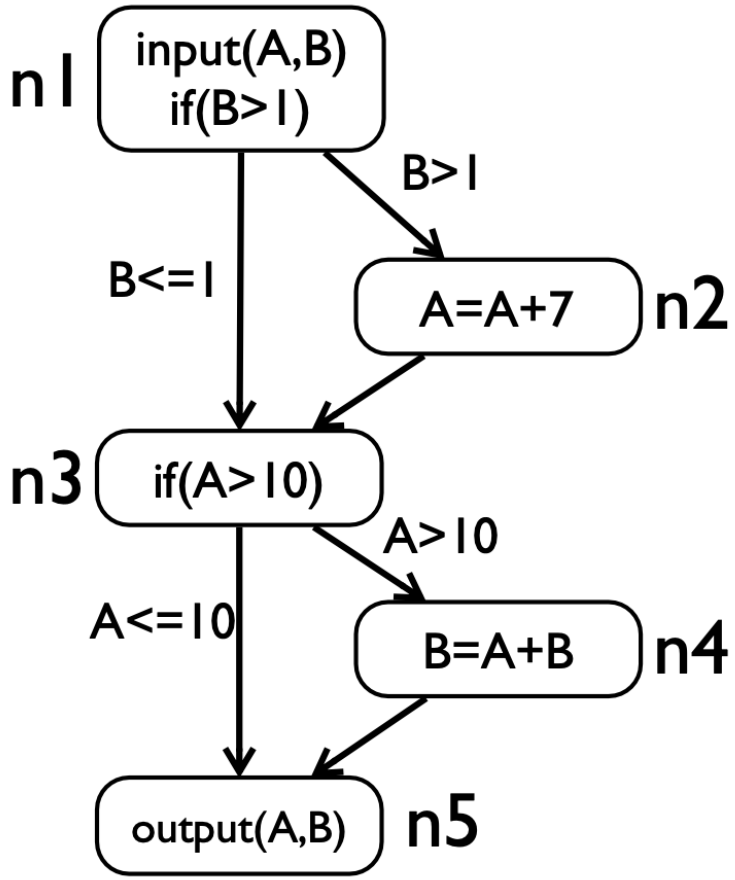
# DU pair: Ví dụ

```
1. input(A,B)
    if (B>1) {
2.     A = A+7
    }
3. if (A>10) {
4.     B = A+B
    }
5. output(A,B)
```



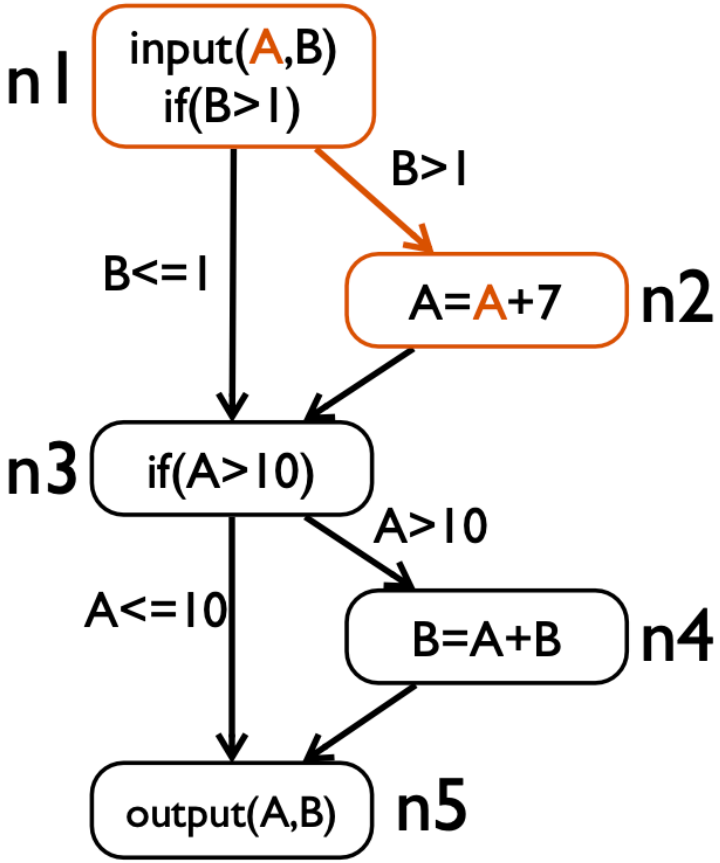
# DU pair: Xác định DU-pairs của biến **A**

| du-pair | path(s) |
| --- | --- |
| ??? | |



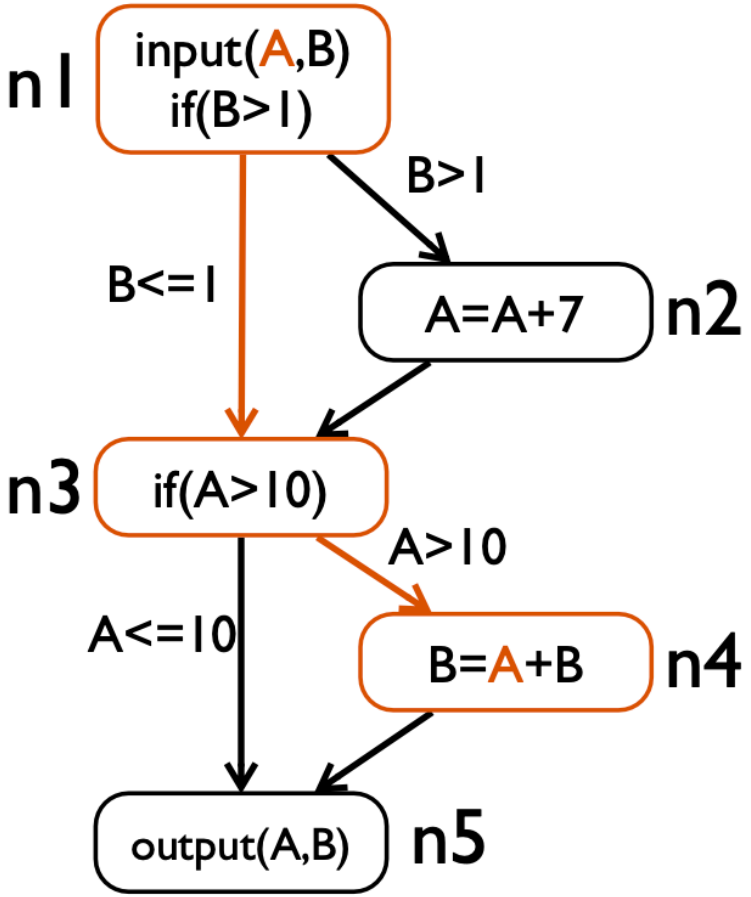
# DU pair: Xác định DU-pairs của biến **A**

| du-pair | path(s) |
| --- | --- |
| (1,2) | <1,2> |



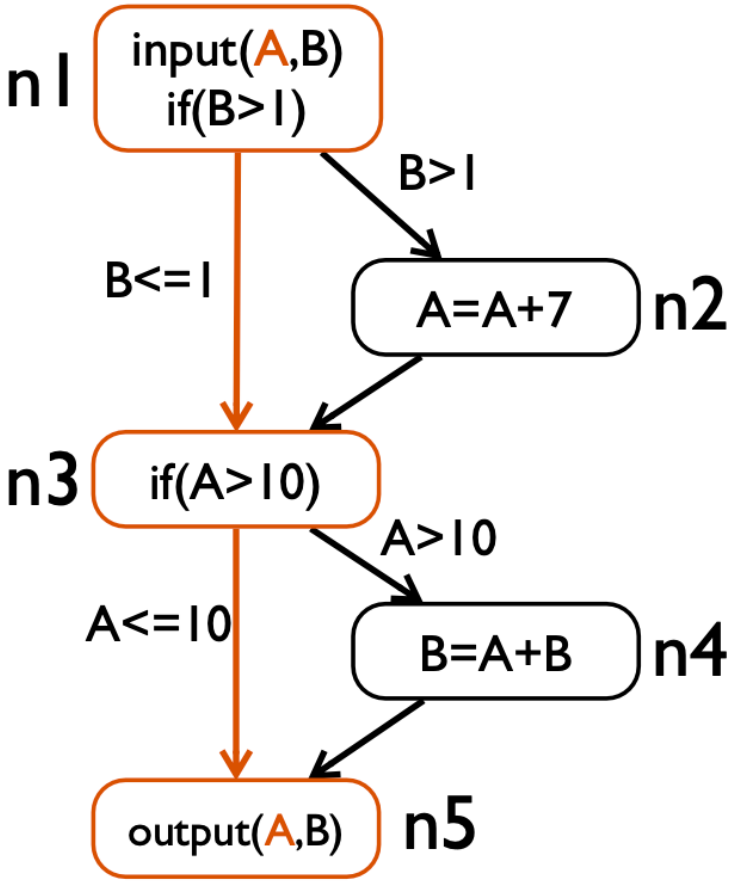
# DU pair: Xác định DU-pairs của biến **A**

| du-pair | path(s) |
|---|---|
| (1,2) | <1,2> |
| **(1,4)** | **<1,3,4>** |



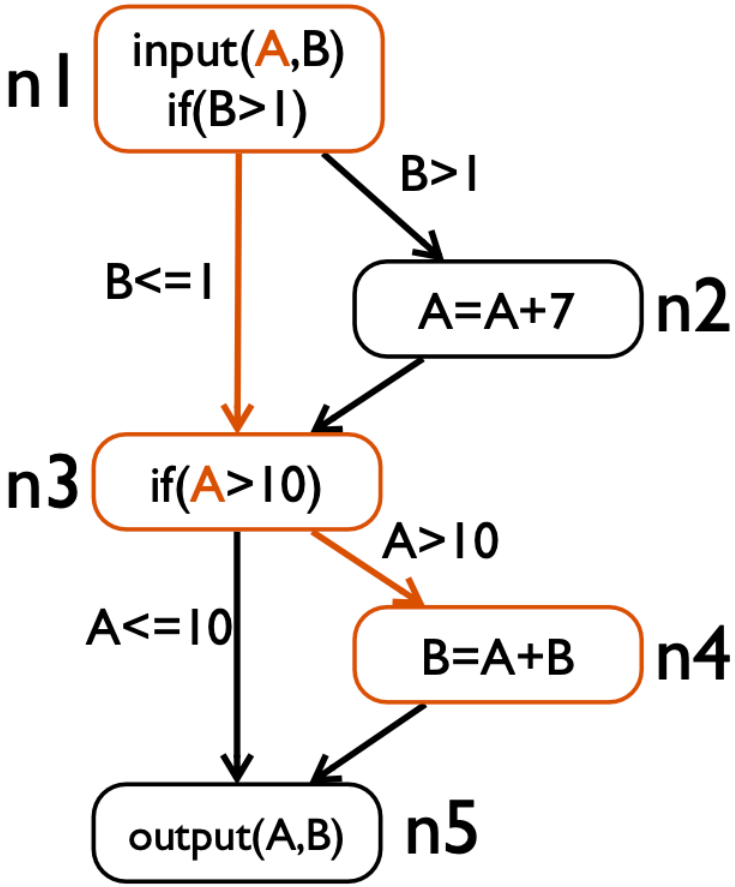
# DU pair: Xác định DU-pairs của biến **A**

| du-pair | path(s) |
|---|---|
| (1,2) | <1,2> |
| (1,4) | <1,3,4> |
| **(1,5)** | <1,3,4,5> |
| | **<1,3,5>** |



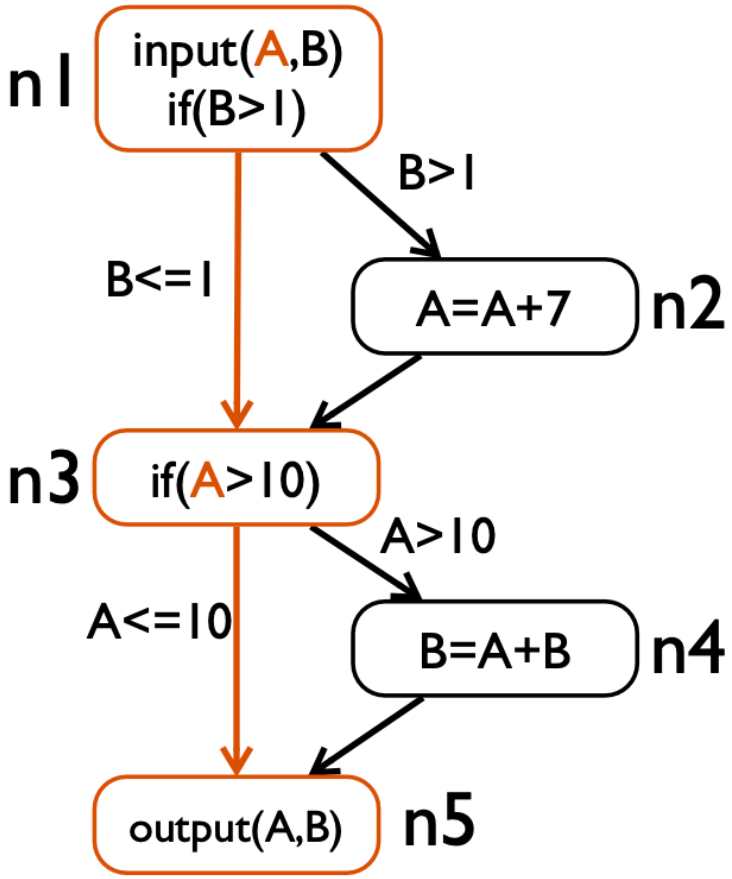
# DU pair: Xác định DU-pairs của biến **A**

| du-pair | path(s) |
|---|---|
| (1,2) | <1,2> |
| (1,4) | <1,3,4> |
| (1,5) | <1,3,4,5> |
| | <1,3,5> |
| (1,<3,4>) | <1,3,4> |



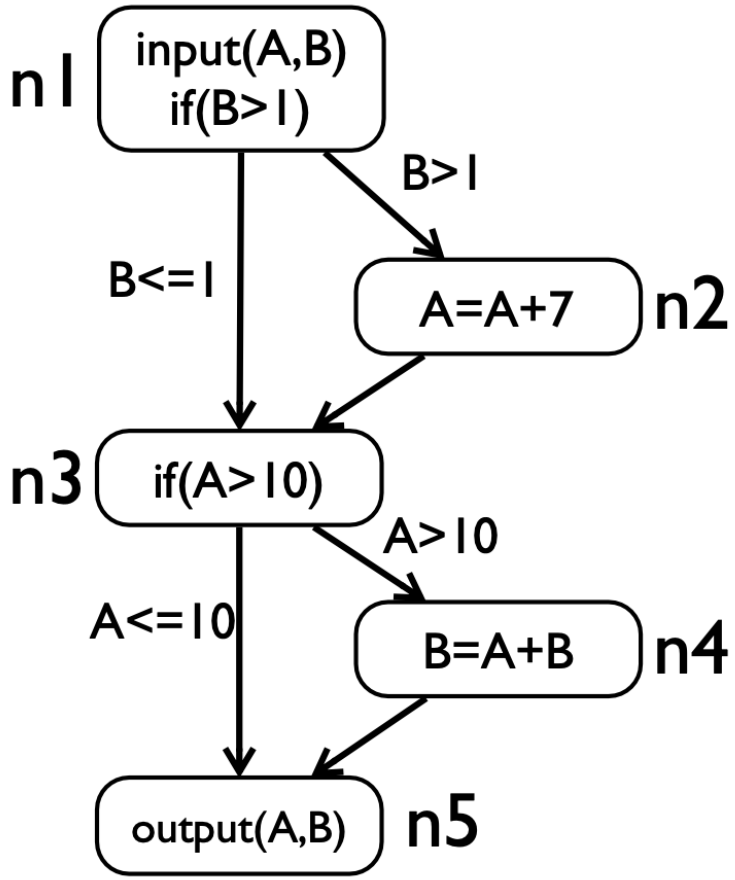
# DU pair: Xác định DU-pairs của biến **A**

| du-pair | path(s) |
|---|---|
| (1,2) | <1,2> |
| (1,4) | <1,3,4> |
| (1,5) | <1,3,4,5> |
| | <1,3,5> |
| (1,<3,4>) | <1,3,4> |
| **(1,<3,5>)** | **<1,3,5>** |



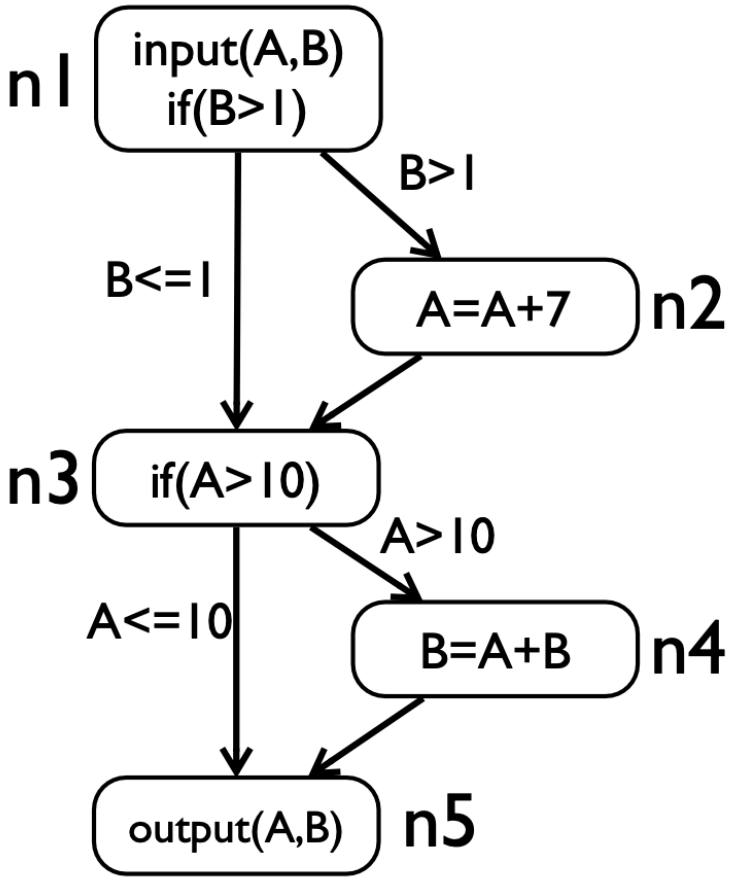
# DU pair: Xác định DU-pairs của biến **A**

| du-pair | path(s) |
|---|---|
| (1,2) | <1,2> |
| (1,4) | <1,3,4> |
| (1,5) | <1,3,4,5> |
| | <1,3,5> |
| (1,<3,4>) | <1,3,4> |
| (1,<3,5>) | <1,3,5> |
| (2,4) | <2,3,4> |
| (2,5) | <2,3,4,5> |
| | <2,3,5> |
| (2,<3,4>) | <2,3,4> |
| (2,<3,5>) | <2,3,5> |



# Bài tập 1: Xác định DU-pairs của biến **B**

| du-pair | path(s) |
| --- | --- |
| ??? | |

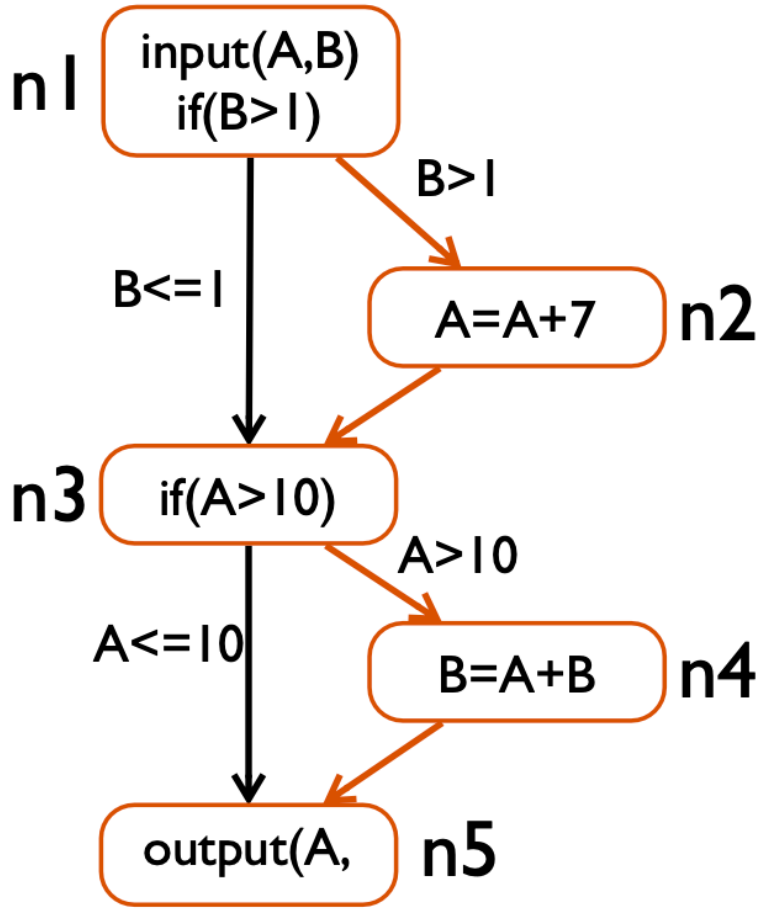


# Data flow coverage

- All-defs
  - Với mỗi biến **v**, ít nhất một đường def-clear từ mỗi **def** của **v** tới ít nhất một c-use hoặc p-use của **v** phải được phủ

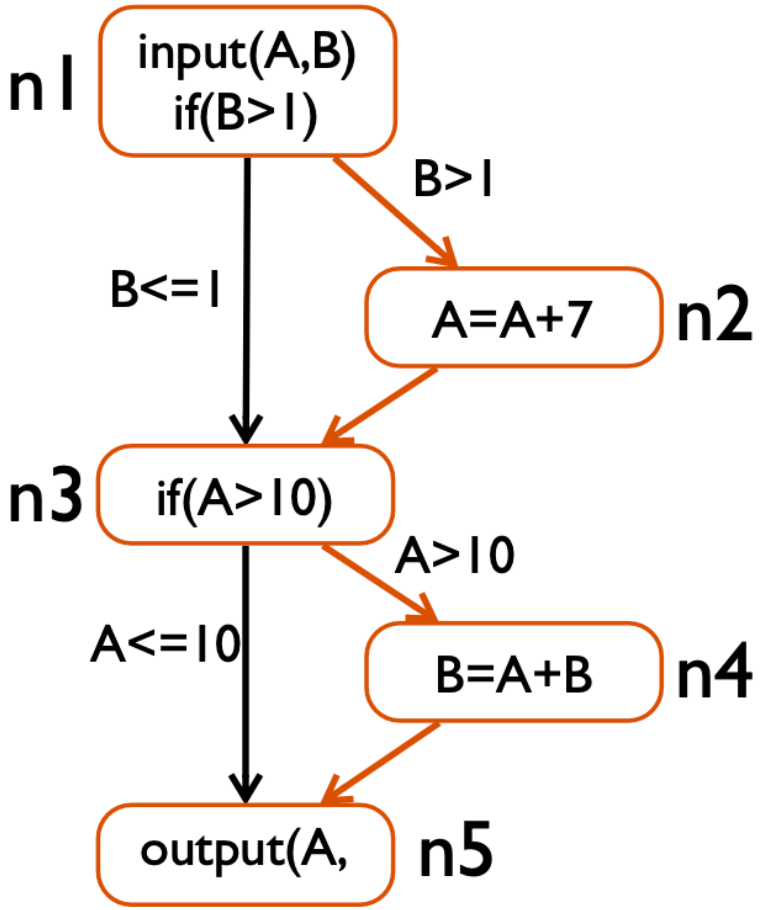
# Đường **<1,2,3,4,5>** phủ những đường def-clear nào của **A**?

| du-pair | path(s) | |
|---|---|---|
| (1,2) | <1,2> | ✓ |
| (1,4) | <1,3,4> | |
| (1,5) | <1,3,4,5> | |
| | <1,3,5> | |
| (1,<3,4>) | <1,3,4> | |
| (1,<3,5>) | <1,3,5> | |
| (2,4) | <2,3,4> | ✓ |
| (2,5) | <2,3,4,5> | ✓ |
| | <2,3,5> | |
| (2,<3,4>) | <2,3,4> | ✓ |
| (2,<3,5>) | <2,3,5> | |



# Đường **<1,2,3,4,5>** phủ những đường def-clear nào của **B**?

| du-pair | path(s) |
|---|---|

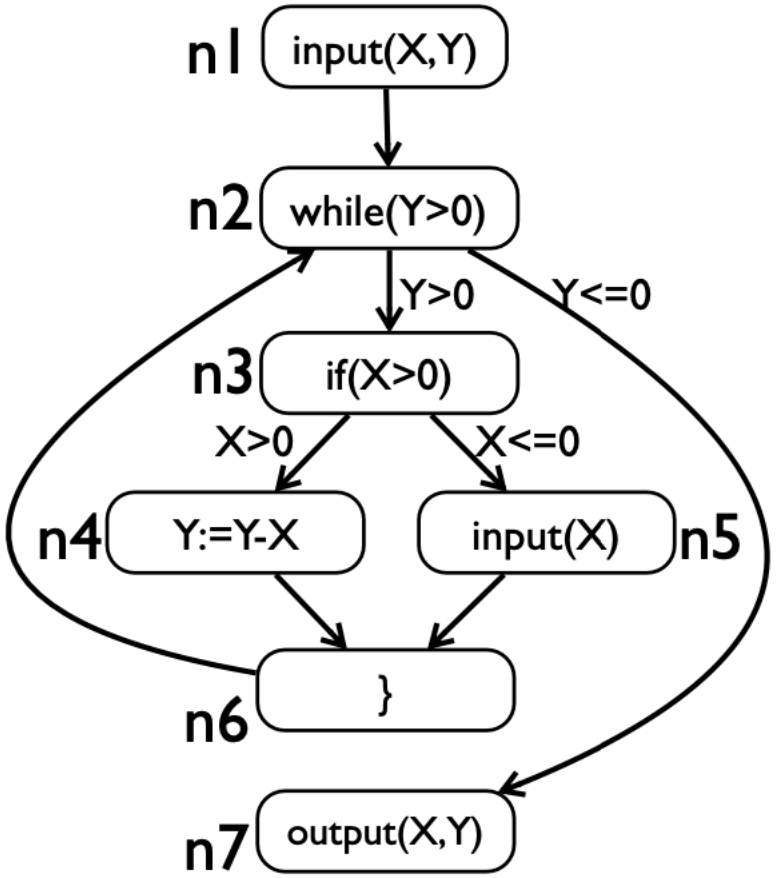


# Data flow coverage

- All-defs
  - Với mỗi biến **v**, ít nhất một đường def-clear từ mỗi **def** của **v** tới ít nhất một c-use hoặc p-use của **v** phải được phủ
- All-uses
  - Với mỗi biến **v**, ít nhất một đường def-clear từ mỗi **def** của **v** tới mỗi c-use và mỗi p-use của **v** phải được phủ
  - Với p-use, bao gồm tất cả cạnh đi ra của dòng lệnh điều kiện
  - Yêu cầu là tất cả DU-pairs được phủ

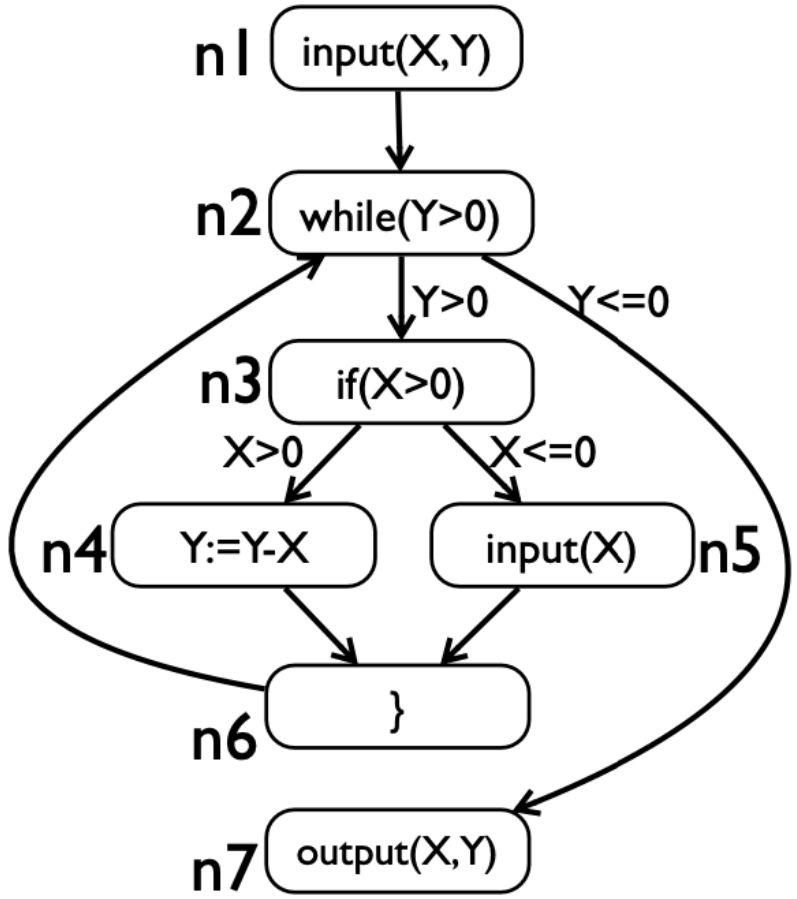
# DU-pair: Ví dụ phức tạp hơn

```
1. input(X,Y)
2. while (Y>0) {
3.     if (X>0)
4.         Y := Y-X
       else
5.         input(X)
6.     }
7. output(X,Y)
```



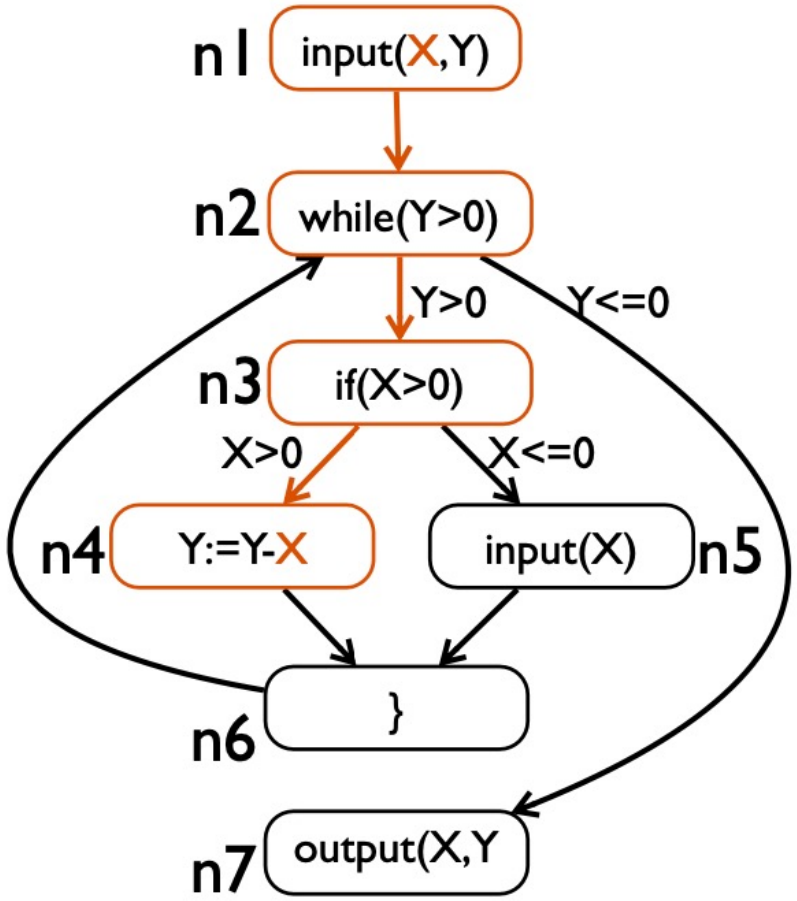
# Tất cả DU-pairs – Biến X

| du-pair | path(s) |
|---|---|
| (1,4) | <1,2,3,4> |
| | <1,2,3,4,(6,2,3,4)*> |
| (1,7) | |
| | |
| (1,<3,4>) | |
| | |
| (1,<3,5>) | |
| | |
| (5,4) | |
| | |
| (5,7) | |
| | |
| (5,<3,4>) | |
| | |
| (5,<3,5>) | |
| | |



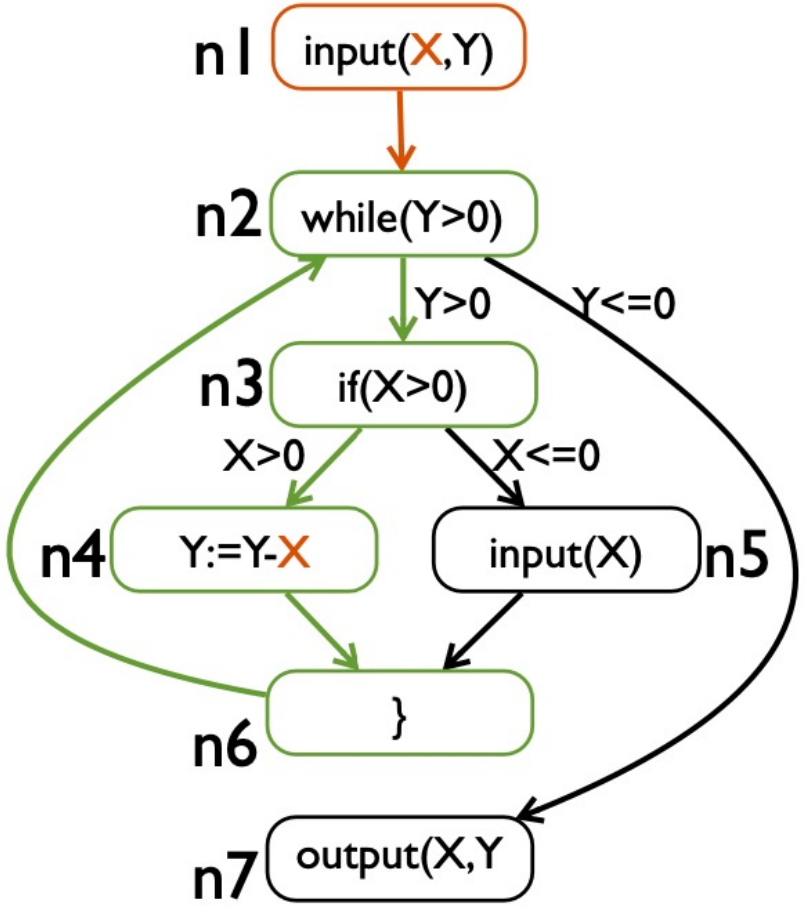
# Tất cả DU-pairs – Biến X

| du-pair | path(s) |
|---|---|
| (1,4) | <1,2,3,4> |
| | <1,2,3,4,(6,2,3,4)*> |
| (1,7) | |
| | |
| (1,<3,4>) | |
| | |
| (1,<3,5>) | |
| | |
| (5,4) | |
| | |
| (5,7) | |
| | |
| (5,<3,4>) | |
| | |
| (5,<3,5>) | |
| | |



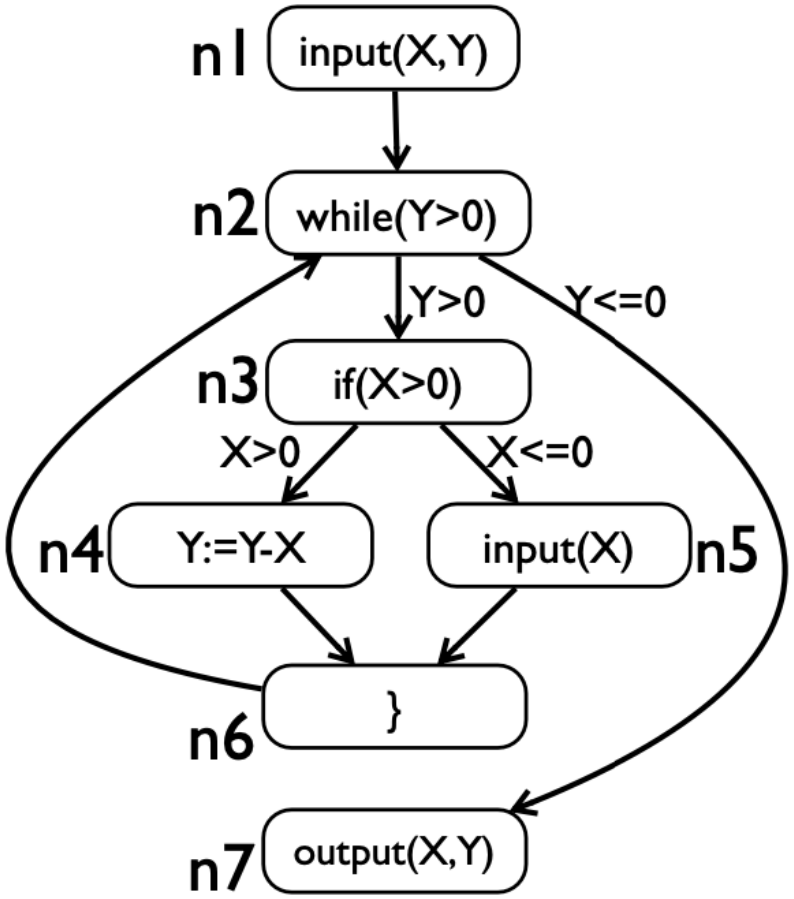
# Tất cả DU-pairs – Biến X

| du-pair | path(s) |
|---|---|
| (1,4) | <1,2,3,4> |
| | <1,2,3,4,(6,2,3,4)*> |
| (1,7) | |
| | |
| (1,<3,4>) | |
| | |
| (1,<3,5>) | |
| | |
| (5,4) | |
| | |
| (5,7) | |
| | |
| (5,<3,4>) | |
| | |
| (5,<3,5>) | |
| | |



# Tất cả DU-pairs – Biến X

| du-pair |
| --- |
| (1,4) |
| (1,7) |
| (1,<3,4>) |
| (1,<3,5>) |
| (5,4) |
| (5,7) |
| (5,<3,4>) |
| (5,<3,5>) |



# Một vài khái niệm nữa…

- Một đường là đường **đơn giản (simple)**, nếu tất cả các cạnh trong đường đó là độc nhất (tất cả các cạnh là khác nhau)
- Một đường là **không vòng lặp (loop-free),** nếu tất cả các đỉnh là độc nhất (tất cả các đỉnh là khác nhau)

# Đường đơn giản (simple) và không lặp (loop-free)

| path | Simple? | Loop-free? |
|---|---|---|
| <1,3,4,2> | | |
| <1,2,3,2> | | |
| <1,2,3,1,2> | | |
| <1,2,3,2,4> | | |

# Đường đơn giản (simple) và không lặp (loop-free)

| path | Simple? | Loop-free? |
|---|---|---|
| <1,3,4,2> | a | a |
| <1,2,3,2> | a | |
| <1,2,3,1,2> | | |
| <1,2,3,2,4> | a | |

# DU-path

- Một đường $P = \langle n_1, n_2, \ldots, n_i, n_j \rangle$ là DU-path của biến **v**, nếu **v** định nghĩa ở $n_1$ và một trong 2 điều kiện sau thoả mãn:
  - Có một c-use của **v** ở $n_j$ và đường $P$ là một **đường def-clear** đơn giản (simple)
  - Có một p-use của **v** ở cạnh $\langle n_j, n_k \rangle$ và $P$ là một **đường def-clear** không lặp (loop-free)

# Xác định DU-paths

| du-pair | path(s) | du-path? |
|---|---|---|
| (5,4) | <5,6,2,3,4> | |
| | <5,6,2,3,4,(6,2,3,4)*> | |
| (5,7) | <5,6,2,7> | |
| | <5,6,2,(3,4,6,2)*,7> | |
| (5,<3,4>) | <5,6,2,3,4> | |
| | <5,6,2,3,4,(6,2,3,4)*> | |
| (5,<3,5>) | <5,6,2,3,5> | |
| | <5,6,2,(3,4,6,2)*,3,5> | |

# Xác định DU-paths

| du-pair | path(s) | du-path? |
|---|---|---|
| (5,4) | <5,6,2,3,4> | a |
| | <5,6,2,3,4,(6,2,3,4)*> | |
| (5,7) | <5,6,2,7> | a |
| | <5,6,2,(3,4,6,2)*,7> | |
| (5,<3,4>) | <5,6,2,3,4> | a |
| | <5,6,2,3,4,(6,2,3,4)*> | |
| (5,<3,5>) | <5,6,2,3,5> | a |
| | <5,6,2,(3,4,6,2)*,3,5> | |

# Dataflow Coverage (tiếp)

- All-DU-paths
  - Với mỗi biến **v**, mỗi đường DU-path từ mỗi **def** của **v** tới mỗi c-use và mỗi p-use của **v** phải được phủ
- All-P-Uses/Some-C-Uses
  - Với mỗi biến **v**, ít nhất một đường def-clear từ mỗi **def** của **v** tới mỗi p-use của **v** phải được phủ
  - Trong trường hợp không có p-use của v, có ít nhất một đường def-clear tới c-use của v được phủ
- All-C-Uses/Some-P-Uses
  - Với mỗi biến **v**, ít nhất một đường def-clear từ mỗi **def** của **v** tới mỗi c-use của **v** phải được phủ
  - Trong trường hợp không có p-use của v, có ít nhất một đường def-clear tới p-use của v được phủ

# Dataflow Coverage (tiếp)

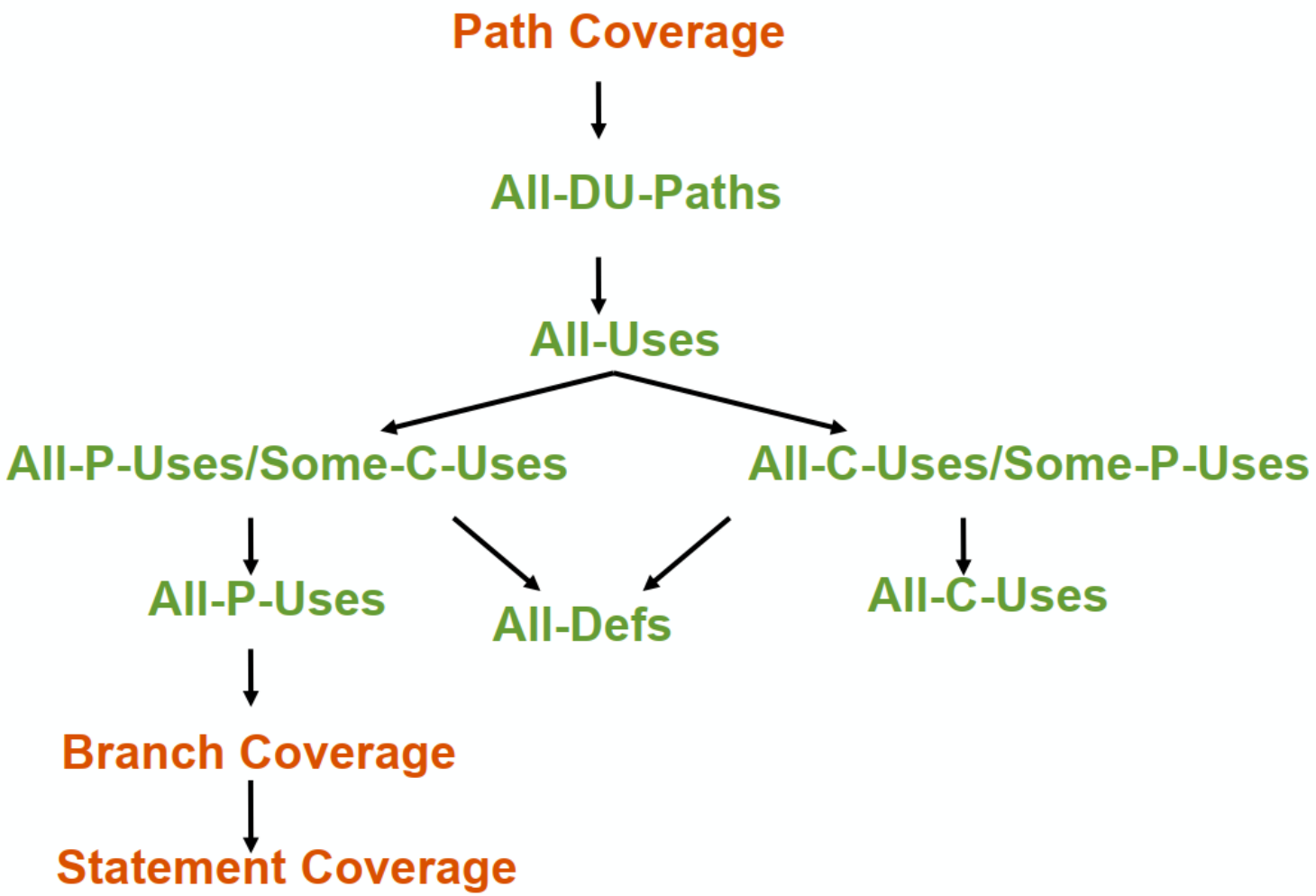
- All-P-Uses
  - Với mỗi biến **v**, ít nhất một đường def-clear từ mỗi **def** của **v** tới mỗi p-use của **v** phải được phủ
- All-C-Uses
  - Với mỗi biến **v**, ít nhất một đường def-clear từ mỗi **def** của **v** tới mỗi c-use của **v** phải được phủ

# Tổng kết các khái niệm về Dataflow Coverage

- Def cho một biến
  → All-defs coverage
- Use cho một biến: p-use và c-use
  → All-uses coverage
- Def-clear path và complete path
- DU-pair

- Simple path và loop-free path
- DU path
  →All-DU-paths
- All-P-Uses/Some-C-Uses
- All-C-Uses/Some-P-Uses
- All-P-Uses
- All-C-Uses

# Tổng kết


Path Coverage
All-DU-Paths
All-Uses
All-P-Uses/Some-C-Uses
All-C-Uses/Some-P-Uses
All-P-Uses
All-Defs
All-C-Uses
Branch Coverage
Statement Coverage

# Buổi học sau