HUTECH
Đại học Công nghệ Tp.HCM

KIỂM THỬ PHẦN MỀM

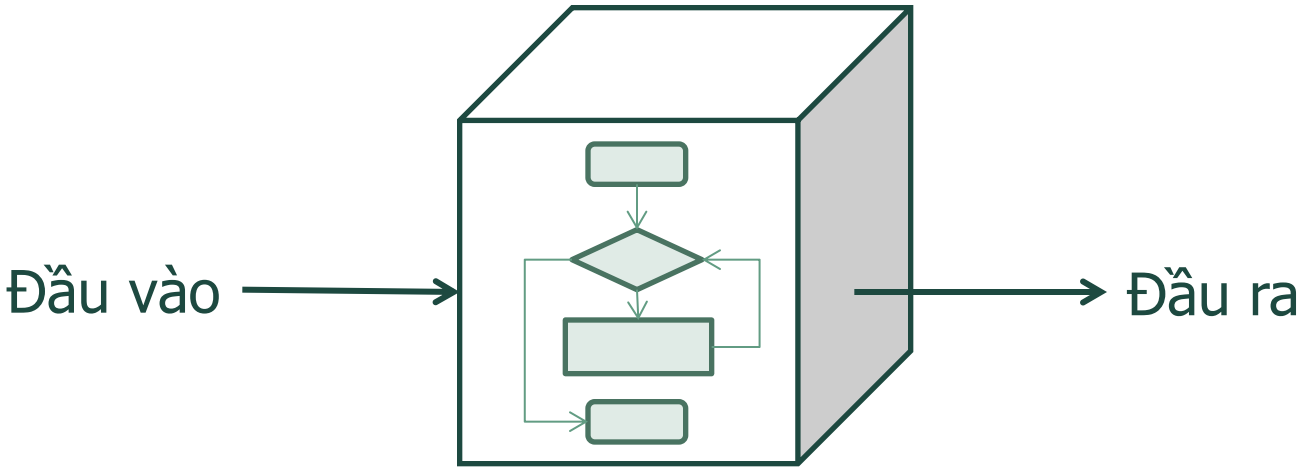
Bài 5:
THIẾT KẾ TEST CASE WHITEBOX

Thời gian: 6 tiết

NỘI DUNG

5.1. TỔNG QUAN KIỂM THỬ HỘP TRẮNG

- Kỹ thuật này còn gọi là structural testing, hoặc glass testing, hoặc open-box testing.
- Tester cần biết cách thức (**HOW**) thực thi bên trong của phần mềm.

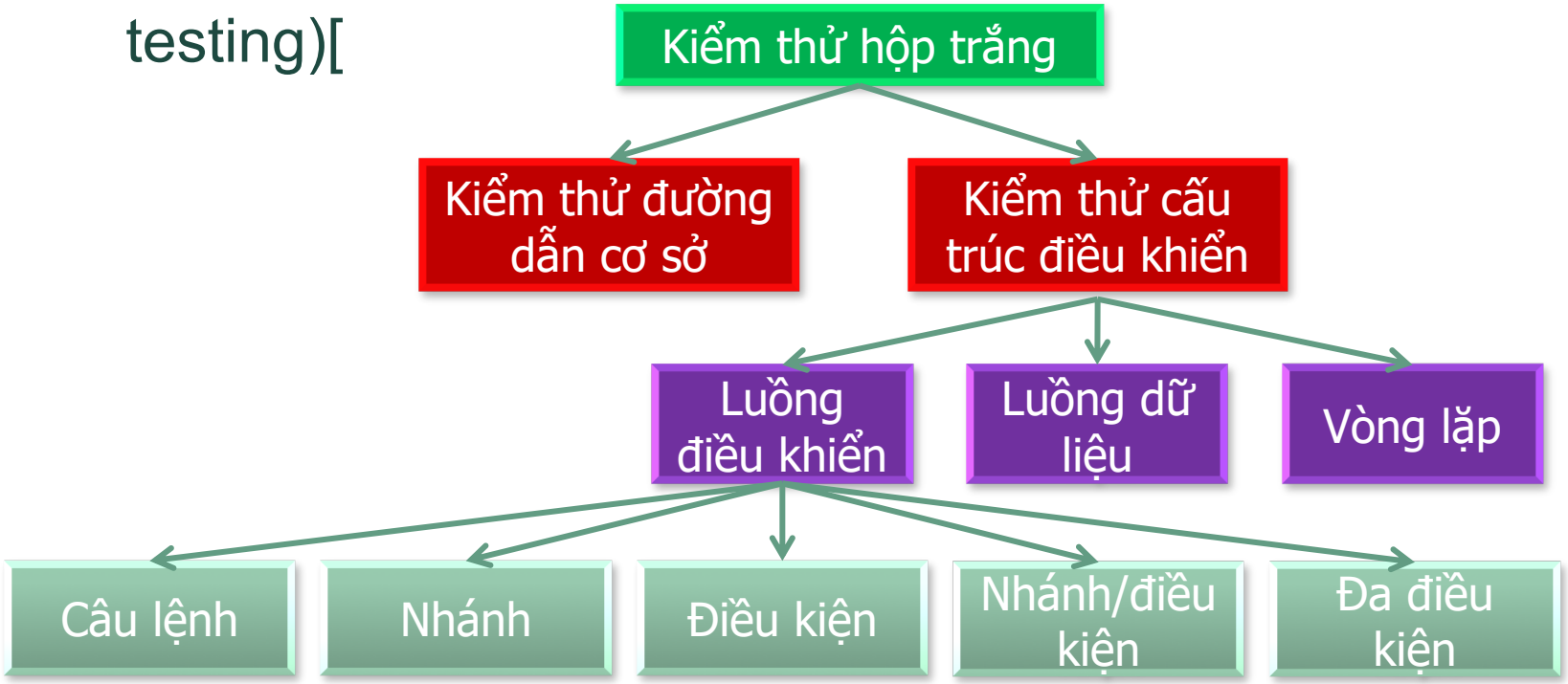


5.1. TỔNG QUAN KIỂM THỬ HỘP TRẮNG

- Ưu điểm
  - **Dễ dàng xác định loại dữ liệu** để kiểm tra, nên việc kiểm tra hiệu quả hơn.
  - Nhờ biết mã nguồn, nên tester **có thể phủ tối đa** khi viết kịch bản kiểm thử.
- Khuyết điểm
  - **Chi phí tăng** vì tester cần đọc, hiểu mã nguồn.
  - Khó xét hết các ngõ ngách trong mã nguồn nên **có thể sót đường dẫn** không được test.

5.1. TỔNG QUAN KIỂM THỬ HỘP TRẮNG

- Tiếp cận kiểm thử hộp trắng
  - Kiểm thử đường dẫn cơ sở (basis path testing)
  - Kiểm thử cấu trúc điều khiển (control structural testing)[



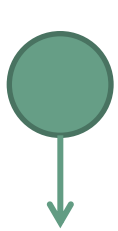
5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

- Kiểm thử đường dẫn cơ sở đảm bảo các **đường dẫn độc lập** được kiểm thử qua ít nhất một lần.
- Các bước thực hiện:
  1. Vẽ đồ thị luồng G.
  2. Tính độ phức tạp Cyclomatic V(G).
  3. Xác định tập cơ sở các đường dẫn độc lập.
  4. Viết test case thực thi mỗi đường dẫn trong tập cơ sở.

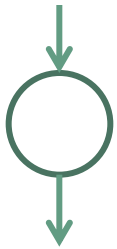
5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

Vẽ đồ thị luồng

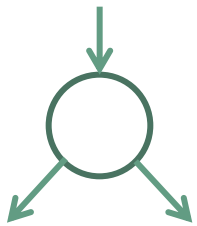
- Đồ thị luồng (flow graph) là đồ thị có hướng, dùng mô tả **luồng điều khiển logic**.
- Đồ thị luồng gồm hai thành phần cơ bản: **các đỉnh** tương ứng với các lệnh/nhóm lệnh **và các cạnh kết nối** tương ứng với dòng điều khiển giữa các lệnh/nhóm lệnh.
- Các nút trong đồ thị luồng



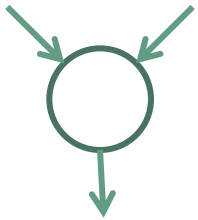
Nút bắt đầu



Khối xử lý



Nút vị từ hay nút quyết định



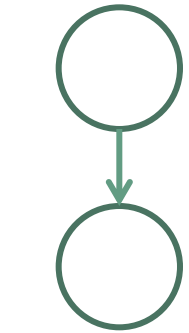
Nút nối

Nút kết thúc

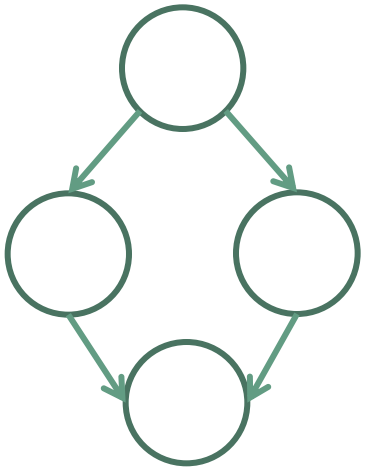
5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

Vẽ đồ thị luồng

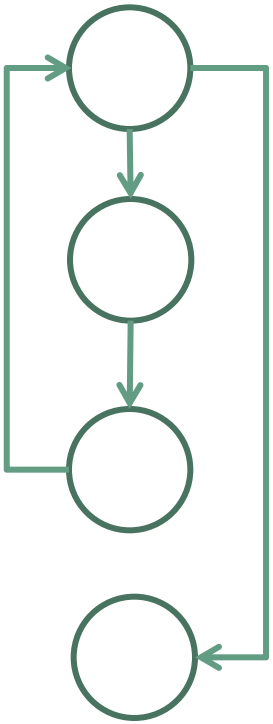
- Các đồ thị luồng cơ bản



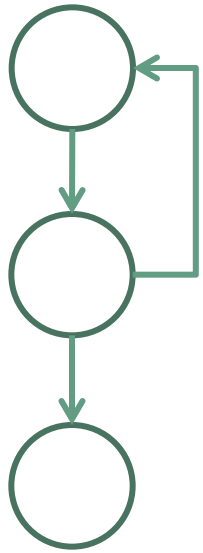
Tuyến tính



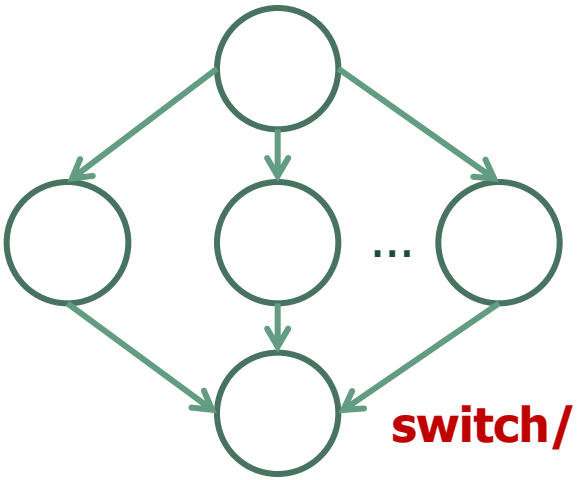
if/else



while



do..while

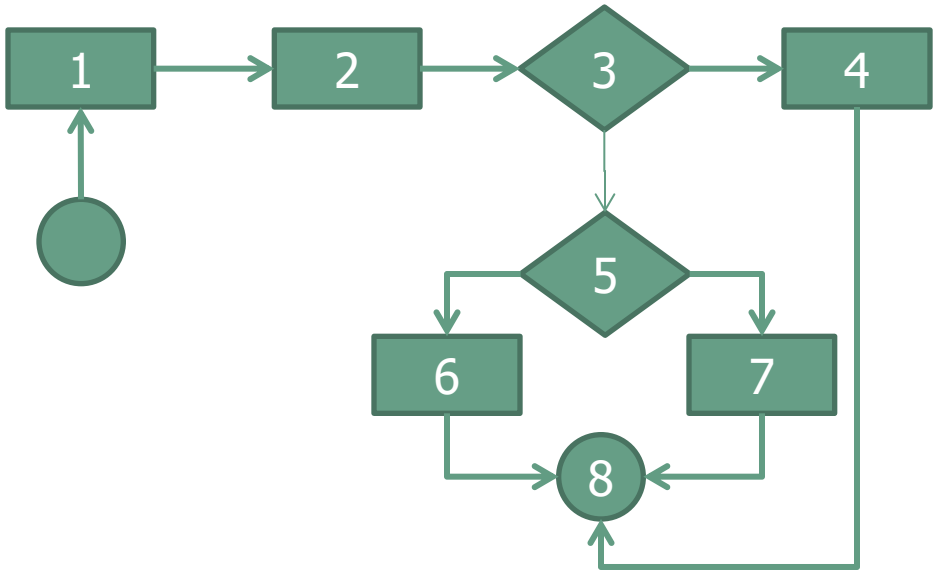


switch/case

5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

Vẽ đồ thị luồng

- **Đồ thị luồng** được xây dựng dựa trên **sơ đồ luồng** điều khiển (flow chart).
- Sơ đồ luồng điều khiển mô tả cấu trúc điều khiển của chương trình.



5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

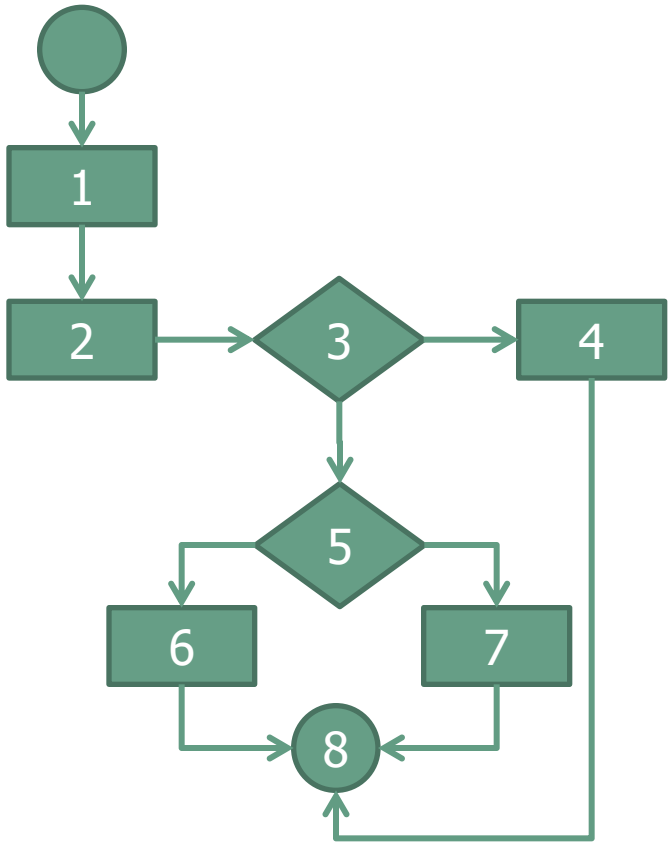
Vẽ đồ thị luồng

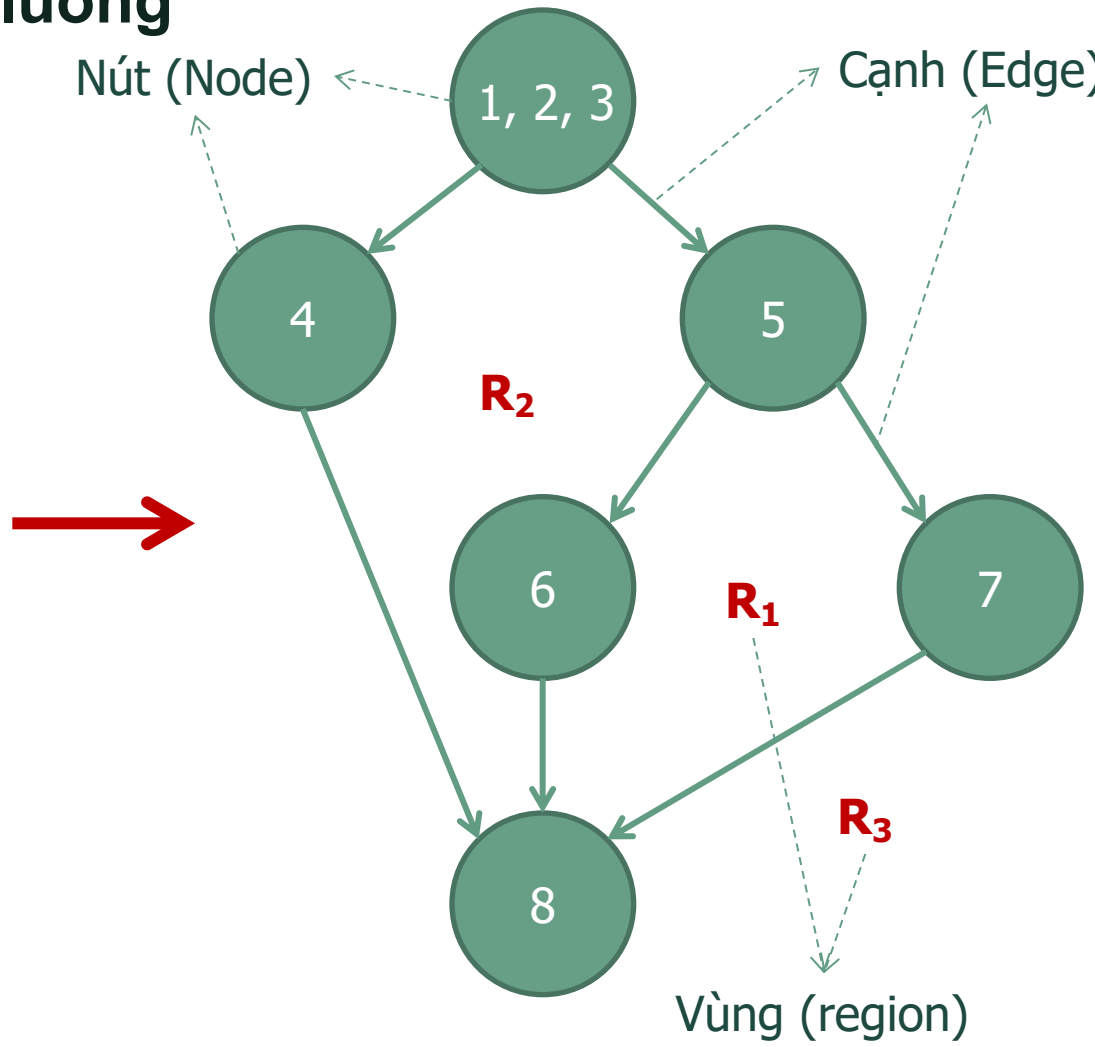
- Các **hình chữ nhật liên tiếp** và **một hình thoi** (điều kiện) liên tiếp nhau trong sơ đồ luồng sẽ tạo thành một nút (node) trong đồ thị luồng.
- Từ điều kiện rẽ thành các cạnh nối tới các node khác.
- Các cạnh và nút có thể hợp lại thành những **vùng khép kín**. Phạm vi bên ngoài đồ thị luồng cũng xem là một vùng.
- Nút chứa biểu thức điều kiện gọi là **nút quyết định** (node)

5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

Vẽ đồ thị luồng

Sơ đồ luồng → đồ thị luồng



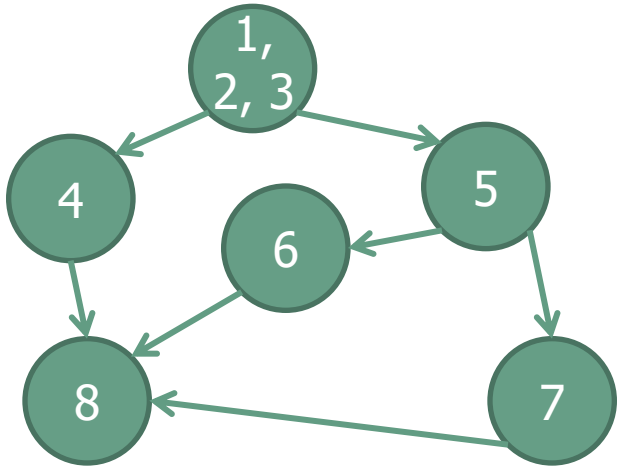


5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

Vẽ đồ thị luồng

Đường dẫn độc lập

- Đường dẫn độc lập là đường thông qua chương trình có **ít nhất** một tập các câu lệnh xử lý mới hoặc điều kiện mới.
- Trong đồ thị luồng, đường dẫn mới sẽ có cạnh mới.
- Ví dụ: các đường dẫn độc lập cơ sở của đồ thị luồng bên dưới
  - 1, 2, 3, 4, 8
  - 1, 2, 3, 5, 6, 8
  - 1, 2, 3, 5, 7, 8



5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

Vẽ đồ thị luồng

Đường dẫn độc lập

- Nếu thiết kế test case thực thi được các đường dẫn độc lập, thì đảm bảo được:
  - Mỗi **câu lệnh** được thực thi **ít nhất một lần**.
  - Mỗi **biểu thức điều kiện** (kể cả trường hợp true và false) được thực thi **ít nhất một lần**

5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

Tính độ phức tạp cyclomatic

- Độ phức tạp Cyclomatic là một độ đo phần mềm cung cấp thước đo **định lượng độ phức tạp về mặt logic** của chương trình.
- Trong ngữ cảnh của kiểm thử đường thì độ đo này là **số lượng các đường độc lập** trong tập cơ sở của chương trình và cung cấp **chặn trên** (upper bound) số lượng các test case được thực thi **đảm bảo phủ các câu lệnh** của chương trình.

5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

Tính độ phức tạp cyclomatic

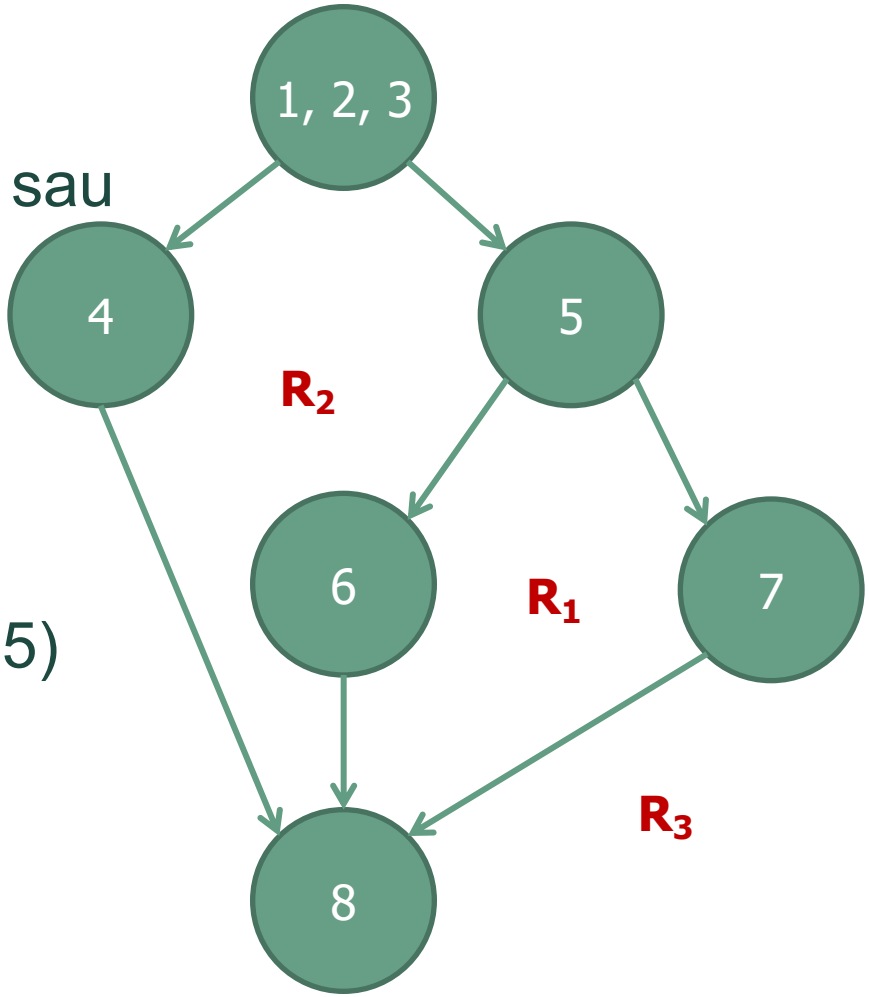
- Độ đo này được tính bằng một trong ba cách sau (ký hiệu V(G) là giá trị độ phức tạp Cyclomatic của đồ thị luồng G):
  - Số lượng các vùng (region) của G
  - V(G) = E – N + 2
    - E là số cạnh của G
    - N là số nút của G
  - V(G) = P + 1
    - P là số các nút vị từ của G

5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

Tính độ phức tạp cyclomatic

- Ví dụ: đồ thị luồng như hình sau
- Có 3 region
- V(G) = E – N + 2

  = 7 – 6 + 2 = 3
- V(G) = P + 1 = 2 + 1 = 3
  - 2 nút vị từ là (1, 2, 3) và (5)

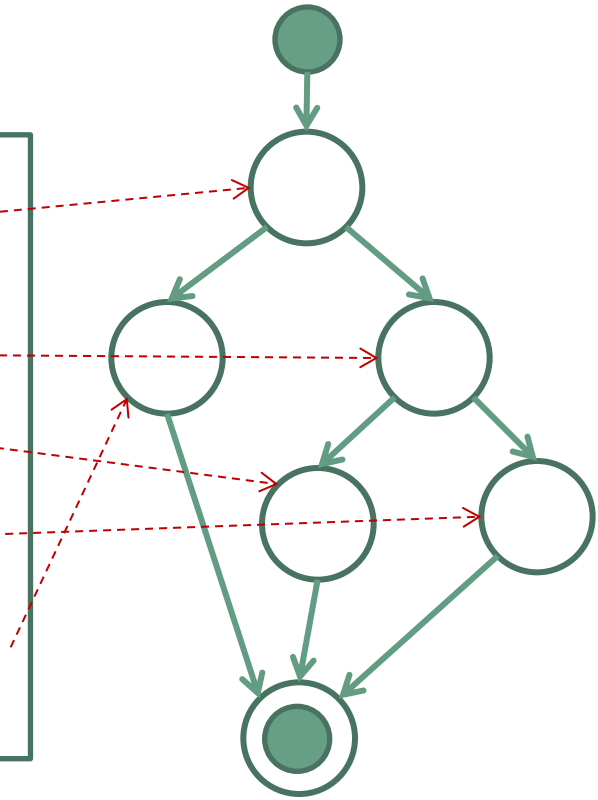


5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

Ví dụ 1

- Viết các test case kiểm thử chương trình giải và biện luận phương trình ax + b = 0, với đó a, b là các số thực.
- Chương trình như sau:

```
void giaiPTBacNhat(double a, double b)
{
   if (a == 0)
      if (b == 0)
         printf("PT vo so nghiem\n");
      else
         printf("PT vo nghiem\n");
   else
         printf("PT co nghiem:%f", -b / a);
}
```

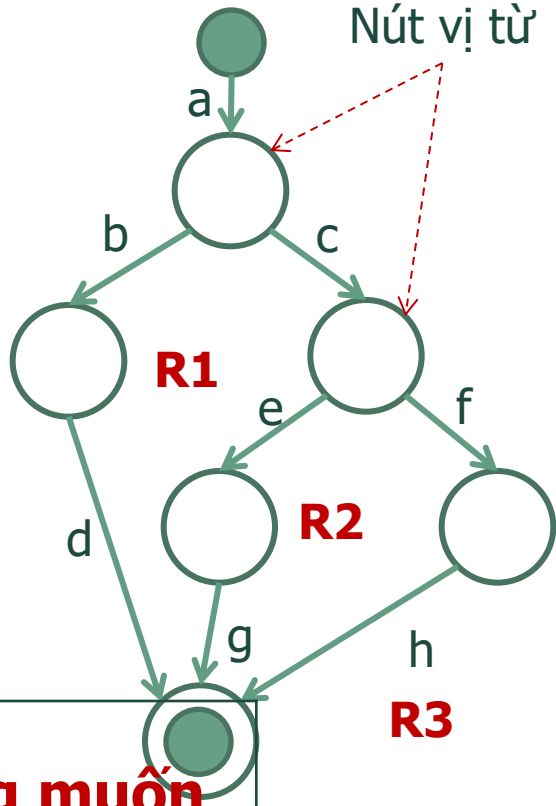


Chương trình minh họa bằng C++

5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

Ví dụ 1 (tt)

- Xác định độ phức tạp Cyclomatic
  - Số lượng các vùng của G: 3
  - V(G) = E – N + 2 = 8 – 7 + 2 = 3
  - V(G) = P + 1 = 2 + 1 = 3
- Xác định tập đường dẫn cơ sở
  - a, b, d
  - a, c, e,g
  - a, c, f,h



Đường dẫn	Đầu vào		Đầu ra mong muốn
	a	b	
a, b, d	5	-10	Nghiem x = 2
a, c, e	0	0	PT vo so nghiem
a, c, f	0	5	PT vo nghiem

5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

Ví dụ 1

- ✓ Thiết kế test case cho từng đường dẫn cơ sở

Đường dẫn	Đầu vào		Đầu ra mong muốn
	a	b	
a, b, d	5	-10	Nghiem x = 2
a, c, e	0	0	PT vo so nghiem
a, c, f	0	5	PT vo nghiem

5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

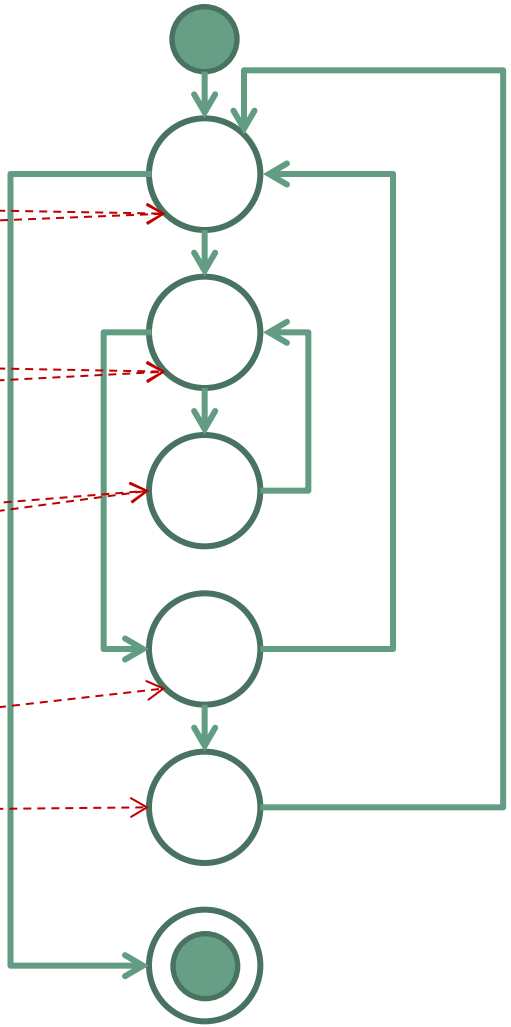
Ví dụ 2

- Viết các test case để kiểm thử chương trình tính tổng các chữ số của số nguyên dương n cho đến khi tổng nhỏ hơn 10.
- Ví dụ:
  - 123456 →1 + 2 + 3 + 4 + 5 + 6=21 → 2 + 1 → 3 < 10
  - 857 → 8 + 5 + 7 = 20 → 2 + 0 = 2 < 10

5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

Ví dụ 2

```
int tinhTong(int n)
{
    int tong = 0;
    while (n)
    {
        tong = 0;
        while (n)
        {
            tong += n % 10;
            n = n / 10;
        }

        if (tong > 10)
            n = tong;
    }
    return tong;
}
```

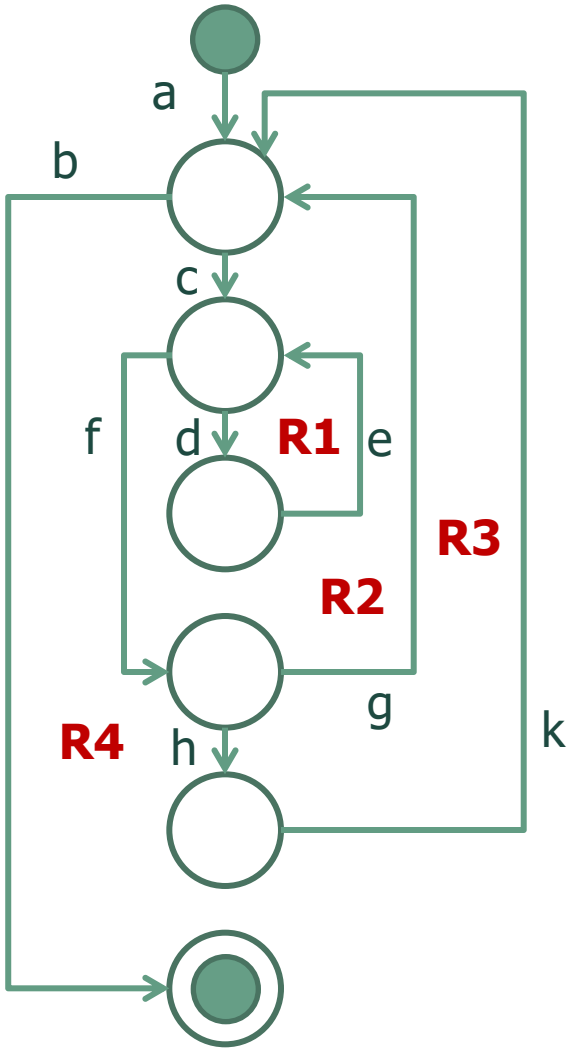


Chương trình minh họa bằng C++

5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

Ví dụ 2

- Xác định độ phức tạp Cyclomatic
  - Số lượng các vùng của G: 4
  - V(G) = E – N + 2 = 9 – 7 + 2 = 4
  - V(G) = P + 1 = 3 + 1 = 4
- Chọn tập đường dẫn cơ sở
  - a,b
  - a,c,d,e,f, g,b
  - a,c,d,e,f, ,k,b
  - a,c,d,,e,f,h,k,b



5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

Ví dụ 2

- Thiết kế test case cho từng đường dẫn cơ sở

Đường dẫn	Đầu vào	Đầu ra mong muốn
	n	
a, c, d, e,f, g, b	12	Tổng là 3
a, c, d, e, f, k, b	987	Tổng là 6
a, c, d, e, f, h, k, b		
a,b	6	Tổng là 6

5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

Bài tập 1

- Thiết kế các test case để phủ đường dẫn cơ sở của đoạn chương trình sau:

```
bool xetHocBong(double diemMH[], int soMH, int diemRL)
{
    double tongDiem = 0;
    for (int i = 0; i < soMH; i++)
        if (diemMH[i] < 5)
            return false;
        else
            tongDiem += diemMH[i];
    double diemTB = tongDiem / soMH;
    if (diemTB >= 9 || (diemTB >= 7 && diemRL >= 80))
        return true;
    return false;
}
```

5.2. KIỂM THỬ ĐƯỜNG DẪN CƠ SỞ

Bài tập 2

```
int BinarySearch(int a[], int n, int x)
{
    int middle, left = 0, right = n - 1, idx = -1;
    while (left <= right) {
        middle = (left + right) / 2;
        if (a[middle] == x){
            idx = middle;
            break;
        }
        else if (a[middle] > x)
            right = middle - 1;
        else
            left = middle + 1;
    }
    return idx;
}
```

5.3. KIỂM THỬ BAO PHỦ

- Kiểm thử bao phủ dùng kiểm tra mức độ phủ (coverage) của các test case.
- Các loại kiểm thử bao phủ:
  - Phủ câu lệnh (statement coverage)
  - Phủ nhánh (branch coverage)
  - Phủ đường (path coverage)
  - Phủ điều kiện (condition coverage)
  - Phủ nhánh và điều kiện (condition and branch coverage)
  - Phủ đa điều kiện (multi-conditions coverage)

5.3. KIỂM THỬ BAO PHỦ

Phủ câu lệnh

- Phủ câu lệnh (statement coverage): **mỗi câu lệnh** được thực thi ít nhất một lần.
- Ví dụ: hàm xét học bổng như sau

```
bool xetHocBong(double d1, double d2, double d3, double diemRL)
{
   double diemTB = 0.0;
   if (d1 >= 5 && d2 >= 5 && d3 >= 5)
      diemTB = (d1 + d2 + d3) / 3;

   bool kq = false;
   if ((diemTB >= 8.5) || (diemTB >= 7.0 && diemRL >= 70))
      kq = true;

   return kq;
}
```

Chương trình minh họa bằng C++

5.3. KIỂM THỬ BAO PHỦ

Phủ câu lệnh

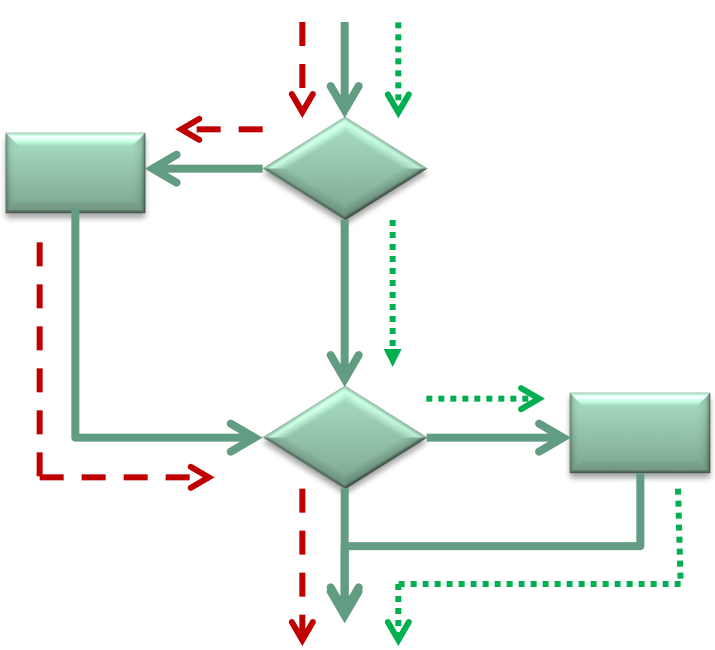
- Để kiểm thử phủ câu lệnh trong hàm trên chỉ cần test case sau:

Đầu vào				Đầu ra mong muốn
d1	d2	d3	diemRL	
7	7	7	70	Hàm trả về kết quả true

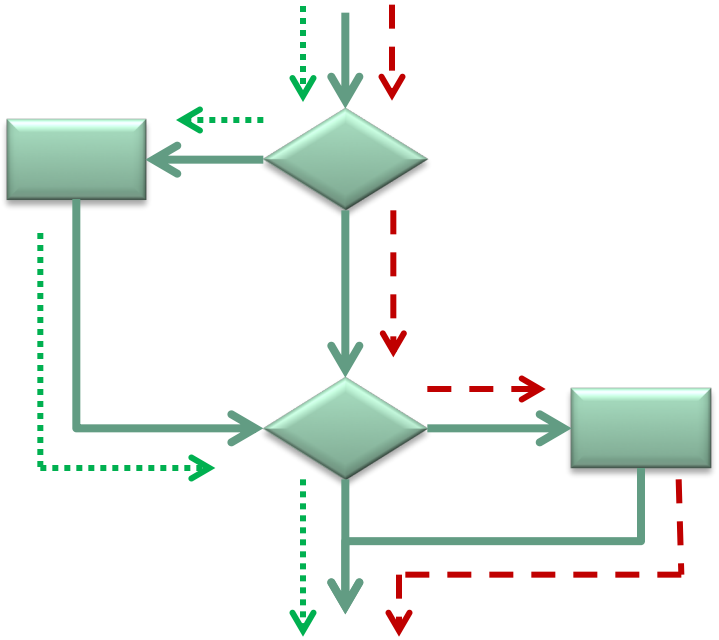
5.3. KIỂM THỬ BAO PHỦ

Phủ nhánh

- Phủ nhánh (branch coverage): **mỗi nhánh** phải được thực hiện ít nhất một lần.
- Phủ nhánh **đảm bảo** phủ câu lệnh.



Hoặc



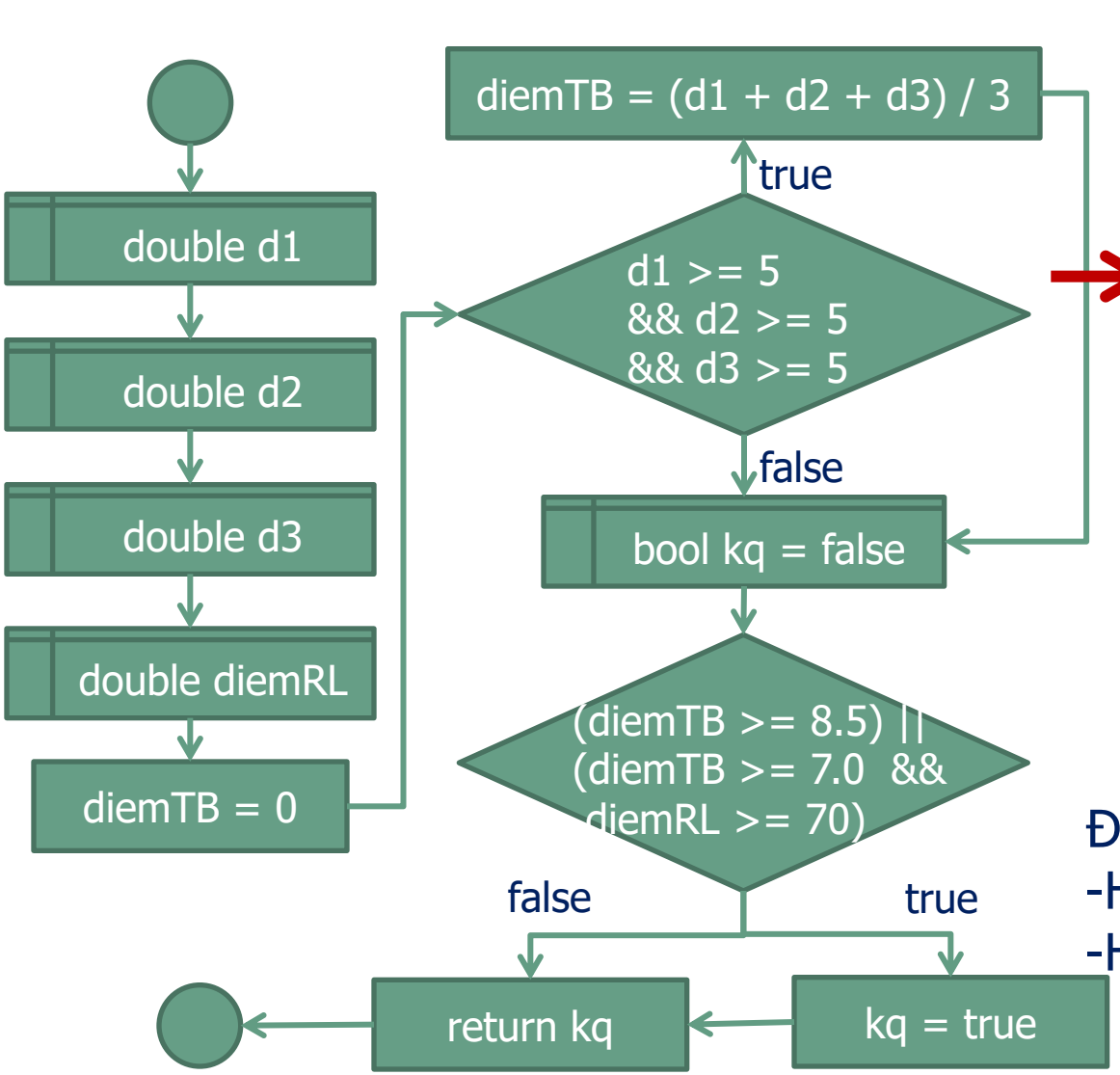
5.3. KIỂM THỬ BAO PHỦ

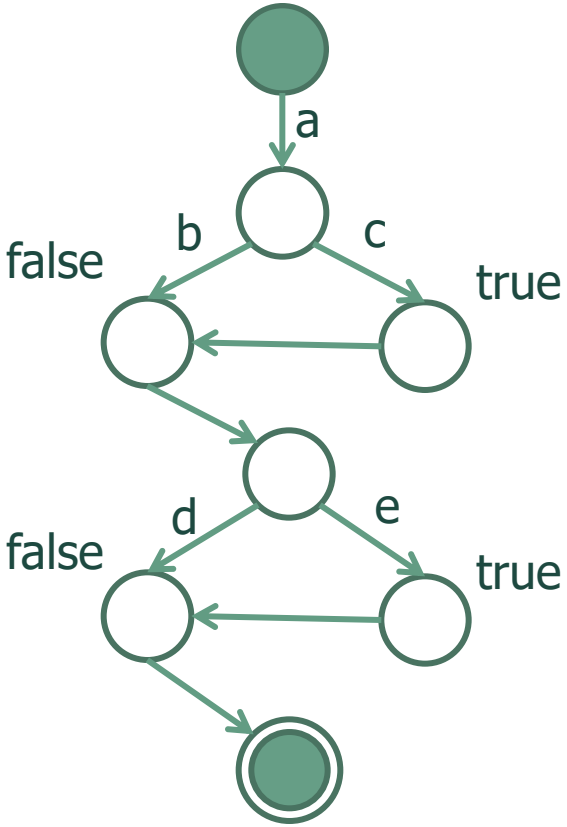
Phủ nhánh

- Nhận xét: trong **đồ thị luồng**
  - Phủ nhánh có nghĩa là **các cạnh được đi qua ít nhất một lần**.
  - Để phủ nhánh phải thiết kế dữ liệu kiểm thử sao cho **mỗi nút vị từ** (predicate) xảy ra tất cả các kết quả (true/false) có thể của nó, nên phủ nhánh còn gọi là **phủ quyết định** (decision coverage).

5.3. KIỂM THỬ BAO PHỦ

Phủ nhánh





Để phủ các nhánh:
-Hoặc **abd** (FF) và **ace** (TT)
-Hoặc **abe** (FT) và **acd** (TF)

5.3. KIỂM THỬ BAO PHỦ

Phủ nhánh

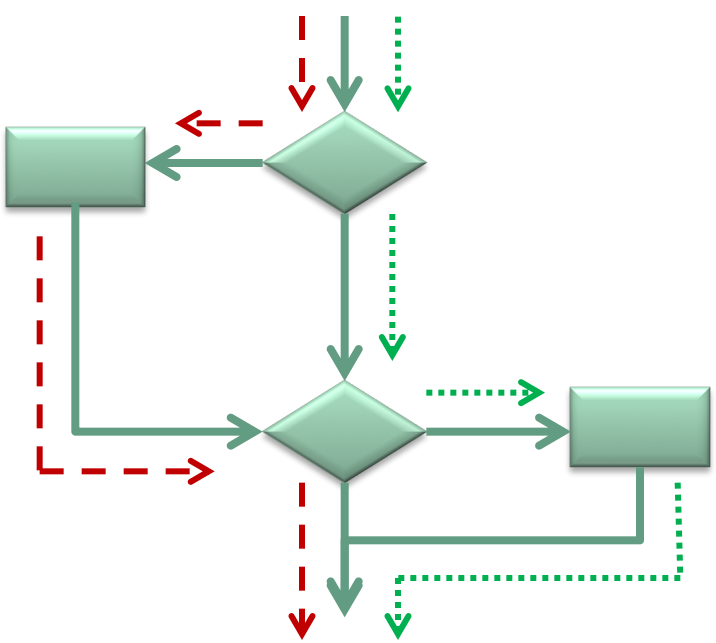
- Chọn abd và ace kiểm thử phủ nhánh, thiết kế các test case như sau:

Nhánh	Đầu vào				Đầu ra mong muốn
	d1	d2	d3	diemRL	
abd	4	5	5	70	Hàm trả về kết quả false
ace	7	7	7	70	Hàm trả về kết quả true

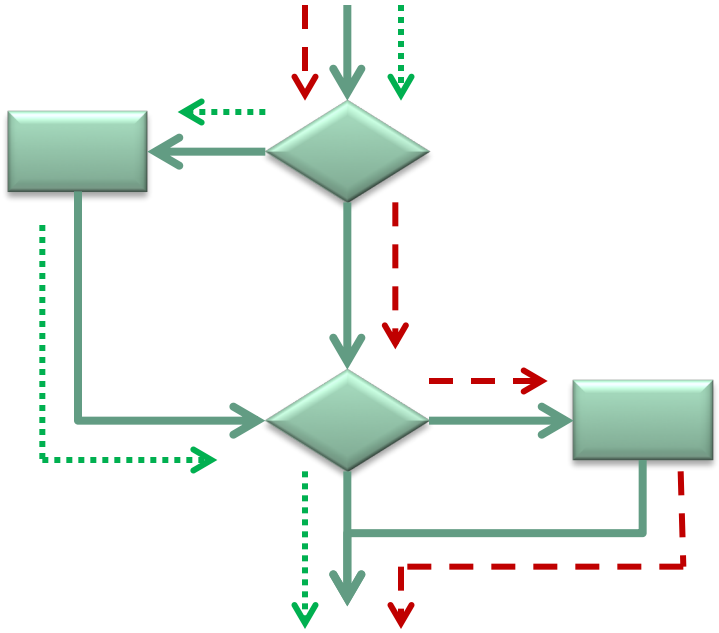
5.3. KIỂM THỬ BAO PHỦ

Phủ đường dẫn

- Phủ đường dẫn (path coverage): mỗi đường dẫn qua ít nhất một lần.
- Đường là một tập các nhánh, nên **phủ đường chắc chắn phủ nhánh**, nhưng ngược lại chưa chắc đúng.



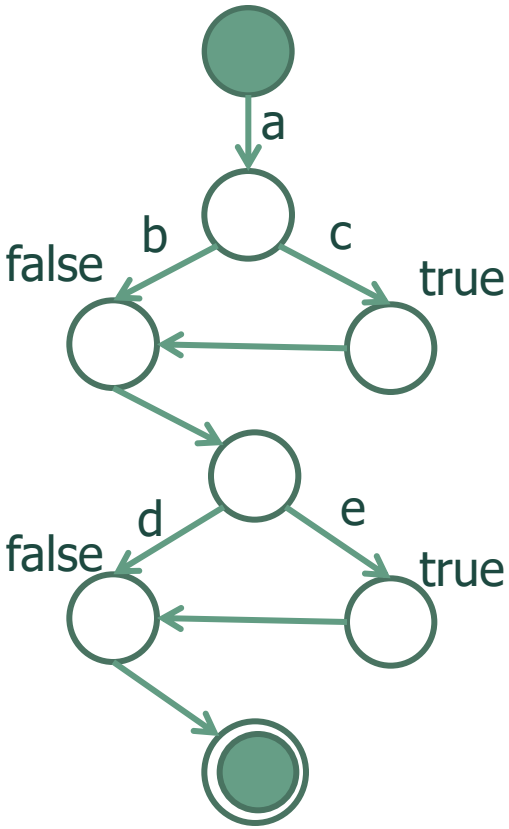
Và



5.3. KIỂM THỬ BAO PHỦ

Phủ đường dẫn

- Ví dụ: quay lại ví dụ xét học bổng, phủ đường dẫn trong đồ thị luồng bằng 4 test case sau đi qua các đường abd, ace, acd, abe.
- Chú ý: không có test case nào đi qua đường abe (đường bất khả thi). Do đó, để phủ đường chỉ cần 3 test case sau



Đầu vào				Đầu ra mong muốn
d1	d2	d3	diemRL	
4	5	5	70	Hàm trả về kết quả false
7	7	7	70	Hàm trả về kết quả true
7	7	7	60	Hàm trả về kết quả false

5.3. KIỂM THỬ BAO PHỦ

Phủ điều kiện

- Thông thường vị từ quyết định nhánh sẽ được thực thi là **tổ hợp nhiều điều kiện**.
- Ví dụ:
  - if (d1 >= 5 && d2 >= 5 && d3 >= 5) ...
  - if ((diemTB >= 8.5) ||

    (diemTB >= 7.0 && diemRL >= 70)) ...
  - if ((nam % 400 == 0) ||

    (nam % 4 == 0 && nam % 100 != 0)) …

5.3. KIỂM THỬ BAO PHỦ

Phủ điều kiện

- Phủ điều kiện (condition coverage): mỗi **điều kiện trong các vị từ** được thực hiện ít nhất một lần cho cả trường hợp **true và false** (**không bắt buộc** các kết hợp giữa chúng).
- Phủ điều kiện **chứa chắc** đảm bảo phủ các nhánh.

5.3. KIỂM THỬ BAO PHỦ

Bài tập

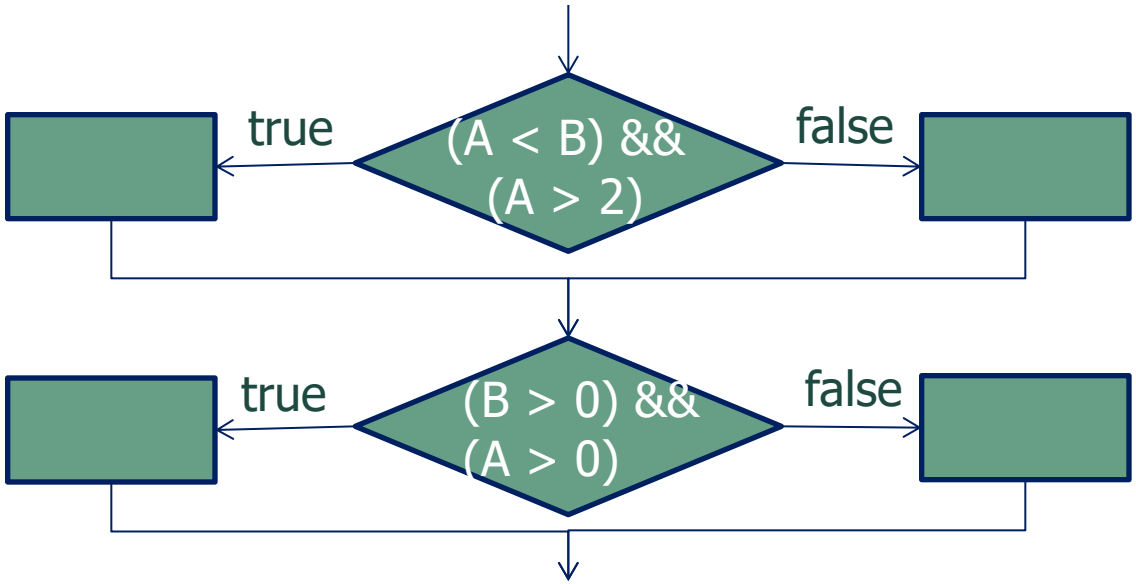
- ✓ Số test case tối thiểu để phủ câu lệnh?
- ✓ Số test case tối đa phủ đường dẫn cơ sở?
- ✓ Số test case tối thiểu phủ điều kiện?
- ✓ Thiết kế test case phủ đường dẫn cơ sở?
- ✓ Thiết kế các test case phủ nhánh của hàm?

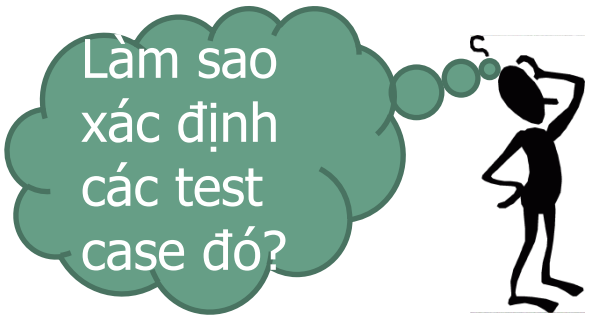
```
int reverse(int n)
{
    bool isNegative = false;
    if (n < 0)
    {
        isNegative = true;
        n = -n;
    }

    int result = 0;
    while (n > 0)
    {
        result = result * 10 + n % 10;
        n /= 10;
    }
    if (isNegative == true)
        result = - result;
    return result;
}
```

5.3. KIỂM THỬ BAO PHỦ

Phủ nhánh và điều kiện

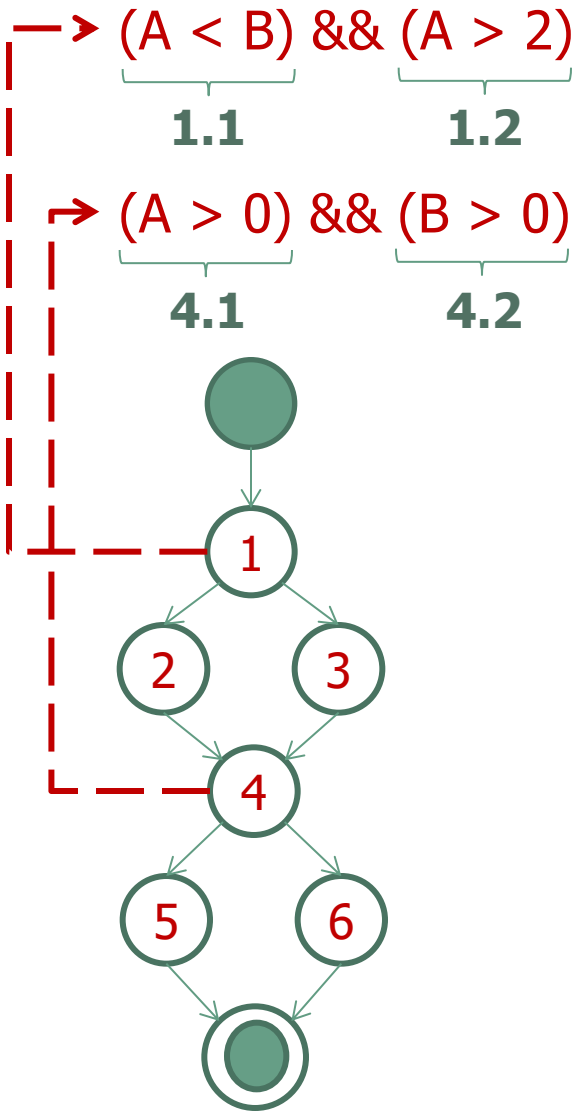
- Phủ nhánh và điều kiện: mỗi **điều kiện** trong các vị từ và **các nhánh** rẽ từ các vị từ đó cũng được thực thi ít nhất một lần.
- Ví dụ: phủ nhánh và điều kiện trong sơ đồ luồng sau với các test case
  - A = 3, B = 4
  - A = -3, B = 4
  - A = -3, B = -4

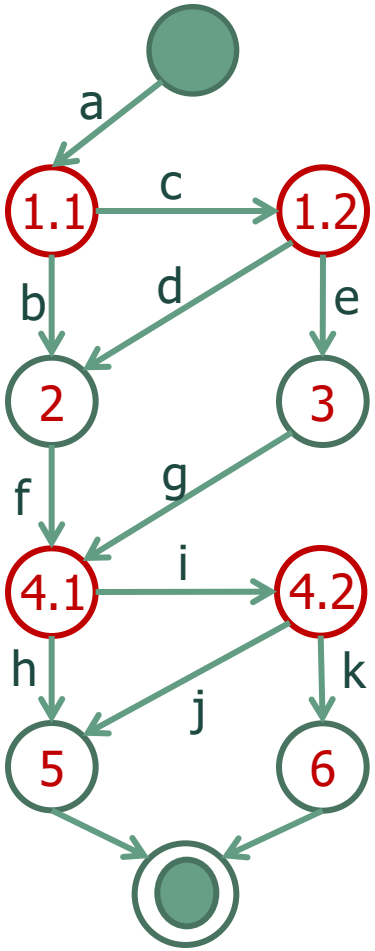




5.3. KIỂM THỬ BAO PHỦ

Phủ nhánh và điều kiện





Ta chọn các **đường dẫn phủ nhánh** của đồ thị này:

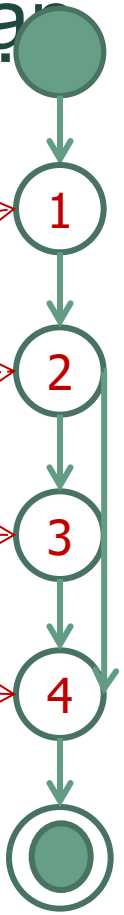
- a, c, e, g, i, k
- a, c, d, f, i, j
- a, b, f, h

5.3. KIỂM THỬ BAO PHỦ

Phủ nhánh và điều kiện

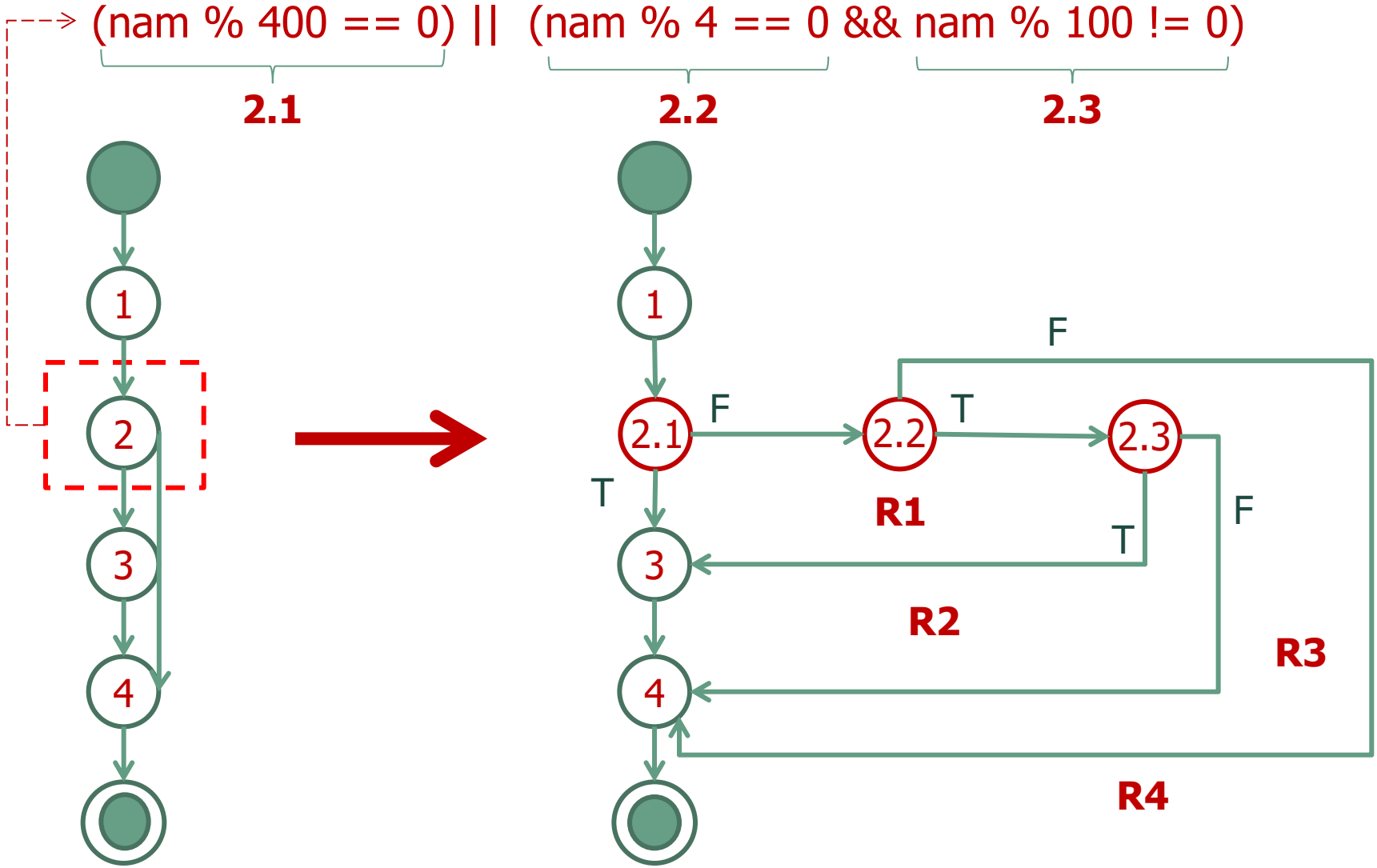
- Ví dụ: thiết kế các test case phủ nhánh và điều kiện của hàm kiểm tra năm nhuận

```
bool ktNamNhuan(int nam)
{
   bool kq = false;
   if ((nam % 400 == 0) ||
       (nam % 4 == 0 && nam % 100 != 0))
      kq = true;

   return kq;
}
```



5.3. KIỂM THỬ BAO PHỦ

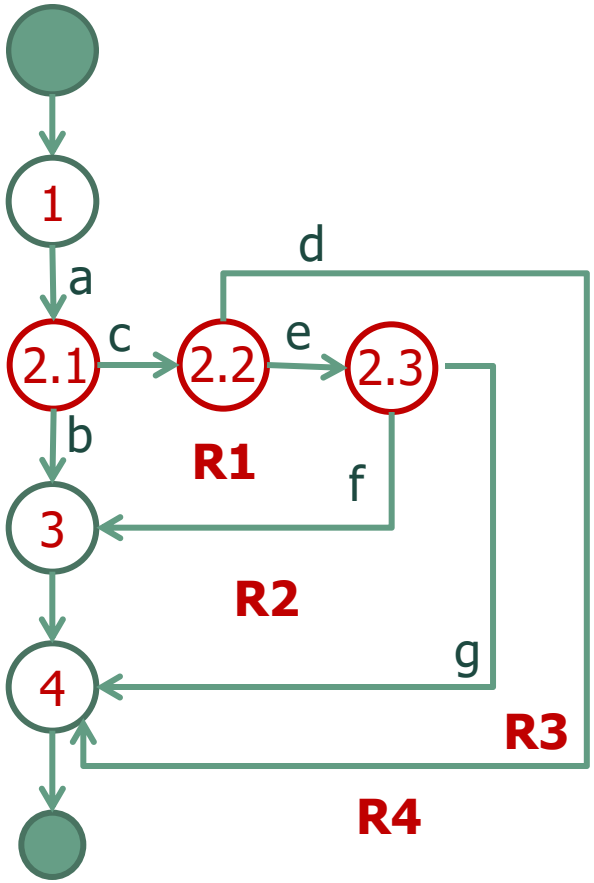
Phủ nhánh và điều kiện



5.3. KIỂM THỬ BAO PHỦ

Phủ nhánh và điều kiện

- Độ phức tạp Cyclomatic:
  - Số lượng các vùng của G: 4
  - V(G) = E – N + 2 = 10 – 8 + 2 = 4
  - V(G) = P + 1 = 3 + 1 = 4
- Các đường dẫn sau đảm bảo phủ nhánh và điều kiện:
  - a, c, e, f
  - a, c, e, g
  - a, c, d
  - a, b



5.3. KIỂM THỬ BAO PHỦ

Phủ nhánh và điều kiện

- Để phủ nhánh và điều kiện cần tối thiểu các test case minh họa:

Đường dẫn	Đầu vào	Đầu ra mong muốn
	Năm	
a, c, e, f	2016	Năm nhuận
a, c, e, g	1900	Không phải năm nhuận
a, c, d	2017	Không phải năm nhuận
a, b	1600	Năm nhuận

5.3. KIỂM THỬ BAO PHỦ

Bài tập

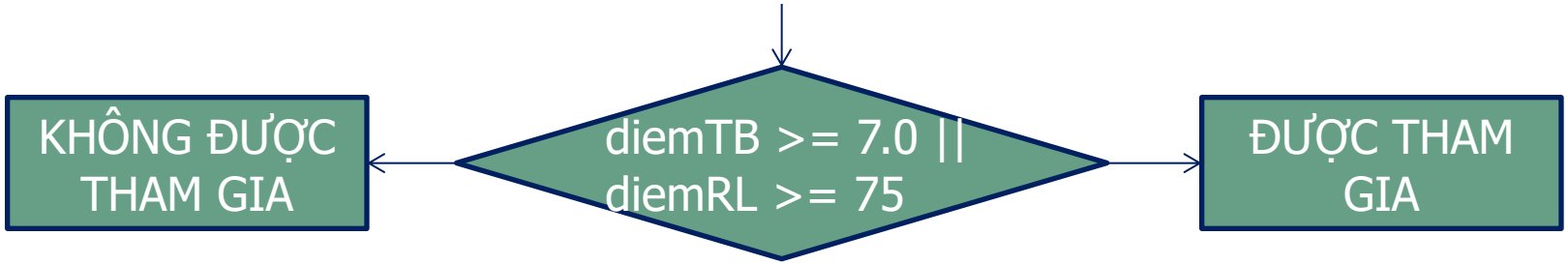
- Viết các test case phủ nhánh và điều kiện hàm:

```
bool xetHocBong(double diemMH[], int soMH, int diemRL)
{
   if (soMH > 0)
   {
        int i;
        double tongDiem = 0;
        for (i = 0; i < soMH; i++)
           tongDiem = tongDiem + diemMH[i];

        double diemTB = tongDiem / soMH;
        if (diemTB >= 8 || (diemTB >= 7 && diemRL >= 80))
           return true;
   }

   return false;
}
```

5.3. KIỂM THỬ BAO PHỦ

Phủ đa điều kiện

- ✓ Phủ đa điều kiện (multi-conditions coverage): mỗi điều kiện trong biểu thức vị từ và **các kết hợp** giữa chúng được thực hiện ít nhất một lần cho các trường hợp true và false.
- ✓ Ví dụ: một sinh viên được xét tham gia chiến dịch tình nguyện của trường nếu hoặc điểm trung bình từ 7.0 trở lên hoặc điểm rèn luyện từ 75 trở lên.



5.3. KIỂM THỬ BAO PHỦ

Phủ đa điều kiện

- Với các test case sau đảm bảo phủ điều kiện:
  - diemTB = 7.0 (true) và diemRL = 75 (true)
  - diemTB = 6.5 (false) và diemRL = 70 (false)
- Để phủ đa điều kiện cần bổ sung các test case:
  - diemTB = 6.5 (false) và diemRL = 75 (true)
  - diemTB = 7.0 (true) và diemRL = 70 (false)
  - Ghi chú: Với các ngôn ngữ lập trình hiện đại thì test case này không thể xảy ra vì diemTB = 7.0 (true) đủ quyết định kết quả biểu thức nên nó sẽ không xét biểu thức con còn lại.

5.4. KIỂM THỬ LUỒNG DỮ LIỆU

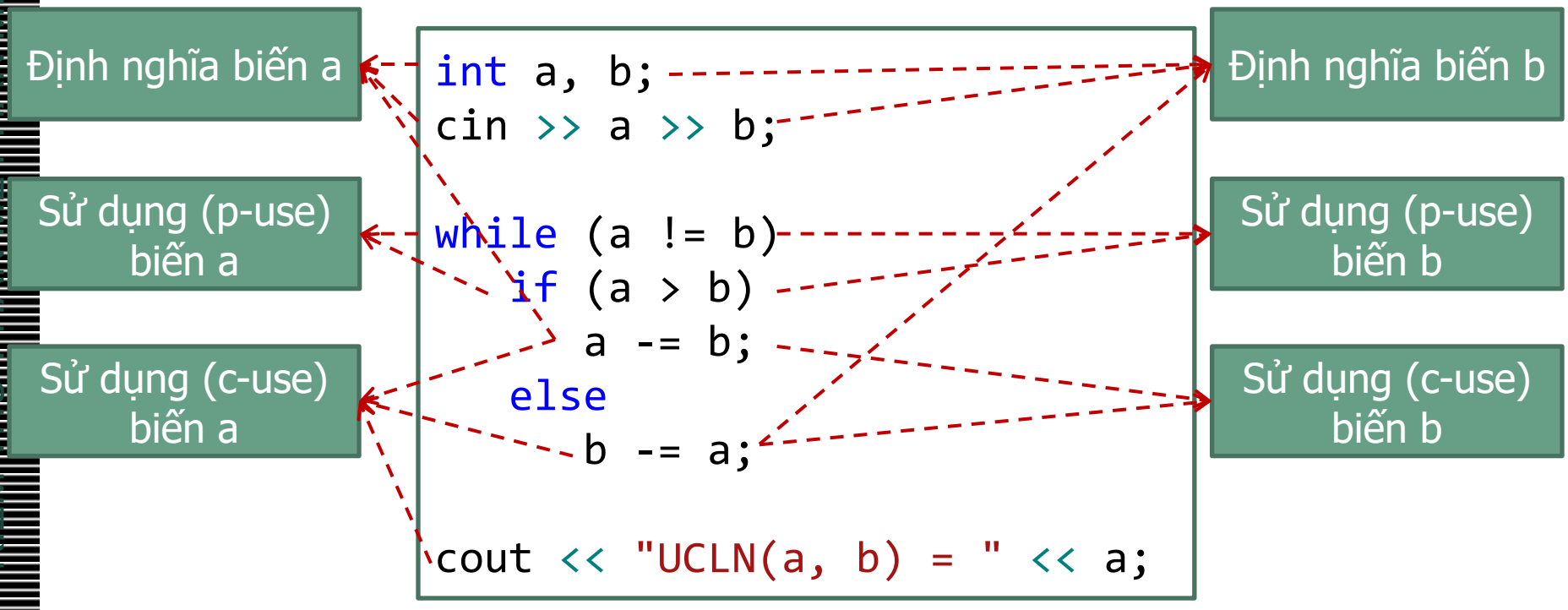
- Phương pháp kiểm thử luồng dữ liệu sẽ kiểm thử **vòng đời của biến** trong từng luồng thực thi của chương trình.
- Vòng đời của một biến được thể hiện thông qua ba hành động:
  - Định nghĩa biến (**Define**).
  - Sử dụng biến (**Use**).
  - Xóa biến (**Delete**).

5.4. KIỂM THỬ LUỒNG DỮ LIỆU

- Kiểm thử luồng dữ liệu lựa chọn các đường dẫn để kiểm thử dựa trên **vị trí** định nghĩa (**define**) và sử dụng (**use**) các biến trong chương trình.
- Đặt
  - **DEF(S)** = {X| câu lệnh S chứa định nghĩa biến X}
    - X = … (nhập, gán, gọi thủ tục)
  - **USE(S)** = {X| câu lệnh S sử dụng biến X}
    - … = X … (xuất, gán, điều kiện)
    - Nếu câu lệnh S là biểu thức vị từ thì ký hiệu là **p-use**
    - Nếu câu lệnh S là biểu thức tính toán thì ký hiệu là **c-use**

5.4. KIỂM THỬ LUỒNG DỮ LIỆU

Ví dụ



Chương trình minh họa bằng C++

5.4. KIỂM THỬ LUỒNG DỮ LIỆU

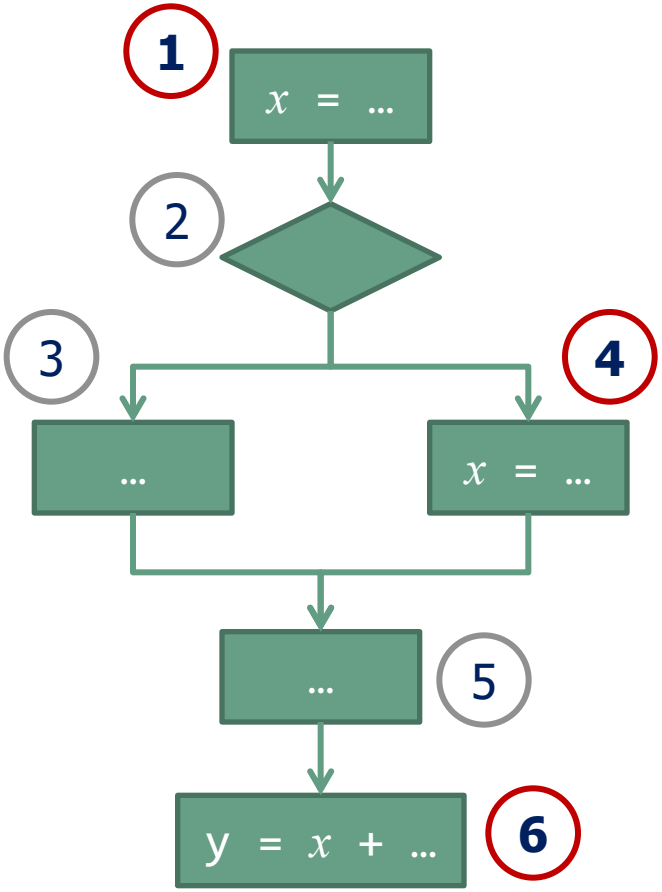
- Kiểm thử luồng dữ liệu giúp phát hiện các vấn đề sau:
  - Một biến được **khai báo, nhưng không sử dụng**.
  - Một biến **sử dụng nhưng không khai báo**.
  - Một biến được **định nghĩa nhiều lần** trước khi được sử dụng.
  - **Xóa biến trước khi sử dụng**.

5.4. KIỂM THỬ LUỒNG DỮ LIỆU

- ✓ Cho biến $x \in DEF(S) \cap USE(S')$, trong đó S và S' là các câu lệnh.
- ✓ **Đường dẫn DU** (definition-use path) của biến *x* là đường nối từ S đến S' trên đồ thị luồng sao cho **không tồn tại** một định nghĩa nào khác của *x* trên đường này.
- ✓ **Cặp DU** (definition-use pairs) của biến là cặp S và S', sao cho **tồn tại ít nhất** một đường DU nối S và S'.

5.4. KIỂM THỬ LUỒNG DỮ LIỆU

- DEF(1) = {*x*, …}
- DEF(4) = {*x*, …}
- USE(6) = {*x*, …}
- (1, 2, 3, 5, 6) và (4, 5, 6) là các đường dẫn DU của biến *x*.
- (1, 2, 4, 5, 6) không là đường của biến *x* vì nó được định nghĩa lại ở câu lệnh 4.
- Các cặp DU là (1, 6) và (4, 6)



5.4. KIỂM THỬ LUỒNG DỮ LIỆU

- Ví dụ: cho biết đầu là cặp DU của biến **kq**

```
bool ktNguyenTo(int n)
{
   bool kq = false; // (1)
   if (n >= 2)
   {
       kq = true; // (2)
       for (int i=2; i<=sqrt(n) && kq == true; i++)//(3)
          if (n % i == 0)
             kq = false;   // (4)
   }
   return kq; // (5)
}
```

5.4. KIỂM THỬ LUỒNG DỮ LIỆU

Ví dụ: cho biết đâu là cặp DU của biến **heSo**

```
float tinhLuong(int loaiNhanVien, int soGioLam)
{
        float heSo = 1.0f; // (1)
        if (loaiNhanVien == 1)
        {
                heSo = 1.5f; // (2)
                if (soGioLam > 40)
                        heSo = heSo + 0.2f; // (3)
        } else if (loaiNhanVien == 2)
                heSo = 1.2f; // (4)

        return 1200000 * soGioLam * heSo; // (5)
}
```

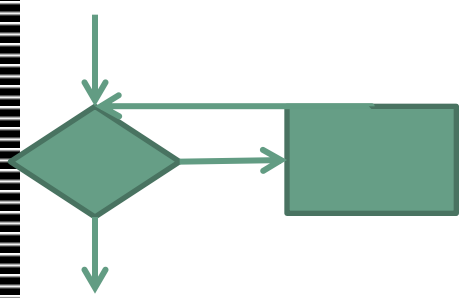
5.4. KIỂM THỬ LUỒNG DỮ LIỆU

- Chiến lược kiểm thử luồng dữ liệu
  - Mỗi **cặp DU** nên được thực thi ít nhất một lần.
  - Mỗi **đường DU** đơn giản (không chứa vòng lặp) nên được thực hiện ít nhất một lần.

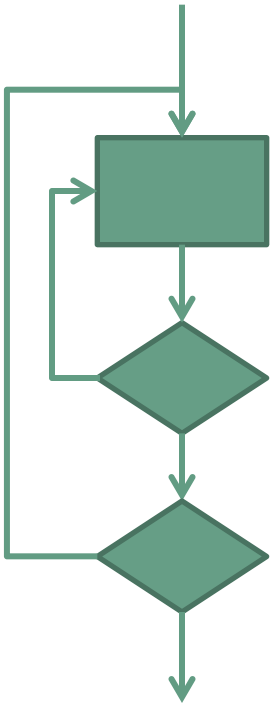
5.5. KIỂM THỬ VÒNG LẶP

- ✓ Kiểm tra **tính hợp lệ** của cấu trúc vòng lặp.
- ✓ Có 4 loại cấu trúc vòng lặp
  - Lặp đơn giản (simple loop)
  - Lặp lồng nhau (nested loop)
  - Lặp nối tiếp (concatenated loop)
  - Lặp không cấu trúc (unstructured loop)

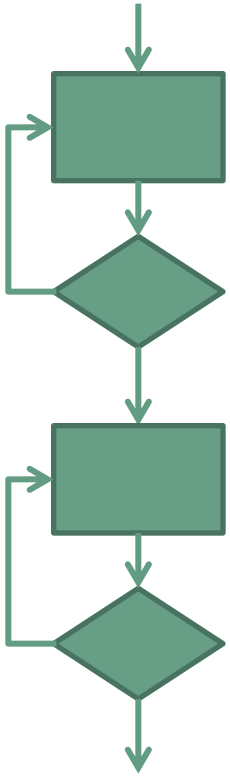
5.5. KIỂM THỬ VÒNG LẶP



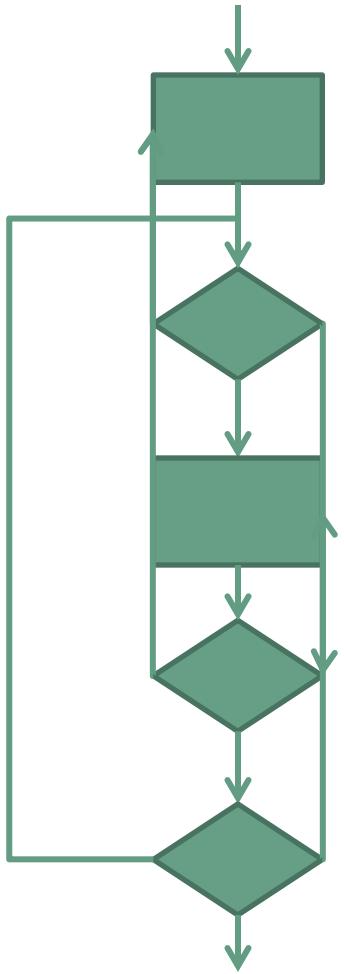
Lặp đơn giản



Lặp lồng nhau



Lặp nối tiếp



Lặp không cấu trúc

5.5. KIỂM THỬ VÒNG LẶP

- Khi kiểm thử vòng lặp cần thực hiện các thời điểm sau:
  - Lúc vừa vào vòng lặp.
  - Lúc đang xử lý trong vòng lặp.
  - Lúc rời khỏi vòng lặp.

5.5. KIỂM THỬ VÒNG LẶP

Lặp đơn giản

- Tập các test case sau cần được thực hiện cho các vòng lặp đơn giản, trong đó n là số tối đa vòng lặp có thể thực thi.
  - **Không** thực hiện lần lặp nào.
  - Thực hiện **1** lần lặp.
  - Thực hiện **2** lần lặp.
  - Thực hiện **m** lần lặp, trong đó m < n.
  - Thực hiện **n – 1**, **n**, **n + 1** lần lặp.

5.5. KIỂM THỬ VÒNG LẶP

Lặp lồng nhau

- Bắt đầu test vòng lặp **trong cùng** (innermost) và các vòng lặp khác thiết lập một giá trị tối thiểu.
- Dùng chiến lược kiểm thử vòng lặp đơn giản cho vòng lặp trong cùng và giữ các vòng lặp ngoài ở giá trị tối thiểu.
- Thực hiện tương tự cho các vòng lặp ngoài, cho đến khi vòng lặp ngoài cùng (outermost) được test.

5.5. KIỂM THỬ VÒNG LẶP

Lặp nối tiếp

- ✓ Nếu các vòng lặp là độc lập nhau thì áp dụng cách tiếp cận cho các vòng lặp đơn giản.
- ✓ Nếu các vòng lặp phụ thuộc nhau thì áp dụng cách tiếp nhận cho các vòng lặp lồng nhau.

5.5. KIỂM THỬ VÒNG LẶP

Lặp không cấu trúc

- Đối với trường hợp này nên yêu cầu thiết kế lại chương trình để đảm bảo tính cấu trúc của chương trình.

HUTECH
Đại học Công nghệ Tp.HCM

KIỂM THỬ PHẦN MỀM

Bài 5:
THIẾT KẾ TEST CASE WHITEBOX

Thời gian: 6 tiết

Q&A