Kiểm thử dòng dữ liệu

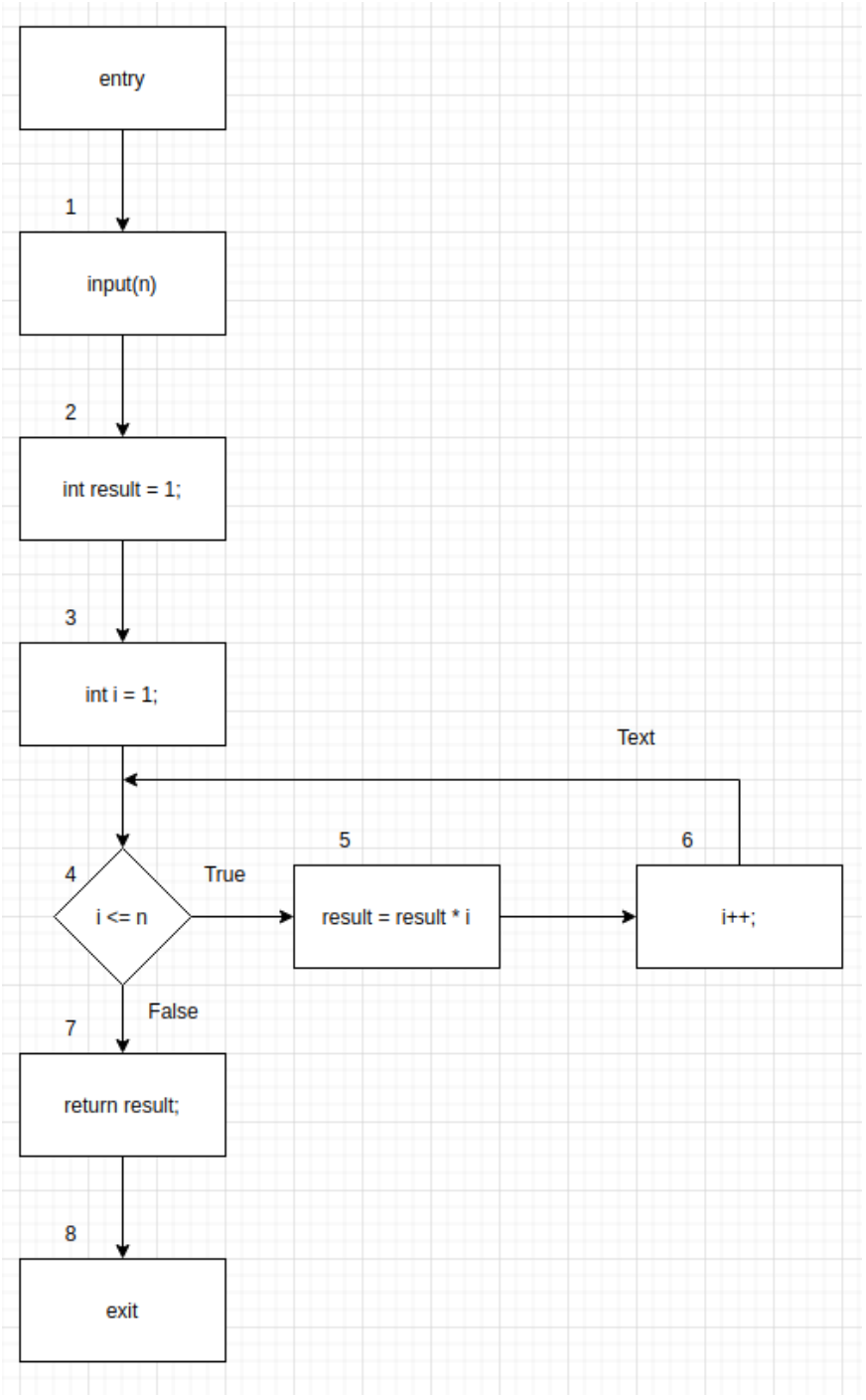
7. Cho hàm `calFactorial` viết bằng ngôn ngữ C như Đoạn mã 7.7.

   - Hãy liệt kê các câu lệnh ứng với các khái niệm *def*, *c−use*, và *p−use* ứng với các biến được sử dụng trong hàm này.
   - Hãy vẽ đồ thị dòng dữ liệu của hàm này.

Đoạn mã 7.7: Mã nguồn C của hàm `calFactorial`

```
int calFactorial (int n){
        int result = 1;
        int i=1;
        while (i <= n){
                result = result *i;
                i++;
        }//end while
        return result;
}//the end
```

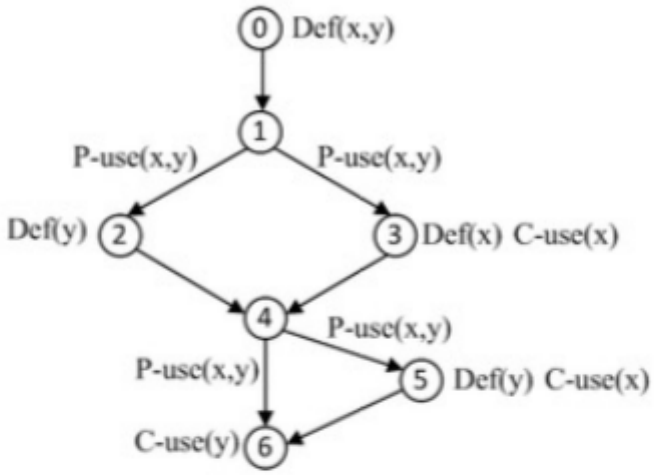
1. Đồ thị dòng dữ liệu



2. Các câu lệnh tương ứng:
    - Biến i:
        - Def(i): {3, 6}
        - c-use(i): {5, 6}
        - p-use(i): {4}

- Biến n:
    - Def(n): {1}
    - c-use(n): không có
    - p-use(n): {4}
- Biến result:
    - Def(result): {2, 5}
    - c-use(result): {5, 7}
    - p-use(result): không có

10. Cho đồ thị dòng dữ liệu như hình 7.11.



Hình 7.11: Một ví dụ về đồ thị dòng dữ liệu và việc sử dụng các biến.

- Hãy xác định tất cả các *Def-clear-path* của các biến x và y.
- Hãy xác định tất cả các du-paths của các biến x và y.
- Hãy xác định tất cả các *All-p-uses/Some-c-uses* và *All-c-uses/Some-p-uses* (dựa vào các chuẩn của kiểm thử dòng dữ liệu).
- Biểu thức của các *p-use*(x, y) tại cạnh (1,3) và (4,5) lần lượt là $x + y = 4$ và $x^2 + y^2 > 17$. Đường đi (0 - 1 - 3 - 4 - 5 - 6) có thực thi được không? Giải thích.
- Tại sao tại đỉnh 3 biến x được định nghĩa và sử dụng nhưng không tồn tại mối quan hệ def-use?

1. Các Def-clear-path của các biến:

- Biến x:
    - (0, 1); (0, 1, 3); (0, 1, 2); (0, 1, 2, 4); (0, 1, 2, 4, 6); (0, 1, 2, 4, 5); (0, 1, 2, 4, 5, 6)

    - (3, 4); (3, 4, 5); (3, 4, 5, 6); (3, 4, 6)
- Biến y:
    - (0, 1); (0, 1, 3); (0, 1, 3, 4); (0, 1, 3, 4, 6)
    - (2, 4); (2, 4, 6)
    - (5, 6)

2. Các du-paths của các biến:

- Biến x:
    - (0, 1); (0, 3); (0, 4); (0, 5)
    - (3, 4); (3, 5)
- Biến y:
    - (0, 1); (0, 4); (0, 6)
    - (2, 4); (2, 6)
    - (5, 6)

3. All-p-uses/some-c-uses và All-c-uses/some-p-uses:

Biến	All-p-uses/some-c-uses	All-c-uses/some-p-uses
x	0, 1(T)	(0, 3)
	0,1(F)	(0, 5)
	0, 4(T)	(3, 5)
	0, 4(F)	
	3, 4(T)	
	3, 4(F)	
y	0, 1(T)	0, 6
	0,1(F)	2, 6
	0, 4(T)	5, 6
	0, 4(F)	
	3, 4(T)	
	3, 4(F)	
	5,6	

4. Nếu biểu thức p-use(x,y) tại cạnh (1,3) và (4,5) lần lượt là x+y=4 và x2+y2>17 thì đường đi (0-1-3-4-5-6) có được thực thi. Tại vì:

- có x = 10; y = -6 thỏa mãn điều kiện

5. Đỉnh 3 biến x được định nghĩa và sử dụng nhưng không tồn tại mối quan hệ def-use bởi vì use(x) trước rồi mới def(x).

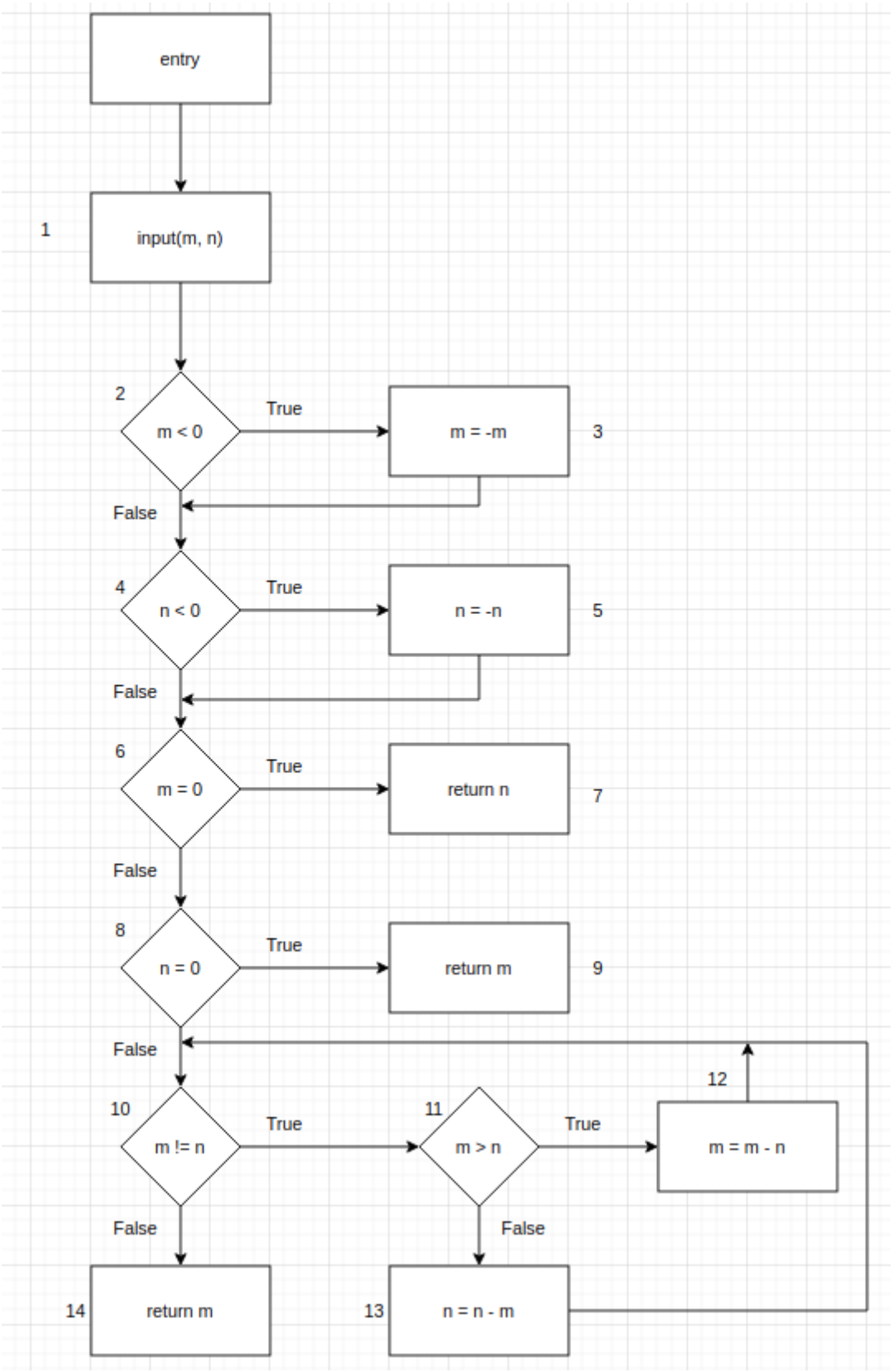
Cho đoạn mã nguồn như hình bên,

1. Xây dựng CFG cho hàm UCLN với đồ thị C2
2. Sinh đường đi và các ca kiểm thử với độ đo C2
3. Sinh đường đi và các ca kiểm thử với độ đo all-def coverage

Đoạn mã 6.4: Mã nguồn của hàm UCLN

```
int UCLN(int m, int n){
        if (m < 0) m = -m;
        if (n < 0) n = -n;
        if (m == 0)   return n;
        if (n == 0)   return m;
        while (m != n) {
                if(m > n)
                        m = m - n;
                else
                        n = n - m;
        }//end while
        return m;
}
```

1. CFG cho hàm UCLN



2. Đường đi và các ca kiểm thử với độ đo C2

STT	test path	test case
1	1,2(F),4(F), 6(F), 8(F), 10(T), 11(T), 12, 10(F), 14	m=6, n=6
2	1,2(T),3,4(T), 5, 6(F), 8(F), 10(F), 14	m=-8, n=-8
3	1,2(F),4(F), 6(T), 7	m=0, n=8
4	1,2(F),4(F), 6(F), 8(T), 9	m=8, n=0
5	1,2(F),4(F), 6(F), 8(F), 10(T), 11(F), 13, 10(F), 14	m=3, n=6

3. Đường đi và các ca kiểm thử độ đo all-def coverage
   def(m): 1, 3, 12
   def(n): 1, 5, 13

	Du-pair	Def-clear-path	Complete Path	Test case
m	1,3	1,2(T),3	1,2(T),3,4(T), 5, 6(F), 8(F), 10(F), 14	m=-8, n=-8
	3,6	3,4(T), 5, 6(F)	1,2(T),3,4(F), 5, 6(F), 8(T), 9	m=-8, n=0
	12,10	12, 10(F), 14	1,2(F),4(F), 6(F), 8(F), 10(T), 11(T), 12, 10(F), 14	m=8, n=4
n	1,5	1,2(T),3,4(T), 5	1,2(T),3,4(T), 5, 6(F), 8(F), 10(F), 14	m=-8, n=-8
	5,7	5, 6(T), 7	1,2(F),4(T),5, 6(T), 7	m=0, n=-8
	13,10	13,10	1,2(F),4(F), 6(F), 8(F), 10(T), 11(F), 13, 10(F), 14	m=4, n=8

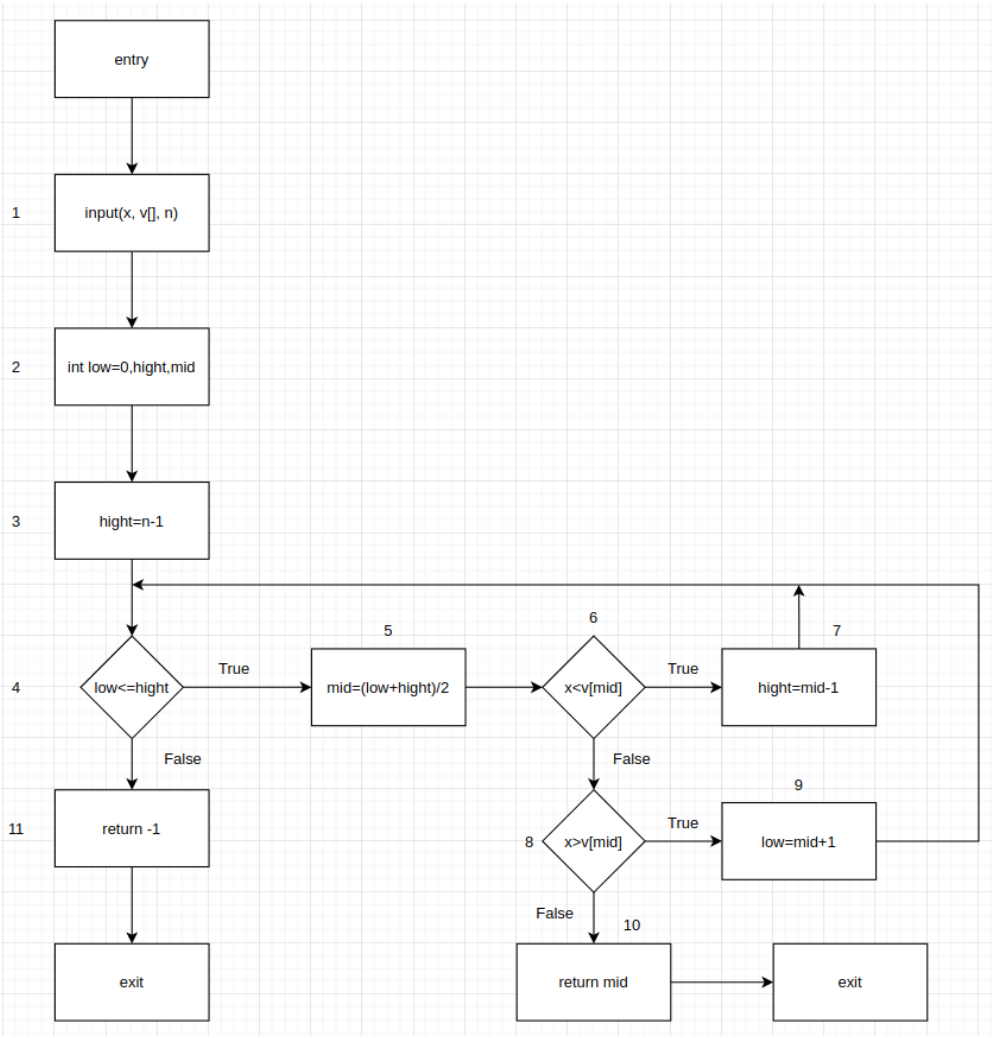
Cho đoạn mã 6.2. Hãy:

- Xây dựng đồ thị luồng điều khiển cho hàm BinSearch
- Sinh các đường đi và các ca kiểm thử với độ đo C2.
- Liệt kê các cặp du-pairs của tất cả các biến trong chương trình
- Sinh các đường đi và các ca kiểm thử với độ đo All-def cho biến **high**
- Sinh các đường đi và các ca kiểm thử với độ đo All-p-use cho biến **x**

Đoạn mã 6.2: Mã nguồn của hàm BinSearch

```
1  int Binsearch(int x, int v[], int n){
2      int low = 0, high, mid;
3      high = n - 1;
4      while (low <= high) {
5          mid = (low + high)/2;
6          if (x < v[mid])
7              high = mid - 1;
           else
8              if (x > v[mid])
9                  low = mid + 1;
               else
10                 return mid;
       }//end while
11     return -1;
   }//the end
```

1. Đồ thị hàm điều khiển



2. Đường đi và ca kiểm thử độ đo C2

STT	Test path	Test case
1	1,2,3,4(T),5,6(T),7,4(F),11	x=0,v=[8],n=1
2	1,2,3,4(T),5,6(F),8(T),9,4(F),11	x=9,v=[8],n=1
3	1,2,3,4(T),5,6(F),8(F),10	x=8,v=[8],n=1

3. Các cặp du-pairs của tất cả các biến

	Du-pair
x	1,6

	1,8
v	1,6
	1,8
n	1,3
low	2,4
	2,5
	9,4
	9,5
high	3,4
	3,5
	7,4
	7,5
mid	5,7
	5,9
	5,10

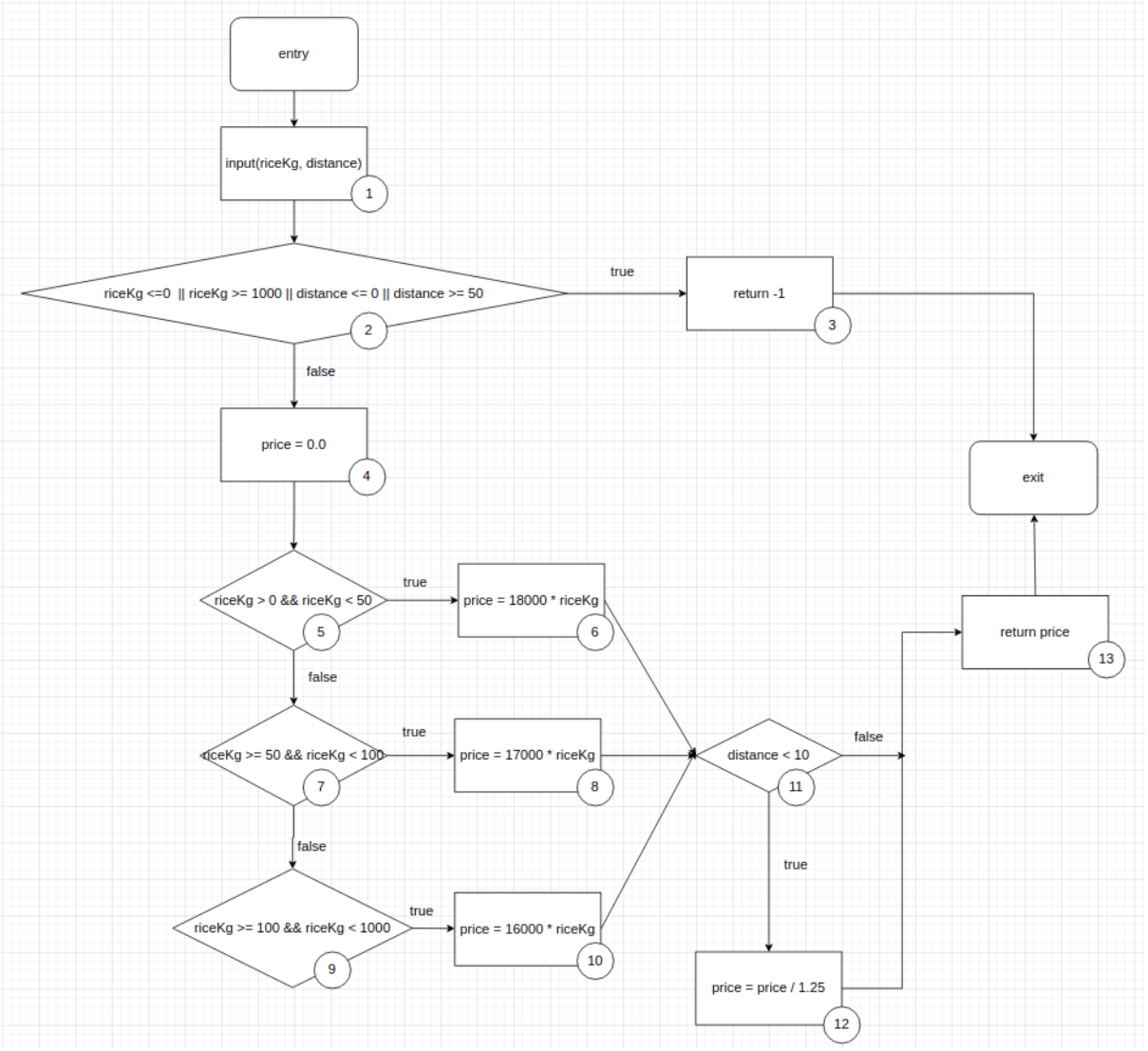
4. Đường đi và ca kiểm thử độ đo all-def cho biến high

	Du-pair	Def-clear-path	Complete Path	Test case
high	3,4	3,4	1,2,3,4(T),5,6(T),7,4(F),11	x=0,v=[8],n=1
	7,4	7,4	1,2,3,4(T),5,6(T),7,4(F),11	x=0,v=[8],n=1

5. Đường đi và ca kiểm thử độ đo all-p-use cho biến x

	Du-pair	Def-clear-path	Complete Path	Test case
x	1,6(F)	1,2,3,4(T),5,6(F)	1,2,3,4(T),5,6(F),8(T),9,4(F),11	x=9,v=[8],n=1
	1,6(T)	1,2,3,4(T),5,6(T)	1,2,3,4(T),5,6(T),7,4(F),11	x=0,v=[8],n=1
	1,8(F)	1,2,3,4(T),5,6(F),8(F)	1,2,3,4(T),5,6(F),8(F),10	x=8,v=[8],n=1
	1,8(T)	1,2,3,4(T),5,6(F),8(T)	1,2,3,4(T),5,6(F),8(T),9,4(F),11	x=9,v=[8],n=1

- Kiểm thử chương trình của bạn với độ phủ all-c-uses/some-p-uses



1. Đường đi và ca kiểm thử độ đo all-c-uses/some-p-uses:

	Du-pair	Def-clear-path	Complete path	test case	expected
riceKg	1,6	1, 2(F), 4, 5(T), 6	1, 2(F), 4, 5(T), 6, 11(F), 13	riceKg=10; distance=20;	180000
	1,8	1, 2(F), 4, 5(F), 7(T), 8	1, 2(F), 4, 5(F), 7(T), 8, 11(F), 13	riceKg=60; distance=20;	1020000
	1,10	1, 2(F), 4, 5(F), 7(F), 9(T), 10	1, 2(F), 4, 5(F), 7(F), 9(T), 10, 11(F), 13	riceKg=200;distance=20;	3200000
distance	1,11	1, 2(F), 4, 5(T), 6, 11	1, 2(F), 4, 5(T), 6, 11(F), 13	riceKg=10; distance=20;	180000
price	4,6	4, 5(T), 6	1, 2(F), 4, 5(T), 6, 11(F), 13	riceKg=10; distance=20;	180000
	4,8	4, 5(F), 7(T), 8	1, 2(F), 4, 5(F), 7(T), 8, 11(F), 13	riceKg=60; distance=20;	1020000
	4,10	4, 5(F), 7(F), 9(T), 10	1, 2(F), 4, 5(F), 7(F), 9(T), 10, 11(F), 13	riceKg=200;distance=5;	3200000
	4,12	4, 5(T), 6, 11(T), 12	1, 2(F), 4, 5(T), 6, 11(T), 12, 13	riceKg=10; distance=5;	144000
	4,13	4, 5(T), 6, 11(T), 12, 13	1, 2(F), 4, 5(T), 6, 11(T), 12,	riceKg=10; distance=5;	144000

			13		

2. Code:

```
class TestMethods(unittest.TestCase):

    def test1(self):
        self.assertEqual(get_price(10, 5), 144000)

    def test2(self):
        self.assertEqual(get_price(10, 20), 180000)

    def test3(self):
        self.assertEqual(get_price(60, 20), 1020000)

    def test4(self):
        self.assertEqual(get_price(200, 20), 3200000)
```