BÁO CÁO KIỂM THỬ PHẦN MỀM – KIỂM THỬ DÒNG DỮ LIỆU

Họ và tên: Hoàng Quốc Bảo

MSSV: 23020012

Bài 1 Các bước trong quy trình kiểm thử dòng dữ liệu động

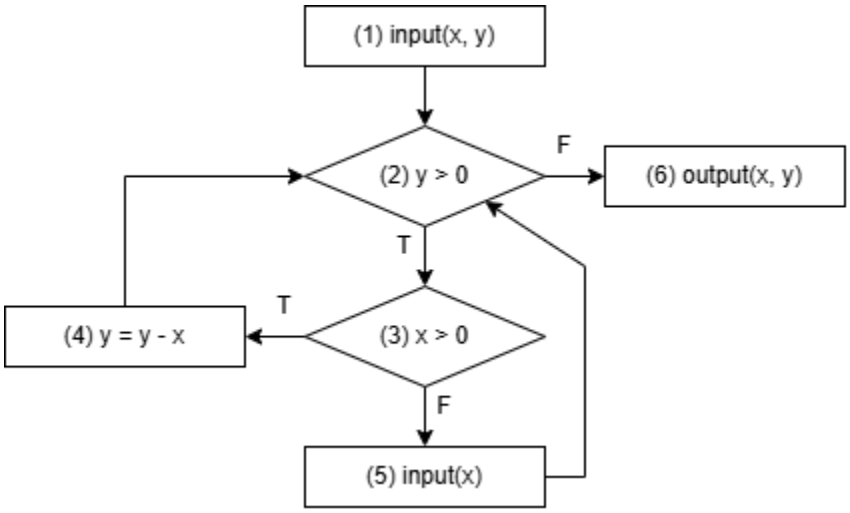
Quy trình tổng quát:
- Vẽ đồ thị luồng điều khiển CFG
- Lựa chọn tiêu chí kiểm thử dòng dữ liệu
- Xác định các đường đi trên CFG thỏa mãn tiêu chí kiểm thử đã chọn
- Sinh các ca kiểm thử tương ứng với các đường đi đã xác đinh

Bài 2

1) Mô tả mã nguồn:

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

2) Đồ thị dòng điều khiển (CFG):



3) Xác định def-use pairs (DU-pairs):
- Biến X:
  * def(x): 1, 5.
  * p-use(x): 3T, 3F
  * c-use(x): 4, 6

Du-pair	Def-clear path	Complete path
(1, 3T)	1-2(T)-3(T)	1-2(T)-3(T)-4-2(F)-6
(1, 3F)	1-2(T)-3(F)	1-2(T)-3(F)-5-2(T)-3(T)-4-2(F)-6
(1, 4)	1-2(T)-3(T)-4	1-2(T)-3(T)-4-2(F)-6
(1, 6)	1-2(T)-3(T)-4-2(F)-6	1-2(T)-3(T)-4-2(F)-6
(5, 3T)	5-2(T)-3(T)	1-2(T)-3(F)-5-2(T)-3(T)-4-2(F)-6
(5, 3F)	5-2(T)-3(F)	1-2(T)-3(F)-5-2(T)- 3(F)-5-2(T)-3(T)-4-2(F)-6
(5, 4)	5-2(T)-3(T)-4	1-2(T)-3(F)-5-2(T)-3(T)-4-2(F)-6
(5, 6)	5-2(T)-3(T)-4-2(F)-6	1-2(T)-3(F)-5-2(T)-3(T)-4-2(F)-6

- Biến Y:
 * def(y): 1, 4.2
 * p-use(y): 2T, 2F
 * c-use(y): 4.1, 6

Du-pair	Def-clear path	Complete path
(1, 2)	1-2(F)	1-2(F)-6
(1, 4.1)	1-2(T)-3(T)-4.1	1-2(T)-3(T)-4-2(F)-6
(1, 6)	1-2(F)-6	1-2(F)-6
(4.2, 2)	4.2-2(F)	1-2(T)-3(T)-4-2(F)-6
(4.2, 4.1)	4.2-2(T)-3(T)-4.1	1-2(T)-3(T)-4-2(T)-3(T)-4-2(F)-6
(4.2, 6)	4.2-2(F)-6	1-2(T)-3(T)-4-2(F)-6

4) Sinh đường đi và ca kiểm thử (mục tiêu: All-use coverage cho X và Y):
Chúng ta cần đảm bảo mọi DU-pair được bao phủ.
- Biến X:
+ complete path {1-2(T)-3(T)-4-2(F)-6}: Test case (x = 4, y = 3)
+ complete path {1-2(T)-3(F)-5-2(T)-3(T)-4-2(F)-6}: Test case {(x = -4, y = 4), (x = 5)}
+ complete path {1-2(T)-3(F)-5-2(T)- 3(F)-5-2(T)-3(T)-4-2(F)-6}: Test case {(x = -5, y = 8), (x = -5, x = 9)}

- Biến Y:
+ complete path {1-2(F)-6}: Test case (x = 2, y = -1)
+ complete path {1-2(T)-3(T)-4-2(F)-6}: Test case (x = 4, y = 3)
+ complete path {1-2(T)-3(T)-4-2(T)-3(T)-4-2(F)-6}: Test case (x = 5, y = 9)

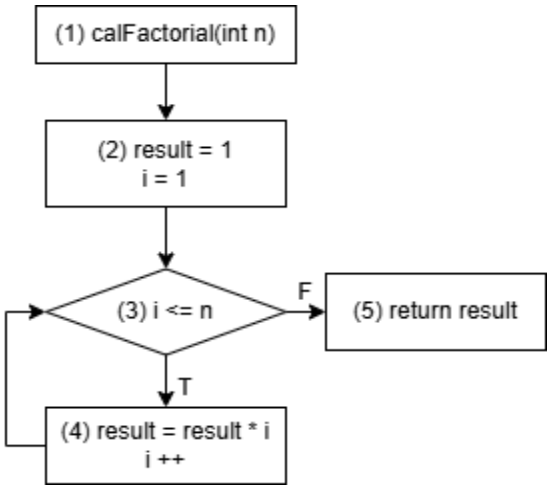
Bài 3

1) Mô tả mã nguồn

```
int calFactorial(int n) {
   int result = 1;
   int i = 1;
   while (i <= n) {
      result = result * i;
      i++;
```

```
    }
    return result;
}
```

2) Đồ thị dòng điều khiển (CFG):

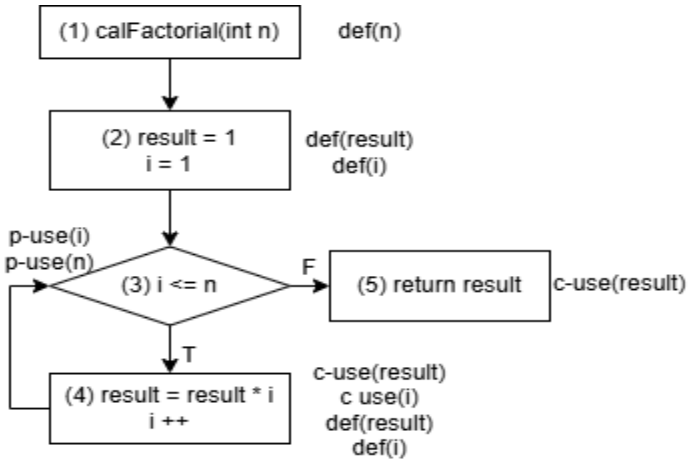


3) Xác định các câu lệnh def, c-use, p-use ứng với các biến:
- Biến result:
  * def(result): {result = 1, result = result * i}
  * p-use(result): không có
  * c-use(result): {result = result * i, return result}

- Biến i:
  * def(i): {i = 1, i ++}
  * p-use(i): while(i <= n)
  * c-use(i): {result = result * i, i ++}

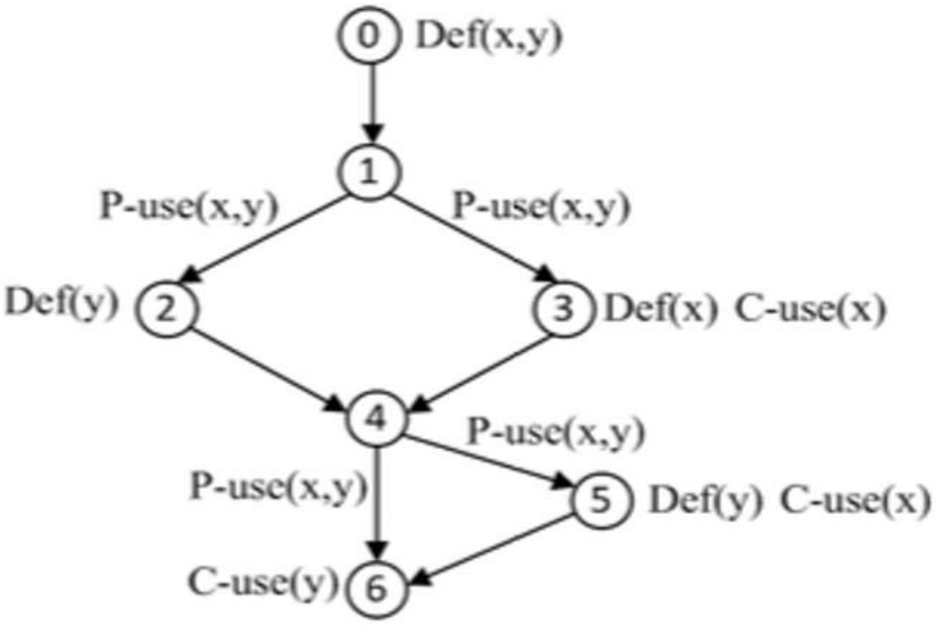
- Biến n:
  * def(n): calFactorial(int n)
  * p-use(i): while(i <= n)
  * c-use(i): không có

4) Đồ thị dòng dữ liệu:



Bài 4

1) Đồ thị:



2) Xác định Def-clear-path và DU-pairs (theo biến x và y):

- Giả thiết các vị trí def và use theo nhãn:
  Def(x) tại node0 và node3; Def(y) tại node0, node2, node5.

- Biến x:
  * def(x): {0, 3}
  * c-use(x): {3, 5}
  * p-use(x): {1L, 1R, 4L, 4R}

- Biến y:

* def(y): {0, 2, 5}
* c-use(y): {6}
* p-use(y): {1L, 1R, 4L, 4R}

Biến	Du pair	Def-clear path	Du path	All p-use /some c-use	All c-use /some p-use
x	(0, 1L)	0-1F	X	X	
	(0, 1R)	0-1T	X	X	
	(0, 3)	không có			X
	(0, 4L)	0-1L-2-4L	X	X	
	(0,4R)	0-1L-2-4R	X	X	
	(0, 5)	0-1L-2-4R-5	X		X
	(3, 1L)	Không có			
	(3, 1R)	Không có			
	(3, 3)	Không có			
	(3, 4L)	3-4L	X	X	
	(3-4R)	3-4R	X	X	
	(3, 5)	3-4R-5	X		X
y	(0, 1L)	0-1L	X	X	
	(0, 1R)	0-1R	X		
	(0, 4L)	0-1R-3-4L	X	X	
	(0, 4R)	0-1R-3-4R	X		
	(0, 6)	0-1R-3-4L-6	X		X
	(2, 1L)	Không có		X	
	(2, 1R)	Không có			
	(2, 4L)	2-4L	X	X	
	(2, 4R)	2-R	X		
	(2, 6)	2-4L-6	X		X
	(5, 1L)	Không có			
	(5, 1R)	Không có			
	(5, 4L)	Không có			
	(5, 4R)	Không có			
	(5, 6)	5-6	X	X	X

3) Biểu thức p-use(x,y) tại cạnh (1,3) và (4,5) lần lượt là: $x + y = 4$ và $x^2 + y^2 > 17$.
Thì đường đi (0-1-3-4-5-6) vẫn được thực hiện.
Bởi vì, không có ràng buộc x và y >= 0 nên x hoặc y có thể < 0. Vì vậy, có Test case (x = -4, y = 8) sẽ thỏa mãn điều kiện trên và đường đi được hoàn thành

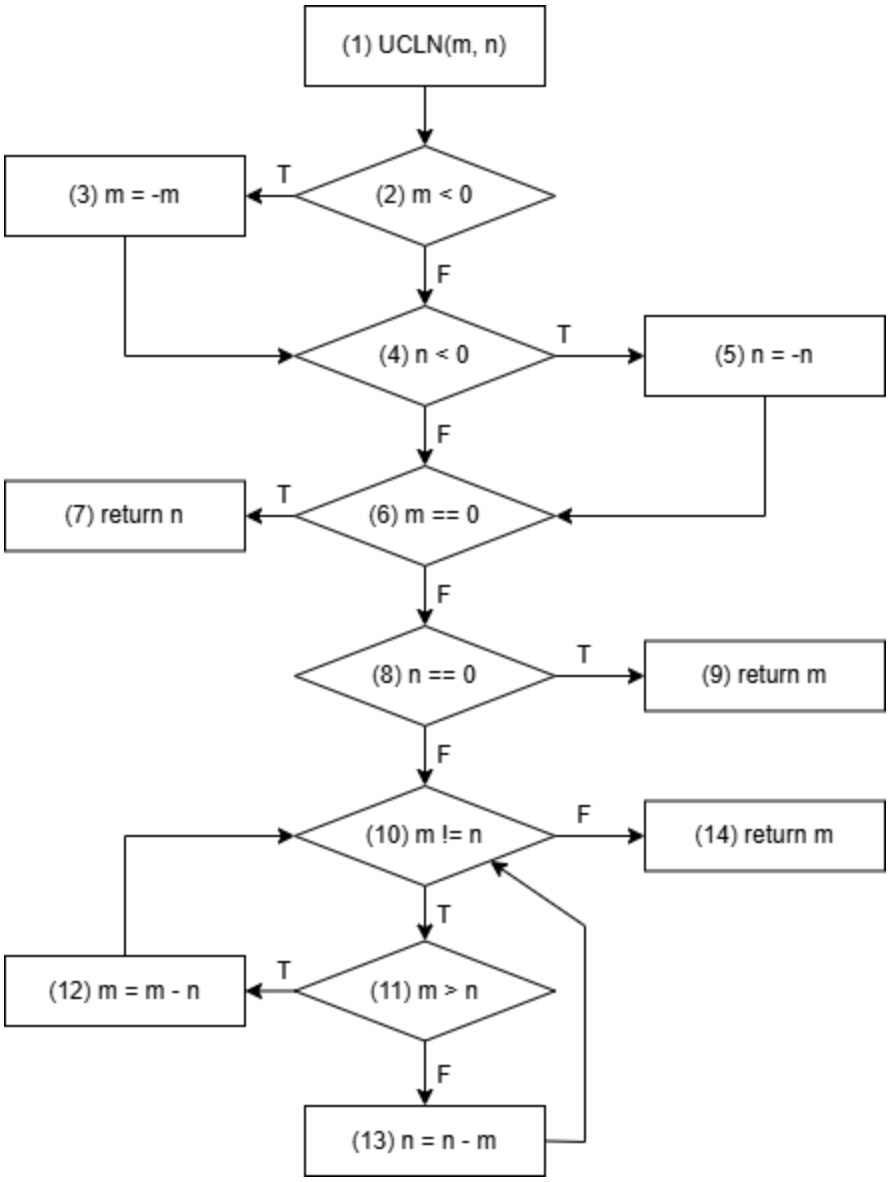
4) Tại đỉnh 3 biến x được định nghĩa và sử dụng nhưng không toàn tại mối quan hệ def-use.
Bởi vì, use(x) được thực thi trước def(x) nên không toàn tại def-clear path tại đỉnh 3 nên không tồn tại mối quan hệ def-use. Nhưng tại đỉnh 3 nếu def(x) được thực thi trước use(x) thì vẫn tồn tại mối quan hệ def-use, vì lúc đó tồn tại def-clear path của biến x.

Bài 5

1) Mô tả mã nguồn (Hàm UCLN - GCD):

```
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;
   }
   return m;
}
```

2) CFG cho hàm UCLN:



3) Xác định def-use pairs (DU-pairs) cho m và n:

- Biến m:
  * def(m): {1, 3, 12}
  * use(m): {2, 3, 6, 9, 10, 11, 12, 13, 14}
- Biến n:
  * def(n): {1, 5, 13}
  * use(n): {4, 5, 7, 8, 10, 11, 12, 13}

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

	(5, 7)	5-6T-7	1-2F-4T-5-6T-7	(m = 0, n =-5)
	(13-10)	13-10	1-2F-4F-6F-8F-10T-11F-13-10F-14	(m = 5, n = 10)

4) Sinh đường đi và ca kiểm thử theo độ đo C2 (decision coverage)
- Độ đo C2 yêu cầu mỗi nhánh (true/false) của các câu điều kiện phải được thực hiện ít nhất một lần. Từ đó ta cần chọn các giá trị đảm bảo cả hai phía của mỗi điều kiện đều được xét.
- Đường đi với kiểm thử độ đo C2

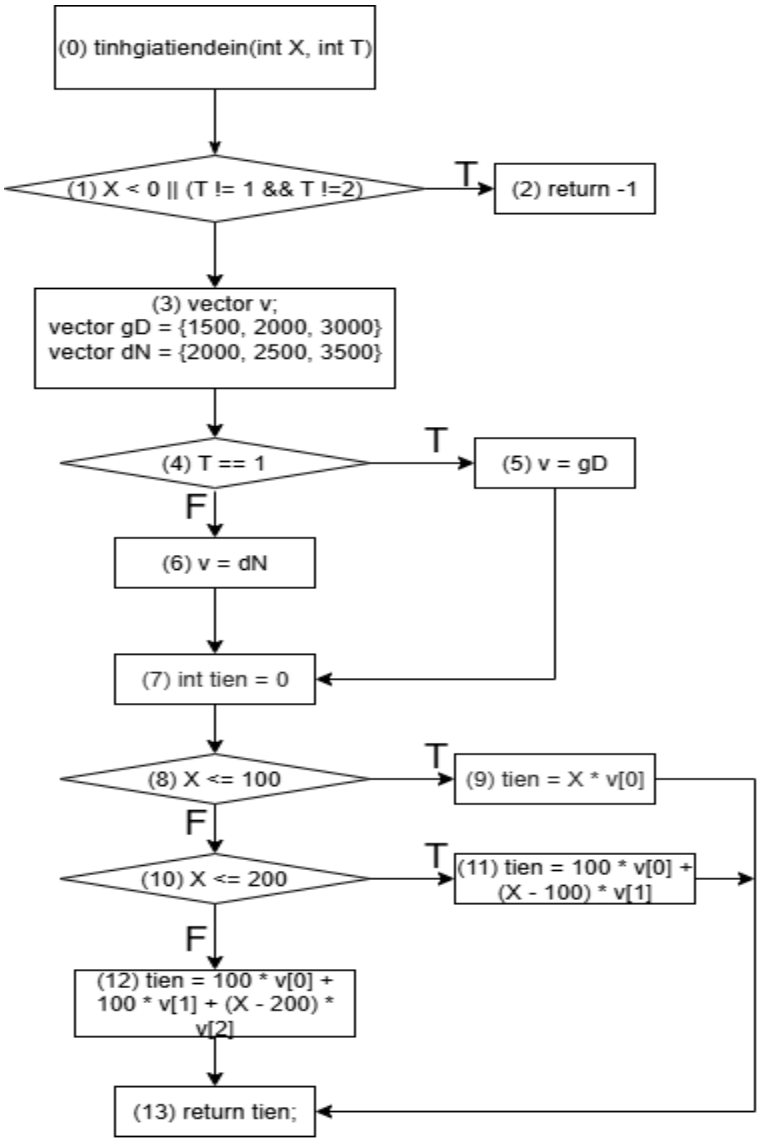
Đường đi	Test case
1-2F-4F-6T-7	(m = 0, n = 0)
1-2F-4F-6F-8T-9	(m = 1, n = 0)
1-2T-3-4T-5-6F-8F-10T-11F-13-10F-14	(m = -5, n = -10)
1-2F-4F-6F-8F-10T-11T- 12-10F-14	(m = -5, n = -10)

Bài 6. Phân tích và kiểm thử All-uses với hàm tinhGiaTienDien

1. Mô tả mã nguồn

```
int tinhGiaTienDien(int X, int T) {
   if ((X < 0) || (T != 1 && T != 2)) return -1;
   vector<int> v = T == 1 ? giaDinh : doanhNghiep;
   int tien = 0;
   if (X <= 100) tien = X * v[0];
   else if (X <= 200) tien = 100 * v[0] + (X - 100) * v[1];
   else tien = 100 * v[0] + 100 * v[1] + (X - 200) * v[2];
   return tien;
}
```

2. Đồ thị dòng điều khiển



- Biến X:
  * def(X): {0}
  * c-use(X): {9, 11, 12}
  * p-use(X): {1T, 1F, 8T, 8F, 10T, 10F}
- Biến T:
  * def(T): {0}
  * p-use(T): {1L, 1R, 4L, 4R}
- Biến v:
  * def(v): {5, 6}
  * use(v): {9, 11, 12}
- Biến giaDinh:
  * def(giaDinh): {3}
  * use(giaDinh): {5}
- Biến doanhNghiep:

* def(doanhNghiep): {3}
* use(doanhNghiep): {6}

- Biến tien:
* def(tien): {7, 9, 11, 12}
* use(tien): {13}

-> kiểm thử chương trình với độ phủ all-uses

Biến	Du pairs	Def-clear path	Complete path	Test case
X	(0, 1T)	0-1T	0-1T-2	(X=-5, T =0)
	(0, 1F)	0-1F	0-1F-3-4F-6-7-8T-9-13	(X=37, T = 2)
	(0, 8T)	0-1F-3-4F-6-7-8T	0-1F-3-4F-6-7-8T-9-13	(X=88, T =2)
	(0, 8F)	0-1F-3-4F-6-7-8F	0-1F-3-4F-6-7-8F-10T-11-13	(X=155, T =2)
	(0, 9)	0-1F-3-4F-6-7-8T-9	0-1F-3-4F-6-7-8T-9-13	(X=66, T =2)
	(0, 10T)	0-1F-3-4F-6-7-8F-10T	0-1F-3-4F-6-7-8F-10T-11-13	(X=155, T =2)
	(0, 10F)	0-1F-3-4F-6-7-8F-10F	0-1F-3-4F-6-7-8F-10F-12-13	(X=268, T =2)
	(0, 11)	0-1F-3-4F-6-7-8F-10T-11	0-1F-3-4F-6-7-8F-10T-11-13	(X=155, T =2)
	(0, 12)	0-1F-3-4F-6-7-8F-10F-12	0-1F-3-4F-6-7-8F-10F-12-13	(X=268, T =2)
T	(0, 1T)	0-1T	0-1T-2	(X=10, T =3)
	(0, 1F)	0-1F	0-1F-3-4F-6-7-8T-9-13	(X=37, T = 2)
	(0, 4T)	0-1F-3-4T	0-1F-3-4T-5-7-8T-9-13	(X=86, T =1)
	(0, 4F)	0-1F-3-4F	0-1F-3-4F-6-7-8T-9-13	(X=88, T =2)
v	(5, 9)	5-7-8T-9	0-1F-3-4T-5-7-8T-9-13	(X=86, T =1)
	(6, 9)	6-7-8T-9	0-1F-3-4F-6-7-8T-9-13	(X=56, T =2)
	(5, 11)	5-7-8F-10T-11	0-1F-3-4T-5-7-8F-10T-11-13	(X=149, T =1)
	(6, 11)	6-7-8F-10T-11	0-1F-3-4F-6-7-8F-10T-11-13	(X=167, T =2)
	(5, 12)	5-7-8F-10F-12	0-1F-3-4T-5-7-8F-10F-12-13	(X=350, T =1)
	(6, 12)	6-7-8F-10F-12	0-1F-3-4F-6-7-8F-10F-12-13	(X=500, T =2)
gia Dinh	(3, 5)	3-4T-5	0-1F-3-4T-5-7-8T-9-13	(X=0, T =1)
doanh Nghiep	(3, 6)	3-4F-6	0-1F-3-4F-6-7-8T-9-13	(X=99, T =2)

Link Github: https://github.com/bao2811/testing.git