**Lab#7 – White-box testing**

# วัตถุประสงค์การเรียนรู้

1. ผู้เรียนสามารถออกแบบการทดสอบแบบ White-box testing ได้
2. ผู้เรียนสามารถวิเคราะห์ปัญหาด้วย Control flow graph ได้
3. ผู้เรียนสามารถออกแบบกรณีทดสอบโดยคำนึงถึง Line coverage ได้
4. ผู้เรียนสามารถออกแบบกรณีทดสอบโดยคำนึงถึง Block coverage ได้
5. ผู้เรียนสามารถออกแบบกรณีทดสอบโดยคำนึงถึง Branch coverage ได้
6. ผู้เรียนสามารถออกแบบกรณีทดสอบโดยคำนึงถึง Condition coverage ได้
7. ผู้เรียนสามารถออกแบบกรณีทดสอบโดยคำนึงถึง Branch and Condition coverage ได้

# โจทย์: Clump counts

Clump counts (<https://codingbat.com/prob/p193817>) เป็นโปรแกรมที่ใช้ในการนับการเกาะกลุ่มกันของข้อมูลภายใน Array โดยการเกาะกลุ่มกันจะนับสมาชิกใน Array ที่อยู่ติดกันและมีค่าเดียวกันตั้งแต่สองตัวขึ้นไปเป็นหนึ่งกลุ่ม เช่น

[1, 2, 2, 3, 4, 4] → 2

[1, 1, 2, 1, 1] → 2

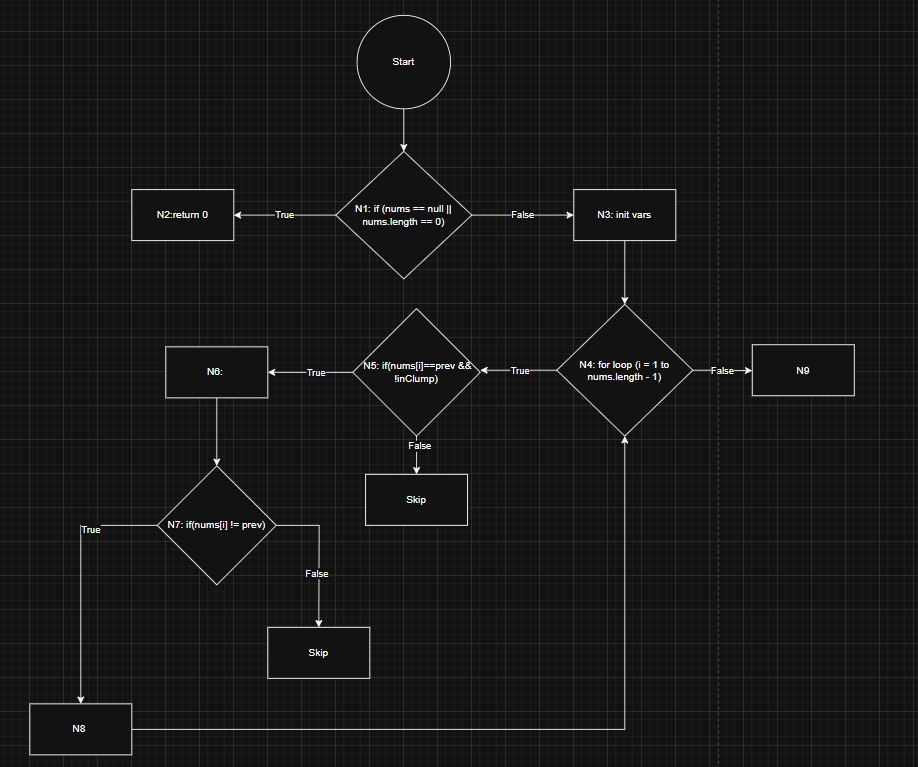
[1, 1, 1, 1, 1] → 1

ซอร์สโค้ดที่เขียนขึ้นเพื่อนับจำนวนกลุ่มของข้อมูลที่เกาะอยู่ด้วยกันอยู่ที่ <https://github.com/ChitsuthaCSKKU/SQA/tree/2025/Assignment/Lab7> โดยที่ nums เป็น Array ที่ใช้ในการสนับสนุนการนับกลุ่มของข้อมูล (Clump) ทำให้ nums เป็น Array ที่จะต้องไม่มีค่าเป็น Null และมีความยาวมากกว่า 0 เสมอ หาก nums ไม่เป็นไปตามเงื่อนไขที่กำหนดนี้ โปรแกรมจะ return ค่า 0 แทนการ return จำนวนกลุ่มของข้อมูล

# แบบฝึกปฏิบัติที่ 7.1 Control flow graph

จากโจทย์และ Source code ที่กำหนดให้ (CountWordClumps.java) ให้เขียน Control Flow Graph (CFG) ของเมธอด countClumps() จากนั้นให้ระบุ Branch และ Condition ทั้งหมดที่พบใน CFG ให้ครบถ้วน

ตอบ



Branch:

B1 จาก Node N1 ไป N2 `nums == null

B2 จาก Node N1 ไป N3 nums != null && nums.length > 0

B3 จาก Node N4 ไป N9 เงื่อนไข i < nums.length เป็นเท็จ (ลูปจบ)

B4 จาก Node N4 ไป N5 i < nums.length` เป็นจริง (ลูปทำงานต่อ)

B5 จาก Node N5 ไป N6 nums[i] == prev && !inClump เป็นจริง

B6 จาก Node N6 ไป N7 `nums[i] != prev

B7 จาก Node N7 ไป N8 nums[i] != prev เป็นจริง

B8 จาก Node N7 ไป N4 nums[i] == prev เป็นจริง

Condition:

C1 `nums == null

C2 i < nums.length (เงื่อนไขของ for loop)

C3 nums[i] == prev && !inClump (ตรวจว่าค่าและยังไม่ได้เริ่ม clump)

C4 nums[i] != prev (ตรวจว่าค่าเปลี่ยนแล้ว)

# แบบฝึกปฏิบัติที่ 7.2 Line Coverage

1. จาก Control Flow Graph (CFG) ของเมธอด countClumps() ในข้อที่ 1 ให้ออกแบบกรณีทดสอบเพื่อให้ได้ Line coverage = 100%
2. เขียนกรณีทดสอบที่ได้ พร้อมระบุบรรทัดที่ถูกตรวจสอบทั้งหมด
3. แสดงวิธีการคำนวณค่า Line coverage

ตอบ

| Test Case No. | Input(s) | Expected Result(s) | Path and Branch |
| --- | --- | --- | --- |
| TC1 | null | 0 | Line No.:[1], [2]  เข้าสู่เงื่อนไข nums == null → return |
| TC2 | {} | 0 | Line No.:[1], [2]  เข้าสู่เงื่อนไข nums.length == 0 → return 0 |
| TC3 | {1, 2, 2, 3} | 1 | Line No.:[1], [3]-[13]  (ยกเว้น [2])  ผ่านเงื่อนไข != null && length > 0, มี 1 clump (2,2), เข้า-ออกลูปครบ |
| TC4 | {1, 1, 2, 2, 2, 3} | 2 | Line No.:[1], [3]-[13]  (ยกเว้น [2])  หลาย clump (1,1 และ 2,2,2), เข้าเงื่อนไข clump ซ้ำหลายรอบ |

Line coverage =

TC1 Coverage: **2 / 13 = 15.38%**

TC2 Coverage: **2 / 13 = 15.38%**

TC3 Coverage: **12 / 13 = 92.31%**

TC4 Coverage: **12 / 13 = 92.31%**

# แบบฝึกปฏิบัติที่ 7.3 Block Coverage

1. จาก Control Flow Graph (CFG) ของเมธอด countClumps() ในข้อที่ 1 ให้ออกแบบกรณีทดสอบเพื่อให้ได้ Block coverage = 100%
2. เขียนกรณีทดสอบที่ได้ พร้อมระบุ Block ที่ถูกตรวจสอบทั้งหมด
3. แสดงวิธีการคำนวณค่า Block coverage

ตอบ

| Test Case No. | Input(s) | Expected Result(s) | Path and Branch |
| --- | --- | --- | --- |
| TC1 | null | 0 | Block: B1, B2 |
| TC2 | {} | 0 | Block: B1, B2 |
| TC3 | {1, 2, 2, 3} | 1 | Block: B1, B3-B6, B7-B9, B10-B12, B13, B14 |
| TC4 | {1, 1, 2, 2, 2, 3} | 2 | Block: B1, B3-B6, B7-B9, B10-B12, B13, B14 |

Block coverage = 14/14\*100 = 100%

# แบบฝึกปฏิบัติที่ 7.3 Branch Coverage

1. จาก Control Flow Graph (CFG) ของเมธอด countClumps() ในข้อที่ 1 ให้ออกแบบกรณีทดสอบเพื่อให้ได้ Branch coverage = 100%
2. เขียนกรณีทดสอบที่ได้ พร้อมระบุ Path และ Branch ที่ถูกตรวจสอบทั้งหมด
3. แสดงวิธีการคำนวณค่า Branch coverage

ตอบ

| Test Case No. | Input(s) | Expected Result(s) | Path and Branch |
| --- | --- | --- | --- |
| TC1 | null | 0 | Path: เข้าบล็อกตรวจสอบ null → return 0  Branch: B1=True |
| TC2 | {} | 0 | Path: เข้าบล็อกตรวจสอบ length=0 → return 0  Branch: B1=True |
| TC3 | {1, 2, 2, 3} | 1 | Path: ผ่าน if-null (False), เข้า loop 3 รอบ, เกิด clump 1 รอบ  Branch: B1=False, B3=True (loop start), B4=False (loop end), B5=True (clump), B6=False, B7=True (change prev), B8=False |
| TC4 | {1, 1, 2, 2, 2, 3} | 2 | Path:ผ่าน if-null (False), เข้า loop หลายรอบ, เกิด clump 2 รอบ  Branch: B1=False, B3=True, B4=False, B5=True, B6=False, B7=True, B8=False |

Branch coverage = (8 / 8) × 100 = 100%

# แบบฝึกปฏิบัติที่ 7.4 Condition Coverage

1. จาก Control Flow Graph (CFG) ของเมธอด countClumps() ในข้อที่ 1 ให้ออกแบบกรณีทดสอบเพื่อให้ได้ Condition coverage = 100%
2. เขียนกรณีทดสอบที่ได้ พร้อมระบุ Path และ Condition ที่ถูกตรวจสอบทั้งหมด เช่น Condition A = T และ Condition B = F
3. แสดงวิธีการคำนวณค่า Condition coverage

ตอบ

| Test Case No. | Input(s) | Expected Result(s) | Path and Condition |
| --- | --- | --- | --- |
| TC1 | null | 0 | Path: เข้าเงื่อนไข nums == null → return 0  Condition: A=True (nums==null), B=– (ไม่ต้องตรวจ)} |
| TC2 | {} | 0 | Path: nums != null แต่ nums.length == 0 → return 0  Condition: A=False, B=True |
| TC3 | {1, 2, 2, 3} | 1 | Path: nums != null && nums.length > 0, เข้า loop 3 รอบ  Condition: A=False, B=False,  รอบแรก: C=False, D=True  รอบสอง: C=True, D=True  รอบสาม: E=True |
| TC4 | {1, 1, 2, 2, 2, 3} | 2 | Path: nums != null && nums.length > 0, เข้า loop  Condition: A=False, B=False,  C=True/False, D=True/False, E=True/False |

Condition coverage = (10 / 10) × 100 = 100%

# แบบฝึกปฏิบัติที่ 7.5 Branch and Condition Coverage (C/DC coverage)

1. จาก Control Flow Graph (CFG) ของเมธอด countClumps() ในข้อที่ 1 ให้ออกแบบกรณีทดสอบให้ได้ C/DC coverage = 100%
2. เขียนกรณีทดสอบที่ได้ พร้อมระบุ Path, Branch, และ Condition ที่ถูกตรวจสอบทั้งหมด
3. แสดงวิธีการคำนวณค่า C/DC coverage
4. เขียนโค้ดสำหรับทดสอบตามกรณีทดสอบที่ออกแบบไว้ด้วย JUnit และบันทึกผลการทดสอบ

ตอบ

| Test Case No. | Input(s) | Expected Result(s) | Actual Result(s) | Path, Branch, and Condition |
| --- | --- | --- | --- | --- |
| TC1 | null | 0 | Pass/Fail: Pass | **Path:** B1=True → return  **Branch:** B1=True  **Condition:** A=True, B=– (ไม่ประเมิน) |
| TC2 | {} | 0 | Pass/Fail:Pass | |  | | --- |  | **Path:** B1=True → return  **Branch:** B1=True  **Condition:** A=False, B=True | | --- | |
| TC3 | {1, 2, 2, 3} | 1 | Pass/Fail:Pass | |  | | --- |  | **Path:** B1=False, B3=True, B5=True/False, B7=True/False, B4=False  **Branch:** B1=False, B3=True, B5(True/False), B7(True/False), B4(False)  **Condition:** A=False, B=False, C=True/False, D=True/False, E=True/False | | --- | |
| TC4 | {1, 1, 2, 2, 2, 3} | 2 | Pass/Fail:Pass | |  | | --- |  | **Path:** B1=False, B3=True (loopหลายรอบ), B5 True/False, B7 True/False, B4 False  **Branch:** ครบทุก branch  **Condition:** ครบทุก condition ทั้ง True และ False ในหลายรอบ | | --- | |

C/DC coverage = 100%