Rapport de Projet :Méthode de Couverture Tous Les Chemins Indépendants (TLCI)

Réalisé par :

Sahboub Manar
Afrit Mohamed Rayan

Groupe :

ILTI02

Professeur :

Mr. Amir Karim

Table des matières

1 Introduction

Les tests structurels, aussi appelés tests "boîte blanche", consistent à analyser le fonctionnement interne d'un programme, contrairement aux tests "boîte noire" qui se basent uniquement sur les entrées et sorties. Le but est de définir des cas de test à partir du graphe de flot de contrôle afin de vérifier que toutes les parties du code sont bien exécutées. Cette méthode permet de repérer des erreurs logiques que des tests classiques pourraient manquer.

2 Principes théoriques

2.1 Objectifs du test structurel

Les tests structurels ont pour objectifs :

- De vérifier la validité du flot de contrôle interne et de la logique de l'application.
- De détecter les branches non parcourues, les erreurs cachées et les boucles mal gérées.
- De compléter les tests fonctionnels pour augmenter la couverture du code.
- D'améliorer la robustesse et la fiabilité du logiciel.

2.2 Complexité cyclomatique

La complexité cyclomatique, introduite par McCabe en 1976, mesure la complexité d'un programme à partir de son graphe de flot de contrôle. Pour un graphe connexe à N nœuds et E arcs, elle est donnée par :

$$V(G) = E - N + 2$$

Cette métrique (également égale au nombre de régions du graphe) correspond au nombre minimal de chemins indépendants nécessaires pour couvrir toute la logique du programme.

2.3 Chemins indépendants

Un chemin indépendant est un chemin dans le graphe de flot de contrôle qui introduit au moins une nouvelle arête non couverte par d'autres chemins.

Identifier $V(G)$ chemins indépendants permet de définir une base de tests exhaustive sans parcourir tous les chemins possibles.

2.4 Position dans la hiérarchie des couvertures

On distingue plusieurs niveaux de couverture :

- Couverture des instructions (TLN) : chaque instruction exécutée au moins une fois.
- Couverture des branches (TLA) : chaque branche conditionnelle parcourue.
- Couverture des conditions multiples (MC/DC) : chaque condition influence individuellement l'issue d'une décision.
- Couverture des chemins indépendants (TLCI) : exécution de $V(G)$ chemins de base.
- Couverture exhaustive (tous les chemins réels) : irréaliste lorsque des boucles existent.

TLCI implique automatiquement TLN et TLA, tout en restant praticable contrairement à la couverture exhaustive.

3 Analyse du critère TLCI

3.1 Points forts

- Il permet de s'assurer que le code est exploré de manière aussi exhaustive que possible sur le plan logique.
- Il favorise l'identification rapide d'erreurs liées aux choix conditionnels ou aux structures en boucle.
- Il offre un indicateur utile de la complexité du logiciel, ce qui aide à organiser et dimensionner les efforts de test.
- Ce critère est particulièrement adapté aux systèmes à haut niveau d'exigence, comme ceux utilisés dans l'aéronautique, l'automobile ou le secteur médical.

3.2 Limites du critère

- Lorsqu'un programme devient plus complexe, le nombre de chemins indépendants à considérer peut rapidement devenir difficile à gérer.

— Les boucles dont on ne connaît pas à l'avance le nombre d'itérations compliquent l'analyse, car elles peuvent générer une infinité de parcours possibles.
— Certains chemins théoriques identifiés par la méthode peuvent, dans la pratique, ne jamais être empruntés lors de l'exécution.
— Enfin, le critère TLCI se limite à ce qui est effectivement codé, et ne permet pas de vérifier si des exigences importantes ont été oubliées dans le développement.

4 Secteurs d'utilisation

La couverture des chemins indépendants est privilégiée dans les domaines où la tolérance à l'erreur est minimale et où des normes strictes imposent un haut niveau de vérification :

— Dans l'aéronautique, conformément à la norme DO-178C et au critère MC/DC.
— Dans le secteur automobile, suivant les exigences de l'ISO 26262.
— Pour les dispositifs médicaux, selon la norme IEC 62304.

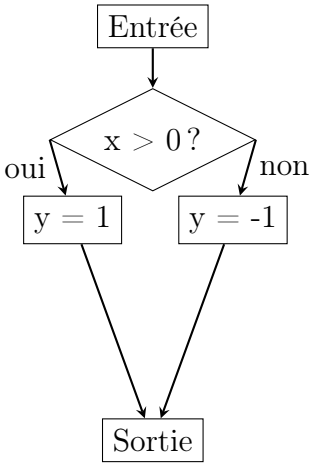
5 Études de cas

Chaque cas pratique ci-dessous comprend : un extrait de code, le graphe de flot associé, le calcul de la complexité cyclomatique $V(G)$, ainsi qu'une liste des chemins indépendants à envisager pour les tests.

5.1 Exemple 1 : If–Else simple

Code :

```
if (x > 0)
    y = 1;
else
    y = -1;
```



Graphe :

$V(G) = 2 - 1 + 2 = 3$, mais seulement 2 chemins indépendants sont nécessaires :

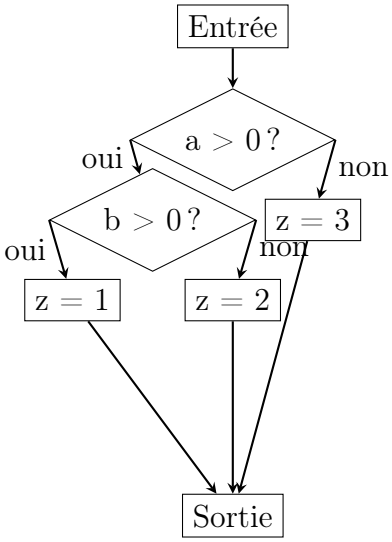
— C_1 : branche vraie (x>0).

— C_2 : branche fausse (x=0).

5.2 Exemple 2 : If imbriqué

Code :

```
if (a > 0) {
    if (b > 0)
        z = 1;
    else
        z = 2;
} else {
    z = 3;
}
```

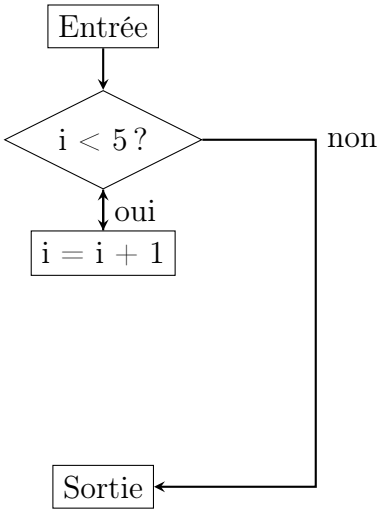


Graphe :
$V(G) = 3$ chemins indépendants :
— C_1 : a>0, b>0.
— C_2 : a>0, b<=0.
— C_3 : a<=0.

5.3 Exemple 3 : Boucle While simple

Code :

```
while (i < 5) {
    i = i + 1;
}
```



Graphe :

$$V(G) = E - N + 2 = 4 - 3 + 2 = 3$$

Chemins indépendants :

— C_1 : Entrée → i<5 (vrai) → i = i+1 → i<5 (faux) → Sortie.

— C_2 : Entrée → i<5 (vrai) → i = i+1 → i<5 (vrai) → boucle supplémentaire.

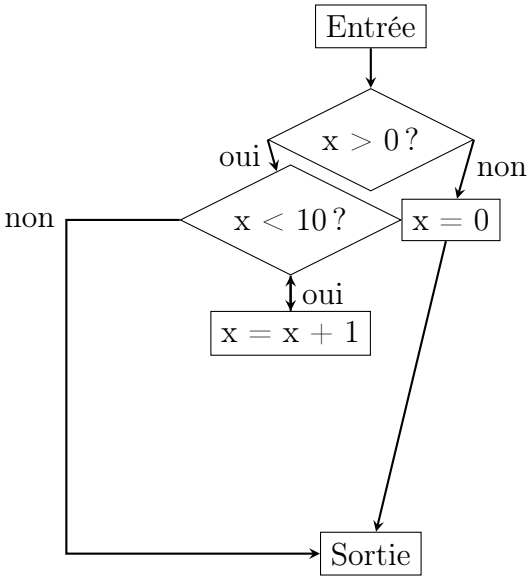
— C_3 : Entrée → i<5 (faux) → Sortie.

5.4 Exemple 4 : If–Else avec boucle

Code :

```
if (x > 0) {
    while (x < 10) {
        x = x + 1;
    }
} else {
    x = 0;
}
```

Graphe :



$$V(G) = E - N + 2 = 7 - 5 + 2 = 4$$

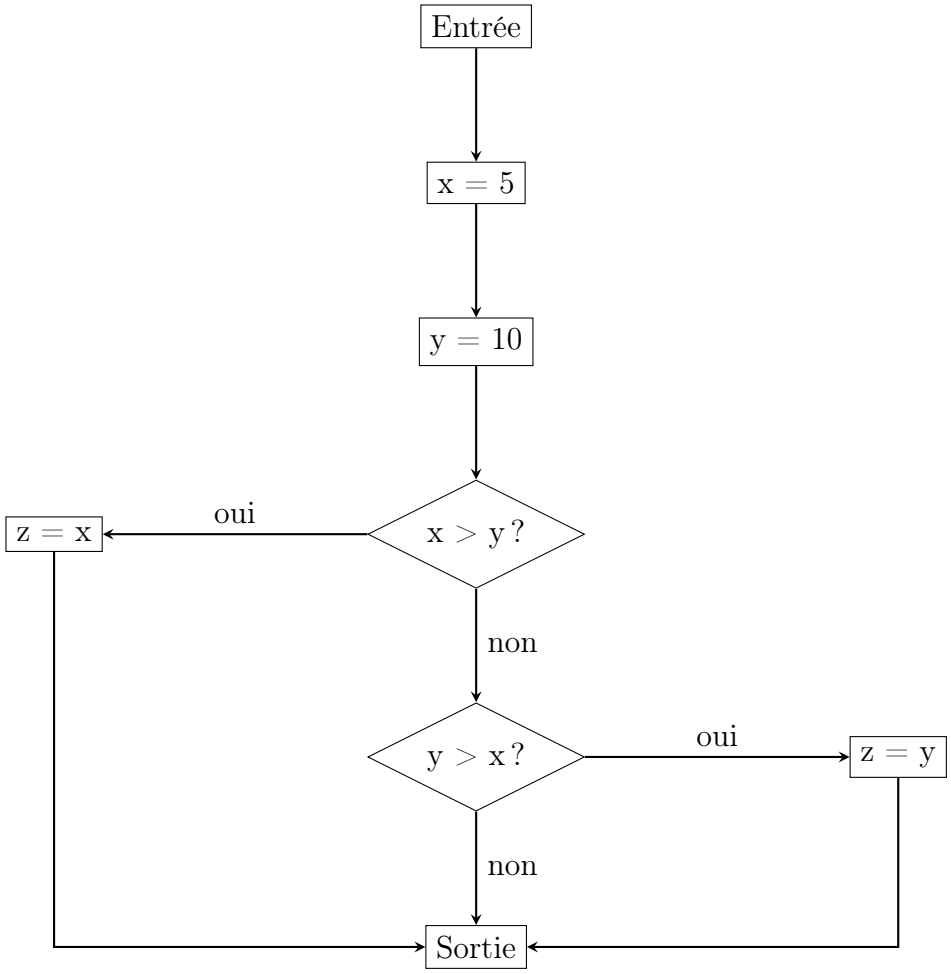
Chemins indépendants :
— C_1 : x>0, x<10 (vrai), itération unique, puis sortie.
— C_2 : x>0, x<10 (vrai), boucles multiples avant sortie.
— C_3 : x>0, x>=10 directement (pas d'itération).
— C_4 : x<=0, affectation x=0.

Exemple 5 : Séquence d'instructions et deux If

Code :

```
x = 5;
y = 10;
if (x > y)
    z = x;
if (y > x)
    z = y;
```

Graphe :



$$V(G) = E - N + 2 = 9 - 7 + 2 = 4$$

Chemins indépendants :

- C_1 : x>y (vrai), y>x (vrai).
- C_2 : x>y (vrai), y>x (faux).
- C_3 : x>y (faux), y>x (vrai).
- C_4 : x>y (faux), y>x (faux).

6 Conclusion

En résumé, la méthode TLCI permet d'obtenir une couverture du code très poussée, en s'appuyant sur la complexité cyclomatique. Elle est particu-

lièrement utile dans les domaines où les erreurs ne sont pas tolérées, comme l'aéronautique ou le médical. Même si elle demande plus d'efforts à cause du nombre élevé de chemins à tester, elle reste une méthode fiable pour améliorer la qualité des logiciels critiques.