TESTS UNITAIRES

2012-10-01

© Emmanuel Chieze
Département d'informatique, UQAM
INF3135

Plan

- ***Généralités***
- Tests en boîte blanche
- Tests en boîte noire

Tests unitaires

3

- 2 approches principales, complémentaires
  - boîte blanche
    - tests structurels
    - on teste l'implémentation d'un module
  - boîte noire
    - tests fonctionnels
    - on teste l'interface d'un module
      - Vérification du contrat de chaque fonction exportable
        - pré-conditions
        - post-conditions
        - exceptions
      - Sans se soucier de son implémentation

Tests unitaires

4

- Conception des tests
  - idéalement, conçus avant de coder la fonctionnalité
  - en réalité, interaction entre l'écriture du code et celle des tests
    - certains cas à problèmes n'apparaissent que lorsque l'on code
  - au pire, écriture des tests après celle du code
  - inacceptable : ne pas tester

Tests sous la forme d'un programme C

5

- convient particulièrement à des tests unitaires
- inclure un but *test* dans le makefile :
  - lance la compilation du programme de test
  - suivie de son exécution
- le but *test* n'est pas considéré par défaut

makefile

6

```
CC = gcc
CFLAGS = -g -W

Dictionnaires.o: Dictionnaires.c Dictionnaires.h

test : test_Dictionnaires
	test_Dictionnaires
test_Dictionnaires : test_Dictionnaires.o Dictionnaires.o
test_Dictionnaires.o : test_Dictionnaires.c Dictionnaires.h

clean :
	rm *.o test_Dictionnaires
```

Tests sous la forme d'un programme C

7

- Utilisation possible d'assertions
  - inconvénient : le programme de test s'arrête dès qu'une assertion est fausse
  - Exemple simple : fractions [Tremblay, 2005]
  - Exemple plus complexe : test_Dictionnaires.c (TP2)
    - OK si assertion ont des effets de bord dans programme de test
    - NOK dans application

Tests sous la forme d'un programme C

8

- Autre possibilité :
  - Remplacer les `assert(`*`condition`*`)` par des `if(`*`condition`*`) printf ("test…: ECHEC\n");`
  - avantage : le programme de test continue les tests même si une condition n'est pas respectée
  - toutefois : quand un test simple échoue, il n'est pas rare que des tests plus complexes échouent.

Plan

9

- Généralités
- ***Tests en boîte blanche***
- Tests en boîte noire

Tests en boîte blanche

10

- Justifications des tests structurels
  - plus de risque d'erreur associé au traitement de cas peu fréquents
    - donc à du code moins souvent parcouru
  - certaines portions de code peuvent être parcourues sans que l'on s'y attende
  - les portions de code peu parcourues sont autant susceptibles de contenir des erreurs de frappe que les autres portions
    - elles doivent être testées
  - on ne peut vérifier les structures de données internes en boîte noire

Niveaux de couverture

11

- Les tests structurels visent à garantir un certain niveau de couverture du code testé
  - couverture des instructions : insuffisant
    - que se passe-t-il lorsqu'un if n'a pas de else associé ?
  - couverture de toutes les enchaînements possibles entre instructions
    - garantit la couverture des instructions
  - couverture de tous les parcours possibles
    - correspond à des tests exhaustifs
    - garantit la couverture de tous les enchaînements possibles
    - impossible en cas de boucle

Principe des tests structurels

12

- Les tests structurels consistent à :
  - représenter le code en un graphe des parcours possibles des instructions du programme
  - déterminer le nombre minimal de tests permettant indirectement de couvrir tous les parcours possibles
  - implémenter les tests en question

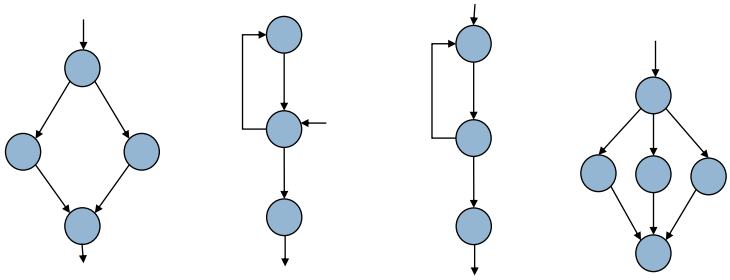
Graphes de flux de contrôle

13

- Chaque nœud représente une séquence d'instructions sans branchement
- Chaque arc représente un lien de contrôle entre 2 nœuds, soit la possibilité de se rendre d'un nœud à l'autre

Graphes élémentaires

14



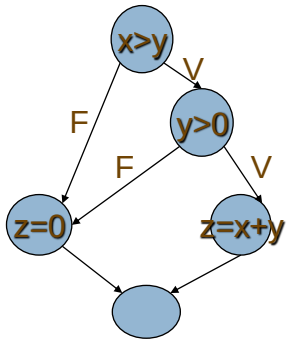
if-then-else **while** **do … while** **switch**

Graphes élémentaires

15

- Attention : dans les if, while et do-while, les conditions sont de simples comparaisons entre valeurs
- Exemple :

```
if (x>y && y>0)
   z = x + y;
else
   z = 0;
```

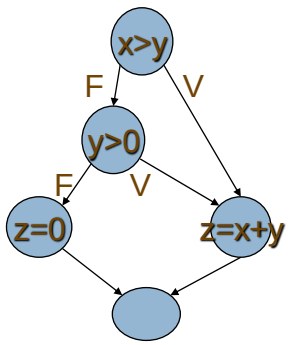


Graphes élémentaires

16

- Attention : dans les if, while et do-while, les conditions sont de simples comparaisons entre valeurs
- Exemple :

```
if (x>y || y>0)
   z = x + y;
else
   z = 0;
```



Graphes de fonctions

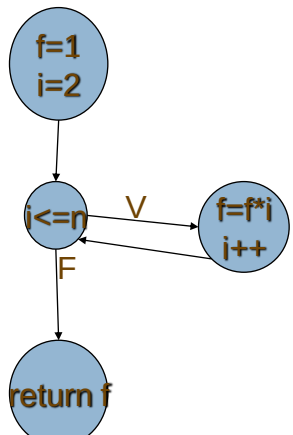
17

- Les graphes de fonctions doivent comporter un seul point d'entrée et un seul point de sortie
  - ajouter au besoin un nœud de sortie vide vers lequel tous les return de la fonction pointeront
  - cf. fonction toto (ci-après)

Exemple de graphe de flux de contrôle

18

```
int fact(int n) {
int i, f = 1;
for (i = 2; i <= n ; i++)
  f = f * i;
return f;
}
```



Complexité cyclomatique

19

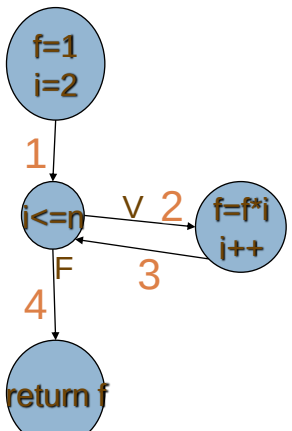
- Nombre minimal de chemins de base permettant de reconstituer tous les parcours possibles du graphe par combinaison linéaire des chemins de base
  - implique la couverture des arcs du graphe (donc de toutes les branches du programme)
  - implique la couverture des nœuds (donc de toutes les instructions du programme)
- Mesure de complexité d'un programme
- V(G) = #Arcs - #Noeuds + 2

Complexité cyclomatique

20

```
int fact(int n) {
  int f = 1, i;
  for (i = 2; i <= n ; i++)
      f = f * i;
  return f;
}
```

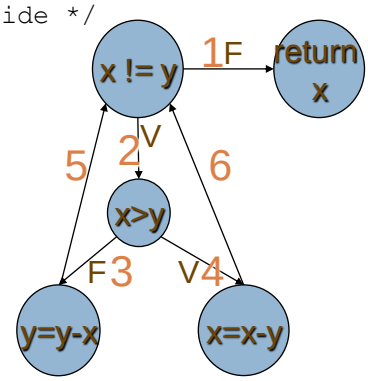
- V(G) = 4 - 4 + 2 = 2
- Chemins possibles : 1-4, 1-2-3-4
- Exemples : n = 1, n = 2



Complexité cyclomatique

21

```
/* pgcd par l'algorithme d'Euclide */
int pgcd(int x, int y) {
  while (x != y)
      if (x > y)
            x = x - y;
      else
            y = y - x;
  return x;
}
```



- V(G) = 6 - 5 + 2 = 3
- Chemins possibles : 1, 2-3-5-1, 2-4-6-1
- Exemples : x=3, y=3 ; x=3, y=6 ; x=6, y=3

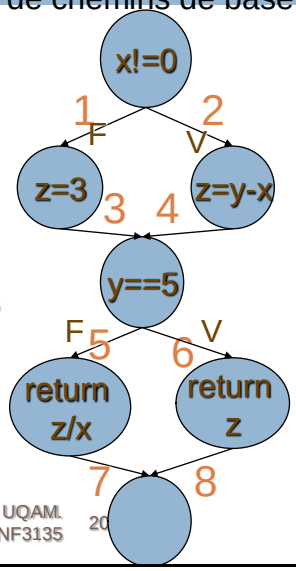
Complexité cyclomatique

22

- Il peut y avoir plusieurs ensembles de chemins de base distincts

```
int toto(int x, y) {
   int z;
   if (x != 0)
        z = y - x;
   else
        z = 3;
   if (y == 5)
        return z;
   else
        return z/x;
}
```

V(G) = 8 - 7 + 2 = 3



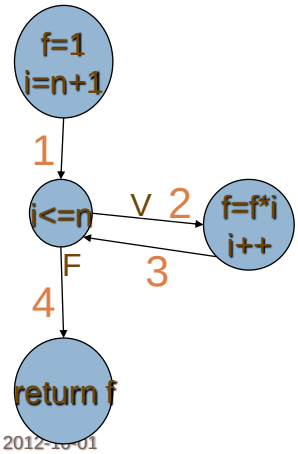
- Chemins possibles :
  - 1-3-6-8, 2-4-5-7, 2-4-6-8
    (x=0, y=5 ; x=2, y=3 ; x=2, y=5)
  - 1-3-6-8, 1-3-5-7, 2-4-6-8
    (x=0, y=5; x=0, y=4; x=2, y=5)

Complexité cyclomatique

23

```
int fact(int n) {
  int f = 1, i;
  for (i = n+1; i <= n ; i++)
      f = f * i;
  return f;
}
```

- V(G) = 4 - 4 + 2 = 2
- Chemins possibles : 1-4, 1-2-3-4
- Impossibles à mettre en oeuvre



Complexité cyclomatique

24

- Limites de la complexité cyclomatique
  - Basée sur la structure des graphes et non sur le contenu des nœuds
    - Certains tests peuvent être impossibles à mettre en œuvre
      - Généralement signe de problème dans le programme
      - Commandes et/ou tests inutiles
      - Assignations de valeurs aberrantes
    - Ne permet pas de détecter toutes les erreurs
      - il aurait fallu avoir un cas (x=0, y !=5) pour tester la division par zéro, donc un parcours 1-3-5-7, absent du premier ensemble de chemins de base

Complexité cyclomatique

25

- Limites de la complexité cyclomatique
  - Insuffisante également en cas de boucle (donc de graphe avec cycles)
    - Car une boucle peut être parcourue un nombre variable de fois
- Mais test de tous les parcours possibles généralement impossible

Tests complémentaires

26

- Palliatifs aux insuffisances de la complexité cyclomatique
  - Ajouter explicitement les cas à problème (x=0 dans le cas d'une division par x)
  - Boucles pouvant être parcourue de 0 à N fois, où N est contrôlé par l'environnement
    - tests aux limites
      - 0 fois
      - 1 fois
      - N-1 fois
      - N fois
      - N+1 fois
    - tests à l'intérieur de l'intervalle (cas courant)
      - N/2 fois par exemple

Tests complémentaires

27

- Palliatifs aux insuffisances de la complexité cyclomatique
  - Boucles imbriquées
    - on teste les boucles de l'intérieur vers l'extérieur
    - pour chaque boucle
      - effectuer le nombre minimal d'itérations des boucles qui lui sont externes,
      - effectuer les tests de la boucle testée comme indiqué précédemment
      - en exécutant un nombre typique de fois les boucles internes

Tests de boucles

28

- Quels sont les cas de tests à prévoir ici ?

```
void additionne_matrices(float a[][NBCOLMAX],
                         float b[][NBCOLMAX],
                         float c[][NBCOLMAX],
                         int nblig, int nbcol)
{
        int i, j;
        for (i=0; i<nblig; i++) {
                for (j=0; j<nbcol; j++)
                        c[i][j] = a[i][j] + b[i][j];
        }
}
```

Plan

29

- Généralités
- Tests en boîte blanche
- ***Tests en boîte noire***

Tests en boîte noire

30

- Complémentaires à ceux en boîte blanche
  - permettent de vérifier l'interface du module
  - basés sur les spécifications fonctionnelles du module

Tests en boîte noire

31

- Tests par génération de valeurs aléatoires
  - risquent de passer à côté des valeurs limites
  - ce sont ces valeurs qui posent problème le plus souvent
- Tests par partitionnement des domaines de valeurs des paramètres et des résultats en classes d'équivalence
  - Pour chacune des classes, prévoir
    - des tests aux limites
    - un test avec une valeur représentative de la classe

Partitionnement des domaines

32

- Exemple : calcul de 1/ln x
  - 4 classes de valeurs (2 invalides, 2 valides)
    - valeurs négatives de x (incluant 0)
    - valeurs de x comprises entre 0 et 1 (exclus)
    - 1
    - valeurs de x supérieure à 1
  - basées sur x mais correspondent également aux classes des résultats
  - tests sur -5, -0.01, 0, 0.01, 0.5, 0.99, 1, 1.01, 18, valeur max

Partitionnement des domaines

33

- Exemple : int fact (int n)
  - entiers codés sur d octets
  - 3 classes de valeurs (2 invalides, 1 valide)
    - valeurs négatives de n (excluant 0)
    - valeurs positives de n (incluant 0) telles que le résultat est codable sur d octets
    - valeurs positives de n telles que le résultat dépasse d octets (pour d = 4 octets, n >= 13)
  - sur 4 octets, tests sur -5, 0, 1, 8, 12, 13, 28

Partitionnement des domaines

34

- Exemple : triangle(a, b, c) indique si les longueurs (a,b,c) définissent un triangle, son type (isocèle, équilatéral, autre) et la nature de son angle le plus grand (aigu, droit obtus)
  - 8 classes de résultats
    - triangle équilatéral aigu (droit et obtus sont impossibles)
    - triangle isocèle aigu/droit/obtus
    - autre triangle aigu/droit/obtus
    - non triangle

Partitionnement des domaines

35

- valeurs limites pour les côtés
  - négatif
  - négatif proche de 0
  - 0
  - positif proche de 0
  - très grand
- valeurs limites pour les classes de résultats
  - point (0,0,0), presque triangle, triangle presque isocèle, triangle presque équilatéral, angle presque nul, angle presque droit

Partitionnement des domaines

36

- Valeurs limites
  - chaînes de caractères et vecteurs : de taille 0, de taille la taille maximale autorisée par la spécification du programme
  - valeurs définies par énumération : première et dernière valeurs de l'énumération

Référence

- [Tremblay, 2005] : Stratégie de tests (unitaires)