# Graphe de flot de contrôle

Production de graphe de contrôle
Production de données de test pour couvrir le graphe de contrôle

# Exercice 1

## Méthode nextForwardPosition

- Produire le graphe de contrôle de la fonction ci-dessous
- Produire des cas de test pour couvrir tous les nœuds du graphe

```
public static Coordinates nextForwardPosition(Coordinates position, Direction direction) {
        if (direction == NORTH)
            return new Coordinates(position.getX(), position.getY() - 1);
        if (direction == SOUTH)
            return new Coordinates(position.getX(), position.getY() + 1);
        if (direction == EAST)
            return new Coordinates(position.getX() + 1, position.getY());
        return new Coordinates(position.getX() - 1, position.getY());
}
```

```
public class Coordinates {

    private int x;
    private int y;

    public Coordinates(int x, int y) {
        this.x = x;
        this.y = y;
    }

…
```

# Correction

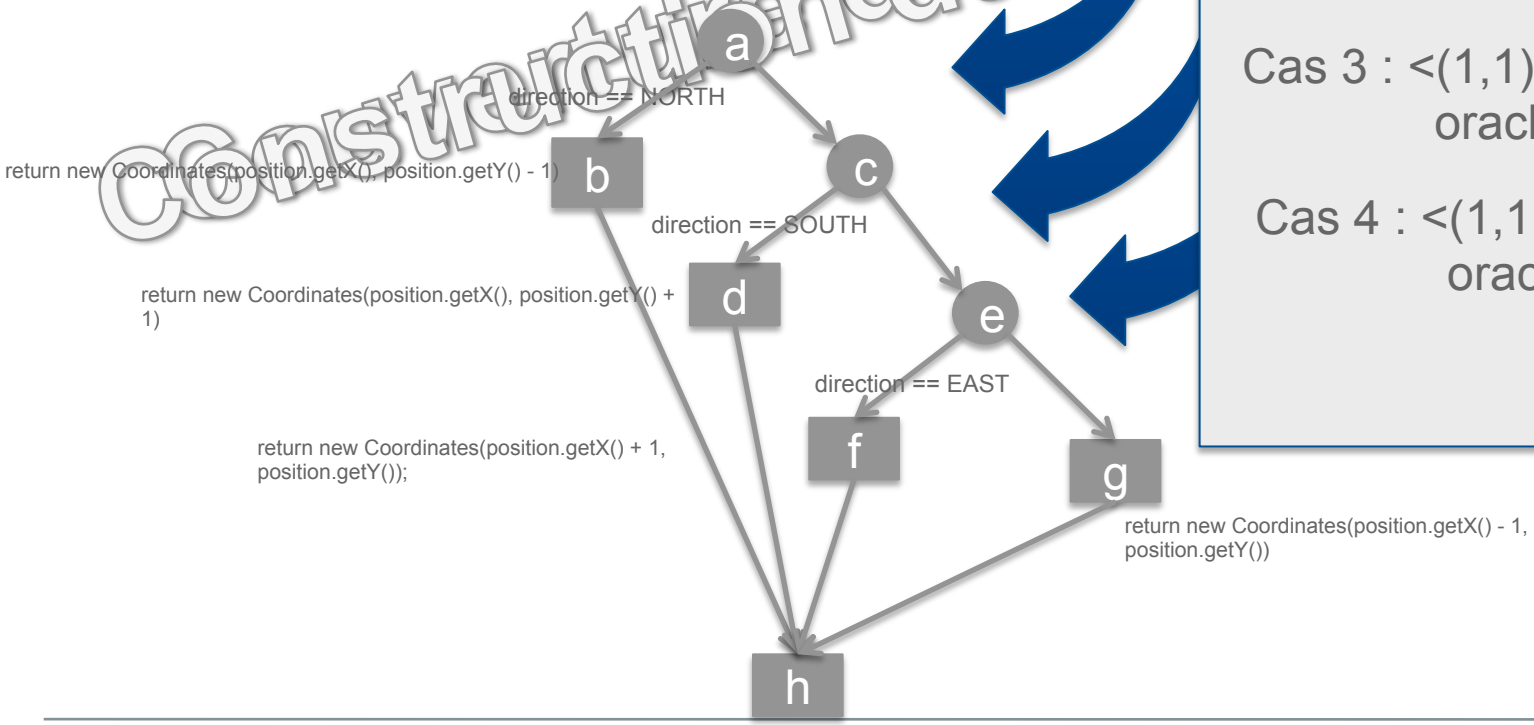
```
public static Coordinates nextForwardPosition(Coordinates position, Direction direction) {
    if (direction == NORTH)
        return new Coordinates(position.getX(), position.getY() - 1);
    if (direction == SOUTH)
        return new Coordinates(position.getX(), position.getY() + 1);
    if (direction == EAST)
        return new Coordinates(position.getX() + 1, position.getY());
    return new Coordinates(position.getX() - 1, position.getY());
}
```

Cas 1 : <(1,1), NORTH>
oracle : (1,0)

Cas 2 : <(1,1), SOUTH>
oracle : (1,2)

Cas 3 : <(1,1), EAST>
oracle : (2,1)

Cas 4 : <(1,1), WEST>
oracle : (0,1)



# Exercice 2

## Méthode letsGo

- Produire le graphe de contrôle de la fonction ci-dessous
- Produire des cas de test pour couvrir tous les arcs du graphe

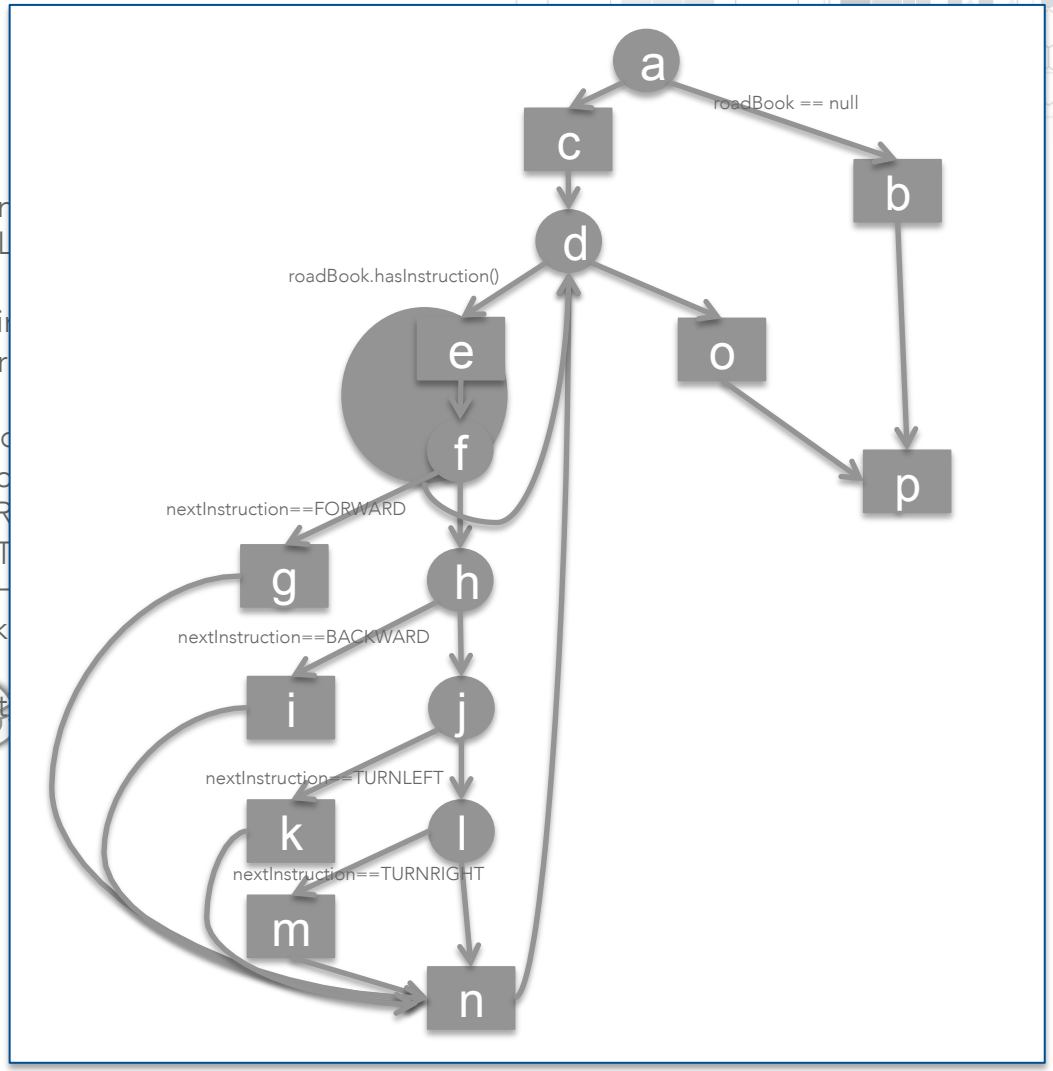
```
public List<CheckPoint> letsGo() throws UnlandedRobotException, UndefinedRoadbookException,
            InsufficientChargeException, LandSensorDefaillance, InaccessibleCoordinate {

    if (roadBook == null) throw new UndefinedRoadbookException();
    List<CheckPoint> mouchard = new ArrayList<CheckPoint>();
    while (roadBook.hasInstruction()) {
        Instruction nextInstruction = roadBook.next();
        if (nextInstruction == FORWARD) moveForward();
        else if (nextInstruction == BACKWARD) moveBackward();
        else if (nextInstruction == TURNLEFT) turnLeft();
        else if (nextInstruction == TURNRIGHT) turnRight();
        CheckPoint checkPoint = new CheckPoint(position, direction, false);
        mouchard.add(checkPoint);
        blackBox.addCheckPoint(checkPoint);
    }
    return mouchard;
}
```

```
class CheckPoint {

    public final Coordinates position;
    public final Direction direction;
    public final boolean manualDirective;

    public CheckPoint(Coordinates position, Direction direction, boolean manualDirective) {
        this.position = position;
        this.direction = direction;
        this.manualDirective = manualDirective;
    }

}

public enum Instruction {
    TURNLEFT,
    BACKWARD,
    TURNRIGHT,
    FORWARD
}
```

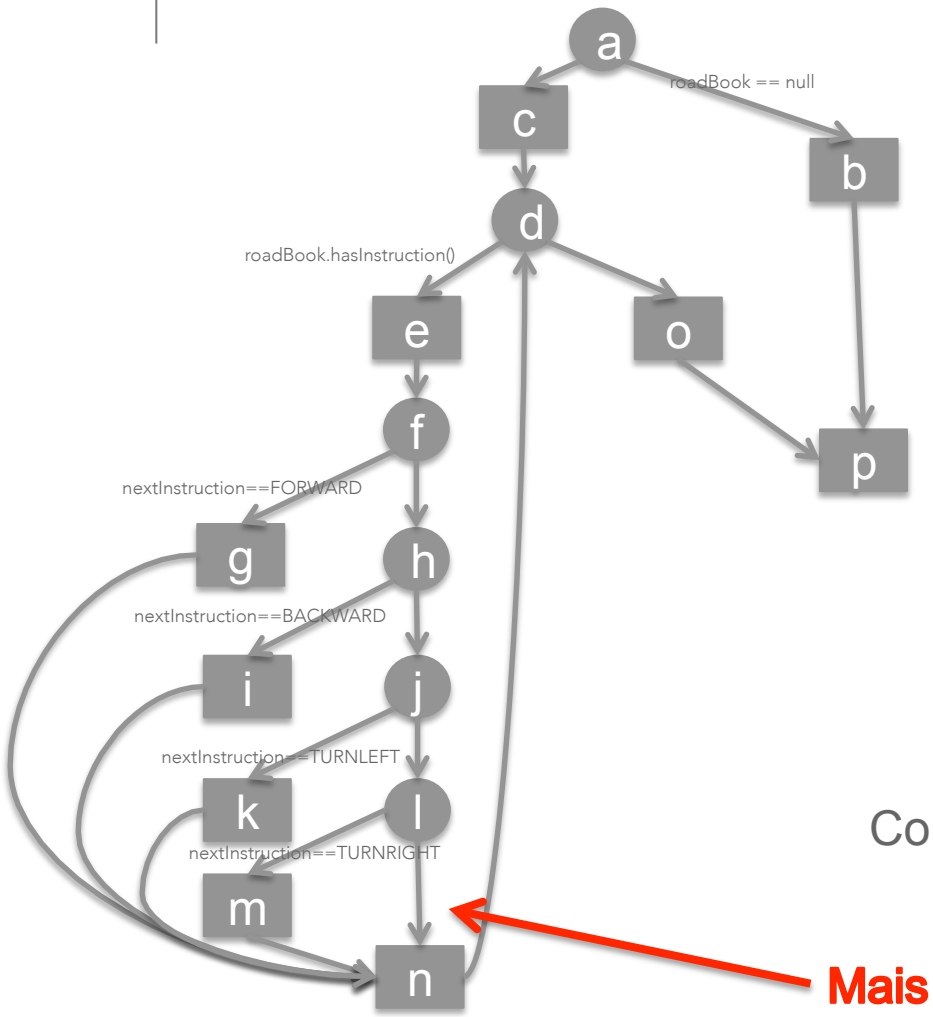
# Correction

```
public List<CheckPoint> letsGo() throws Ur
                InsufficientChargeException, L

    if (roadBook == null) throw new Undefi
    List<CheckPoint> mouchard = new Arr
    while (roadBook.hasInstruction()) {
        Instruction nextInstruction = roadBoo
        if (nextInstruction == FORWARD) mo
        else if (nextInstruction == BACKWAR
        else if (nextInstruction == TURNLEFT
        else if (nextInstruction == TURNRIGH
        CheckPoint checkPoint = new Check
        mouchard.add(checkPoint);
        blackBox.addCheckPoint(checkPoint
    }
    return mouchard;
}
```

Couverture



# Correction



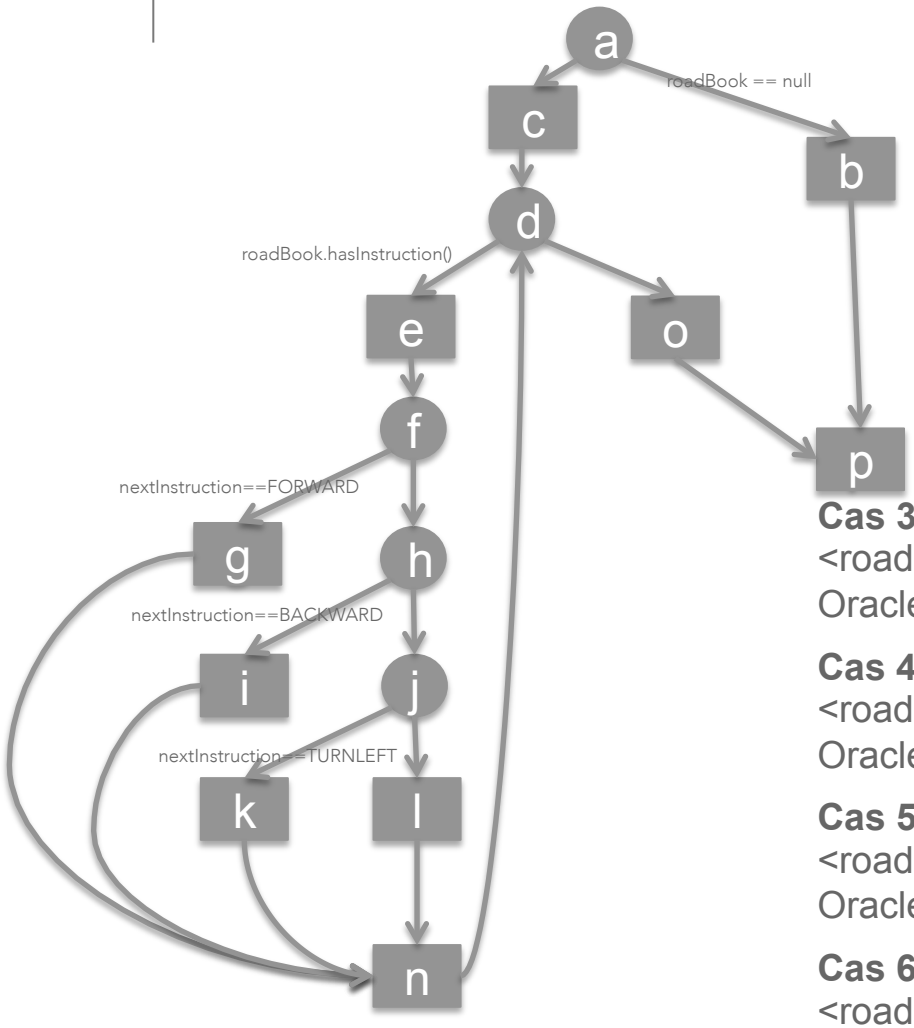

Cas 1 :
<roadBook = null, position = (0,0), direction = NORTH>
Oracle : UndefinedRoadBookException

Cas 2 :
<roadBook = [FORWARD, TURNRIGHT, BACKWARD, TURNLEFT],
position = (0,0), direction = NORTH>
Oracle : [<(0,-1),NORTH,false>, <(0,-1),EAST,false>,
<(-1,-1),EAST,false>, <(-1,-1),NORTH,false

Couverture tous nœuds obtenue

**Mais pas tous arcs**

# Extension –tous chemins indépendants




Mc Cabe : 6

**Cas 1** :
<roadBook = null, position = (0,0), direction = NORTH>
Oracle : UndefinedRoadBookException

**Cas 2** :
<roadBook = [], position = (0,0), direction = NORTH>
Oracle : []

**Cas 3** :
<roadBook = [FORWARD], position = (0,0), direction = NORTH>
Oracle : [<(0,-1),NORTH,false>]

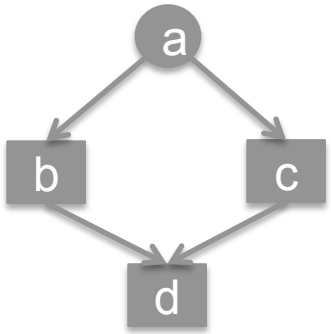
**Cas 4** :
<roadBook = [BACKWARD], position = (0,0), direction = NORTH>
Oracle : [<(0,1),NORTH,false>]

**Cas 5** :
<roadBook = [TURNLEFT], position = (0,0), direction = NORTH>
Oracle : [<(0,0),EAST,false>]

**Cas 6** :
<roadBook = [TURNRIGHT], position = (0,0), direction = NORTH>
Oracle : [<(0,0),WEST,false>]

# Choisir la couverture

```
if (x>y)
    max=x;
else
    max=y;
return max
```

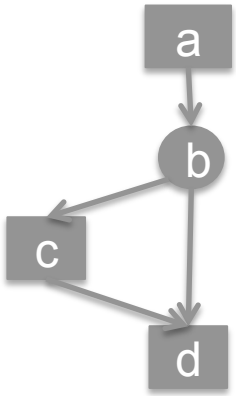



x=3, y=2
Oracle 3

x=1, y=2
Oracle 2

tous nœuds

```
read(x, y);
if (x<0)
    y=-y;
return y/x;
```

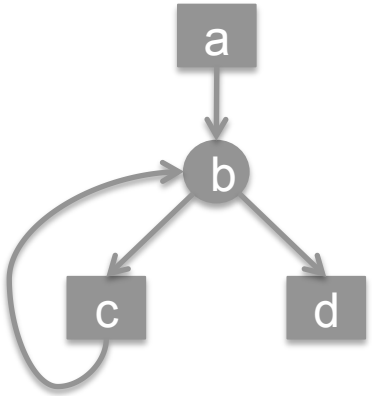



x=-2, y=4
Oracle 2

x=2, y=4
Oracle 2

tous arcs

```
read(x, y);
i=0;
while(i<x) {
    y=2*y;
    i++;
}
return y/x;
```




x=-2, y=4
Oracle 2

x=2, y=4
Oracle 2

chemins

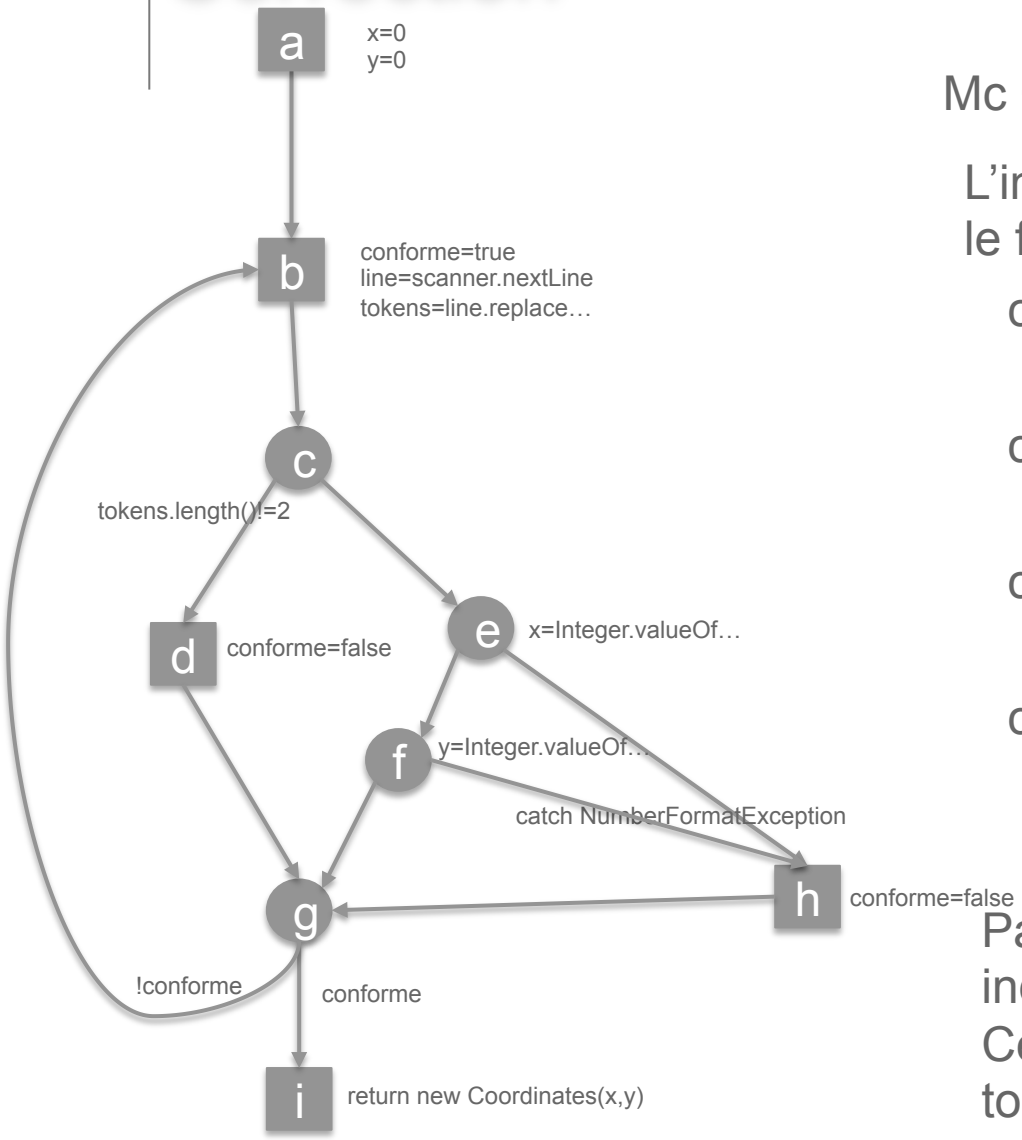
# Exercice 3

## Méthode lireCoordonnee

- Produire le graphe de contrôle de la fonction ci-dessous
- Produire des cas de test pour couvrir tous chemins indépendants

```
static Coordinates lireCoordonnee(Scanner scanner) {
    boolean conforme;
    int x = 0;
    int y = 0;
    do {
        conforme = true;
        String line = scanner.nextLine();
        String[] tokens = line.replace("(", " ").replace(")", " ").split(",");
        if (tokens.length != 2) {
            conforme = false;
            System.out.println("Format incorrect. c, l ou (c, l)");
        }
        else
            try {
                x = Integer.valueOf(tokens[0].trim());
                y = Integer.valueOf(tokens[1].trim());
            } catch (NumberFormatException e) {
                conforme = false;
            }
    } while (!conforme);
    return new Coordinates(x, y);
}
```

# Correction




Mc Cabe : 5

L'input du programme est le flux d'entrée standard

cas1 : (5,6)\n
couvre abcefgi

cas2 : x,2\n (5,6)\n
couvre abcehgbcefgi

cas3 : 1,y\n (5,6)\n
couvre abcefhgbcefgi

cas4 : (5,3,4)\n (5,6)\n
couvre abcdgbcefgi

Pas de 5° chemin linérairement indépendant et exécutable.
Ces 4 chemins permettent d'exprimer tous les chemins exécutable par combinaison

# Correction

cas1 : (5,6)\n
couvre abcefgi

cas2 : x,2\n (5,6)\n
couvre abcehgbcefgi

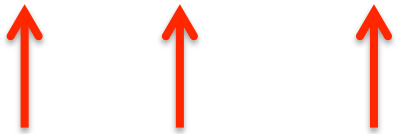
cas3 : 1,y\n (5,6)\n
couvre abcefhgbcefgi

cas4 : (5,3,4)\n (5,6)\n
couvre abcdgbcefgi

$$
\begin{aligned}
x1 + x2 + x3 + x4 &= 0\\
x1 + 2x2 + 2x3 + 2x4 &= 0\\
x4 &= 0 \quad \leftarrow\\
x1 + 2x2 + 2x3 + x4 &= 0\\
x1 + x2 + 2x3 + x4 &= 0\\
x2 &= 0 \quad \Leftarrow\\
x3 &= 0 \quad \Leftarrow\\
x2 + x3 + x4 &= 0\\
x2 + x3 &= 0
\end{aligned}
$$

Pour montrer que les 4 chemins sont linéairement indépendants, il faut établir les vecteurs des chemins en fonction de la base des arcs (aka nombre de fois que le chemin emprunte chaque arc). Puis établir un système d'équations ou pour chaque arc de la base on établit une équation à 4 inconnues dont les coefficients sont le nombre de fois que l'arc est emprunté. Si le système admet comme seule solution 0 pour toutes les inconnues, les chemins sont linéairement indépendants.

| ab | bc | cd | ce | dg | ef | eh | fg | fh | gb | gi | hg |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 2 | 0 | 2 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 2 | 0 | 2 | 0 | 2 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 2 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |



# Correction

cas1 : (5,6)\n
couvre abcefgi

cas2 : x,2\n (5,6)\n
couvre abcehgbcefgi

cas3 : 1,y\n (5,6)\n
couvre abcefhgbcefgi

cas4 : (5,3,4)\n (5,6)\n
couvre abcdgbcefgi

cas5 : (5,3,4)\n x,2\n (5,6)\n
couvre abcdgbcehgbcefgi

$$
\begin{aligned}
x1 + x2 + x3 + x4 + x5 &= 0\\
x1 + 2x2 + 2x3 + 2x4 + 3x5 &= 0\\
x4 + x5 &= 0\\
x1 + 2x2 + 2x3 + x4 + 2x5 &= 0\\
x1 + x2 + 2x3 + x4 + x5 &= 0\\
x2 + x5 &= 0\\
x3 &= 0\\
x2 + x3 + x4 + 2x5 &= 0\\
x2 + x3 + x5 &= 0
\end{aligned}
$$

| ab | bc | cd | ce | dg | ef | eh | fg | fh | gb | gi | hg |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 2 | 0 | 2 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 2 | 0 | 2 | 0 | 2 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 2 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 3 | 1 | 2 | 1 | 1 | 1 | 1 | 0 | 2 | 1 | 1 |

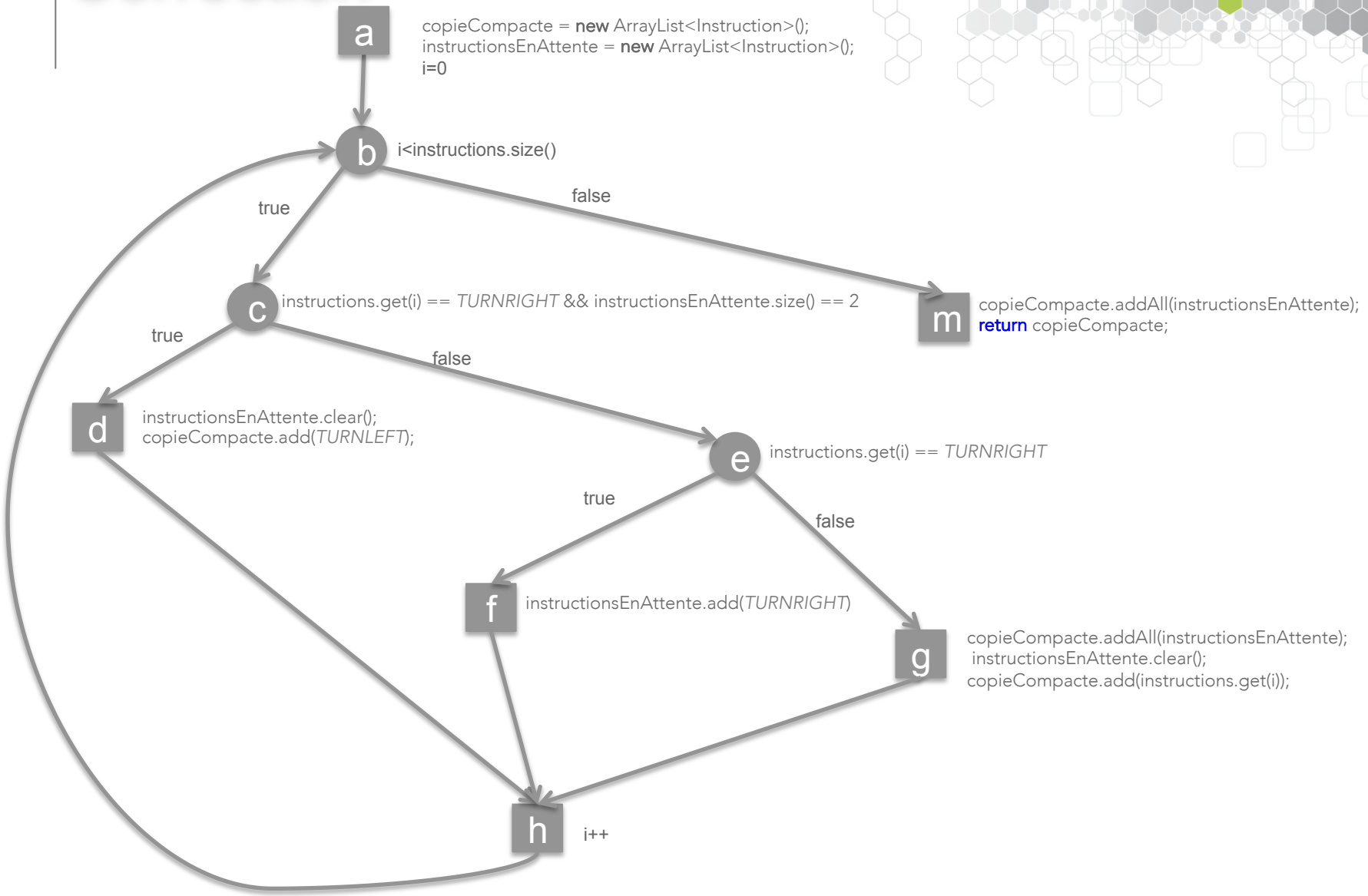
# Exercice 4

## Méthode compacte

- Produire le graphe de contrôle de la fonction ci-dessous
- Produire des cas de test pour couvrir tous les arcs et FPC

```
static List<Instruction> compacte(List<Instruction> instructions) {
    List<Instruction> copieCompacte = new ArrayList<Instruction>();
    List<Instruction> instructionsEnAttente = new ArrayList<Instruction>();
    for (int i = 0; i < instructions.size(); i++) {
        if (instructions.get(i) == TURNRIGHT && instructionsEnAttente.size() == 2) {
            instructionsEnAttente.clear();
            copieCompacte.add(TURNLEFT);
        } else if (instructions.get(i) == TURNRIGHT)
            instructionsEnAttente.add(TURNRIGHT);
        else {
            copieCompacte.addAll(instructionsEnAttente);
            instructionsEnAttente.clear();
            copieCompacte.add(instructions.get(i));
        }
    }
    copieCompacte.addAll(instructionsEnAttente);
    return copieCompacte;
}
```

# Correction


a
copieCompacte = new ArrayList<Instruction>();
instructionsEnAttente = new ArrayList<Instruction>();
i=0
b
i<instructions.size()
true
false
c
instructions.get(i) == TURNRIGHT && instructionsEnAttente.size() == 2
m
copieCompacte.addAll(instructionsEnAttente);
return copieCompacte;
true
false
d
instructionsEnAttente.clear();
copieCompacte.add(TURNLEFT);
e
instructions.get(i) == TURNRIGHT
true
false
f
instructionsEnAttente.add(TURNRIGHT)
g
copieCompacte.addAll(instructionsEnAttente);
instructionsEnAttente.clear();
copieCompacte.add(instructions.get(i));
h
i++

# Correction

La fonction admet en paramètre une liste d'instructions

Cas 1 :
< instructions= [FORWARD, TURNRIGHT, TURNRIGHT, TURNRIGHT] >
Oracle : [FORWARD, TURNLEFT]

couvre abceghbcefhbcefhbcdhbm donc tous les arcs mais n'atteint pas le critère FPC

instructions.get(i) == *TURNRIGHT* && instructionsEnAttente.size() == 2
est couvert à vrai et à faux mais pour atteindre la couverture FPC il faut :

| instructions.get(i) == *TURNRIGHT* | instructionsEnAttente.size() == 2 | |
|---|---|---|
| vrai | vrai | atteind au 3eme TURNRIGHT |
| vrai | faux | atteind au 1er TURNRIGHT |
| faux | vrai | Non atteind |

# Correction

Il faut donc compléter d'un autre cas de test

Cas 2 :
< instructions= [TURNRIGHT, TURNRIGHT, FORWARD] >
Oracle : [TURNRIGHT, TURNRIGHT, FORWARD]

qui couvre abcefhbcefhbceghbm et permet d'atteindre le critère FPC

Finalement le critère FPC mais aussi MCC est atteint

| instructions.get(i) == *TURNRIGHT* | instructionsEnAttente.size() == 2 | |
|---|---|---|
| vrai | vrai | cas 1 au 3eme TURNRIGHT |
| vrai | faux | cas 1 au 1er TURNRIGHT |
| faux | vrai | cas 2 à FORWARD |
| faux | faux | cas 1 à FORWARD |