Softwarequalität

Vorlesung 11 – Testen: White-box-Verfahren (Kontrollflussorientierte Verfahren)

Prof. Dr. Joel Greenyer

20. Juni 2016

White-/Glass-Box-Tests

In der letzten Vorlesung…

- **White-/Glass-Box-Tests** (Wdh.)
  - Testen eines Subjekts mit Kenntnis über dessen inneren Aufbau
  - Idee: Sicherstellen, dass möglichst viele Programmteile durch Tests ausgeführt werden (**Coverage**) – In Code, der nicht ausgeführt wurde, kann auch kein Fehler gefunden werden
  - **Sollwerte** werden weiterhin **von einer Spezifikation** abgeleitet
    - Sollwerte sind prinzipiell **nicht** aus dem Code ableitbar!

- Nachteil von Glass-Box-Tests:
  - Fehler können gefunden werden, aber nicht **fehlende** Funktionen

Beispiel

In der letzten Vorlesung…

- **Frage**: Welche Tests machen hier Sinn?

```java
/**
 * calculate the Manhattan distance
 * of two points p1 = (x1, y1), p2 = (x2, y2)
 * by calling manhattan(x1-x2, y1-y2)
 * @param a
 * @param b
 * @return |a| + |b|
 */
public int manhattan(int a, int b){
        if (a < 0)
                a = -a;
        if (b < 0)
                b = -b;
        return a+b;
}
```

Testüberdeckung (Coverage)

- **Ziel**: Kenntnis der internen Struktur systematisch nutzen
- **Kontrollflussorientierte** Tests:
  - Wie viele **Anweisungen** des Programms wurden abgedeckt?
  - Wie viele **Zweige** des Programms wurden abgedeckt?
  - Wie viele **Pfade** des Programms wurden abgedeckt?
  - Wie viele **Bedingungen** des Programms wurden abgedeckt?
  - …
- **Datenflussorientierte** Tests
  - Idee: Testen verschiedener Kombinationen von Schreib- und Lesezugriffen auf Variablen

White-/Glass-Box-Tests – Vorgehensweise

- Schritt 1: Analyse der Programmstruktur:
  – Aufbau eines Kontrollflussgraphen

- Schritt 2: Testkonstruktion
  – Erstellen einer Testfallmenge, sodass ein bestimmtes Überdeckungskriterium erfüllt ist
    - z.B. „alle Anweisungen durch mindestens einen Test abgedeckt“
  – (Woher kommen die Sollwerte?
    - Aus der Spezifikation – nicht aus dem Code!!)

- Schritt 3: Testdurchführung
  – Wie bei Black-Box-Tests auch

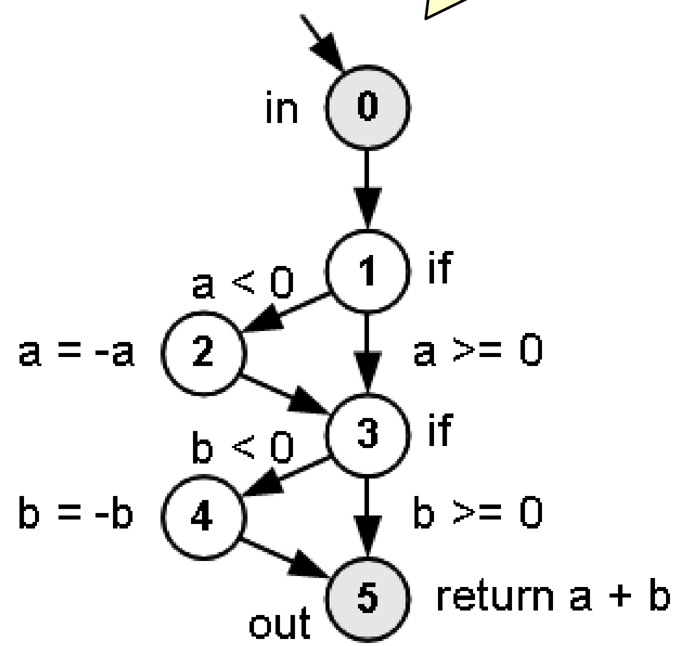
basiert auf Hoffmann, Dirk W. *Software-Qualität*, eXamen.press, 2. Auflage, Springer Berlin Heidelberg, 2013

Kontrollflussgraphen

- Kontrollflussgraphen beschreiben mögliche Positionsfolgen des Programmzeigers (program counters)
- Beispiel:

```java
/**
 * calculate the Manhattan distance
 * of two points p1 = (x1, y1), p2 = (x2, y2)
 * by calling manhattan(x1-x2, y1-y2)
 * @param a
 * @param b
 * @return |a| + |b|
 */
public int manhattan(int a, int b){
        if (a < 0)
                a = -a;
        if (b < 0)
                b = -b;
        return a+b;
}
```

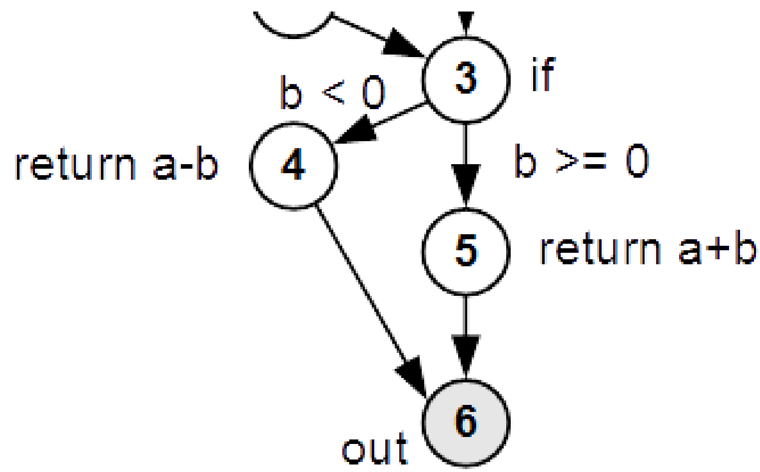
Zusätzlich hat ein Kontrollflussgraph immer **genau einen** Startknoten „**in**“ und **genau einen** Endknoten „**out**“.



Kontrollflussgraphen

- Ein Kontrollflussgraph hat genau einen Startknoten **in** und genau einen Endknoten **out**
  - Das vereinfacht die Definition einiger Überdeckungskriterien, die wir kennenlernen werden
- Das klingt nach einer Einschränkung – ist aber keine
  - Entweder einfach einen Knoten out als Nachfolger aller `return`-Knoten einfügen

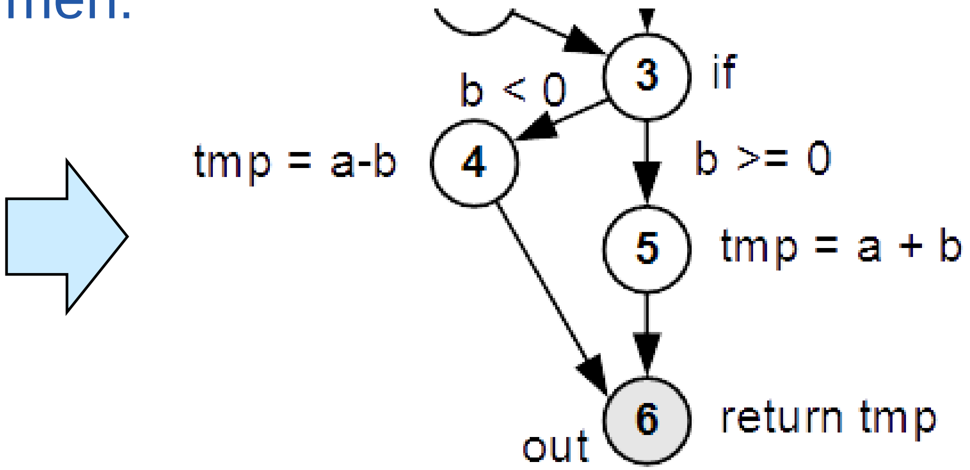
```
...
      if (a < 0)
            a = -a;
      if (b < 0)
            return a-b;
      return a+b;
```



Kontrollflussgraphen

- Ein Kontrollflussgraph hat genau einen Startknoten **in** und genau einen Endknoten **out**
  - Das vereinfacht die Definition einiger Überdeckungskriterien, die wir kennenlernen werden
- Das klingt nach einer Einschränkung – ist aber keine
  - Entweder einfach einen Knoten out als Nachfolger aller `return`-Knoten einfügen
  - Oder: Die Methode in äquivalente Methode mit nur einer `return`-Anweisung umformen:

```
...
    if (a < 0)
            a = -a;
    if (b < 0)
            tmp = a-b;
    else tmp = a+b;
    return tmp;
```

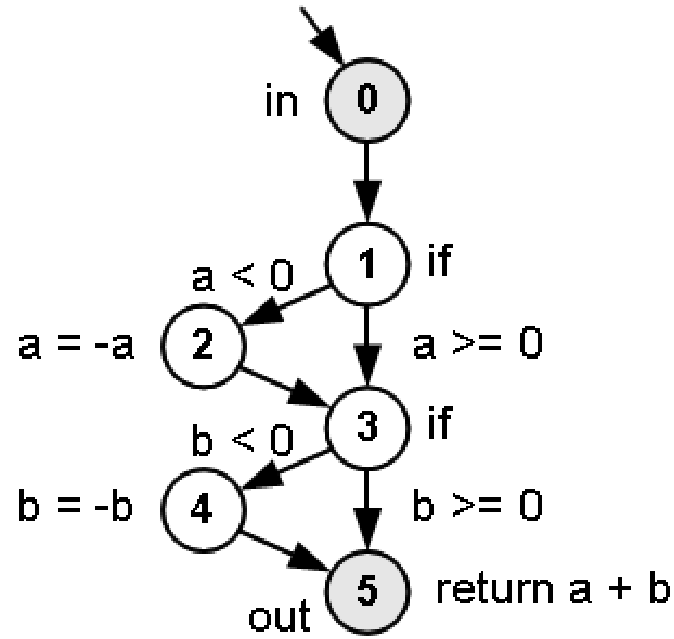


Kontrollflussgraphen

- Kontrollflussgraphen beschreiben mögliche Positionsfolgen des Programmzeigers (program counters)
- Beispiel:

```
/**
 * calculate the Manhattan distance
 * of two points p1 = (x1, y1), p2 = (x2, y2)
 * by calling manhattan(x1-x2, y1-y2)
 * @param a
 * @param b
 * @return |a| + |b|
 */
public int manhattan(int a, int b){
1------- if (a < 0)
2-------------- a = -a;
3------- if (b < 0)
4-------------- b = -b;
5------- return a+b;
}
```

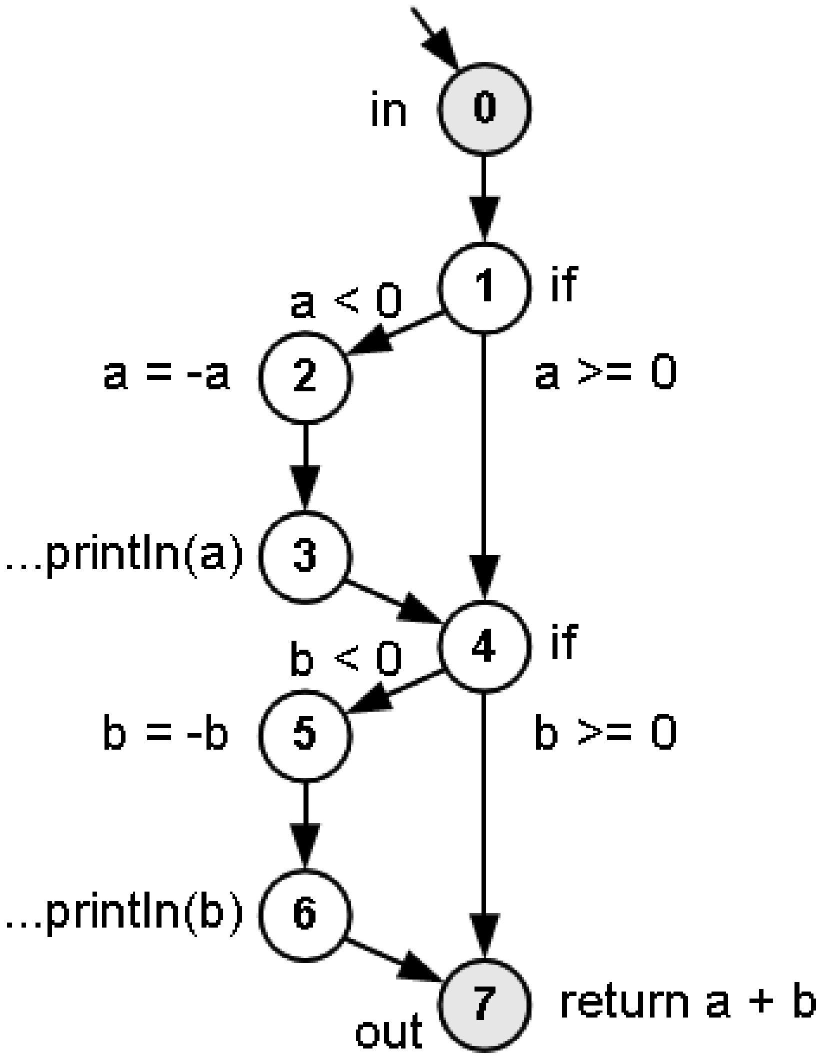
Tipp: Zeilennummern als Knotenzahlen wählen!



Kontrollflussgraphen

- **Expandierte** Kontrollflussgraphen
  - Pro Anweisung einen Knoten

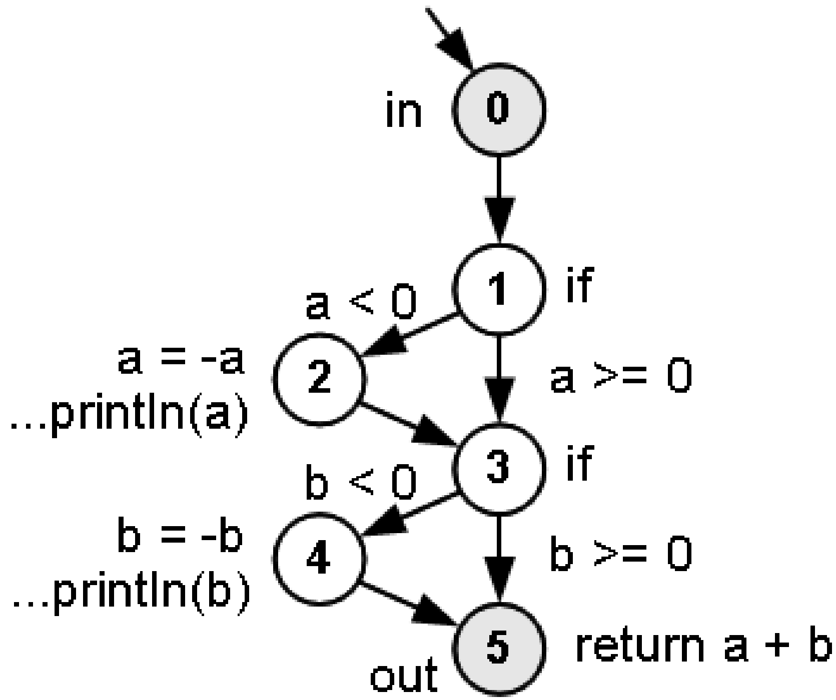
```
public int manhattan(int a, int b){
        if (a < 0){
                a = -a;
                System.out.println(a);
        }if (b < 0){
                b = -b;
                System.out.println(b);
        }
        return a+b;
}
```



Kontrollflussgraphen

- **Kollabierte** Kontrollflussgraphen
  - Verzweigungsfreie Blöcke werden zu einem Knoten

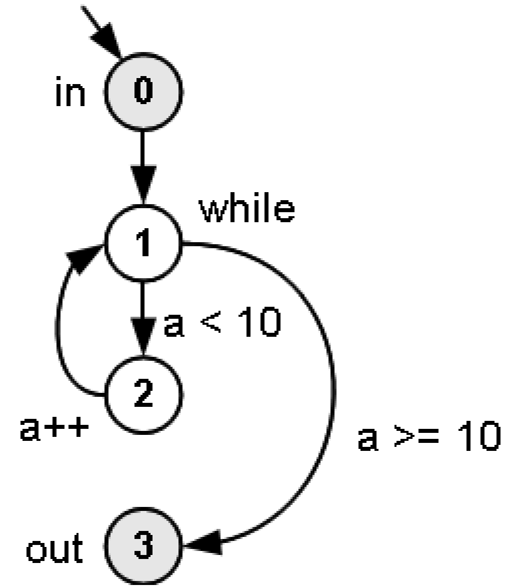
```java
public int manhattan(int a, int b){
        if (a < 0){
                a = -a;
                System.out.println(a);
        }if (b < 0){
                b = -b;
                System.out.println(b);
        }
        return a+b;
}
```



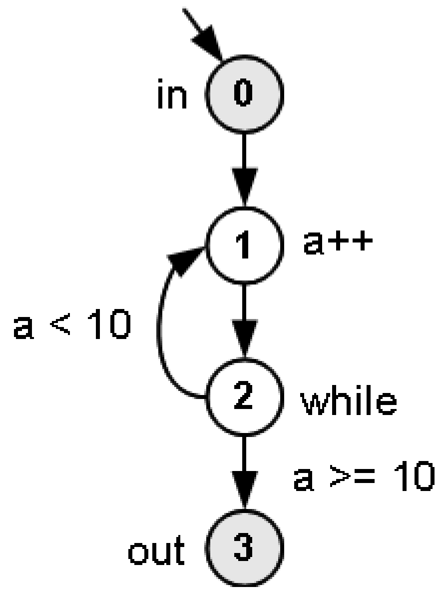
Kontrollflussgraphen

- Darstellung von Schleifen:

```
while(a<10){
        a++;
}
```




```
do {
        a++;
} while(a<10);
```



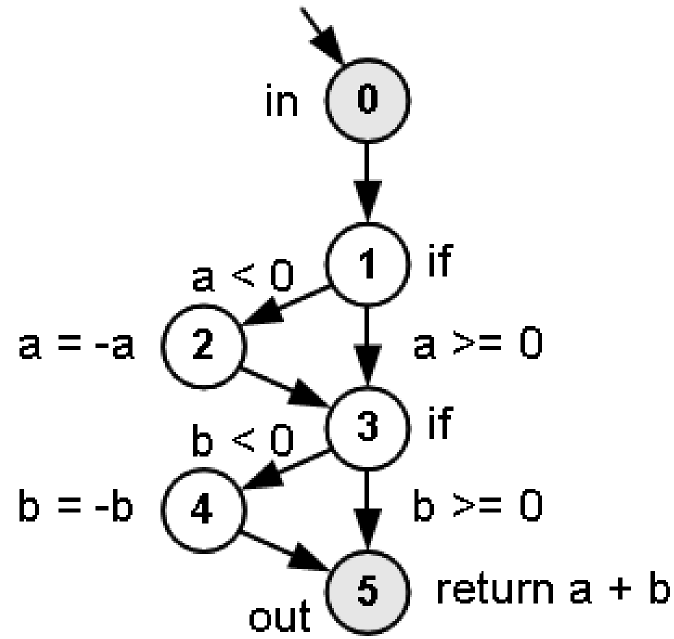
Anweisungsüberdeckung (Statement Coverage o. Node Coverage, NC)

- Forderung: Jeder Knoten des Kontrollflussgraphen überdecken
- Welche Menge an Testfällen überdecken alle Anweisungen?
  - Ein Testfall:

```
manhattan(-1, -1); Soll-Ergebnis: 2
```

Manchmal sprechen wir im Folgenden nicht über die Sollwerte. Aber die gehören natürlich zu jedem Testfall dazu!

```java
public int manhattan(int a, int b){
        if (a < 0)
                a = -a;
        if (b < 0)
                b = -b;
        return a+b;
}
```



Zweigüberdeckung (Branch Coverage o. Edge Coverage, EC)

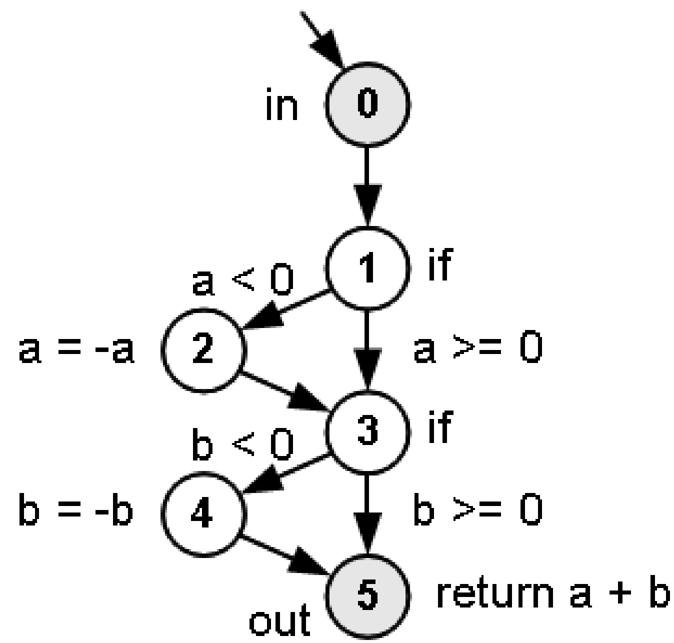
- Forderung: Jede Kante des Kontrollflussgraphen überdecken
- Welche Menge an Testfällen überdecken alle Kanten?
  - Zwei Testfälle:

manhattan(-1, -1); Soll-Ergebnis: 2
überdeckt (0,1), (1,2), (2,3), (3,4), (4,5)

manhattan(1, 1); Soll-Ergebnis: 2
überdeckt (0,1), (1,3), (3,5)

Alle Kanten:
(0,1), (1,2), (2,3), (3,4), (4,5), (1,3), (3,5)

```java
public int manhattan(int a, int b){
        if (a < 0)
                a = -a;
        if (b < 0)
                b = -b;
        return a+b;
}
```



Anweisungs- und Zweigüberdeckung

- Zweigüberdeckung schließt Anweisungsüberdeckung mit ein
- Anweisungsüberdeckung ist oft nicht ausreichend
  - Nur Tests, wo if-Bedingung wahr ist (wenn kein else vorhanden)
- Zweigüberdeckung wird von einigen Standards gefordert
  - Z.B. RCTA-DO-178B („Software Considerations in Airborne Systems and Equipment Certification“) für alle Komponenten, bei denen Fehler signifikant, aber nicht sicherheitskritisch sind
- Laut Experimenten von Wei *et al.* [1] ist aber auch Zweiüberdeckung kein gutes Kriterium
  - Hohe Zweigüberdeckung war durch Zufallstests schnell erreicht
  - Über 50% der Fehler erst durch weitere Zufallstests gefunden
  - also kein gutes Kriterium um Zufallstest abzubrechen

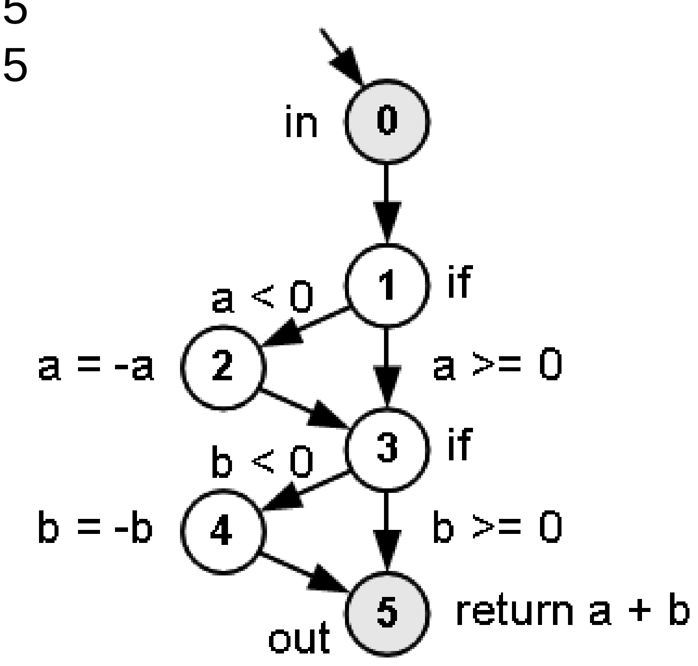
[1] Wei, Y.; Meyer, B. & Oriol, M. Is Branch Coverage a Good Measure of Testing Effectiveness? *Empirical Software Engineering and Verification, Springer Berlin Heidelberg,* 2012, *7007*, 194-212

Pfadüberdeckung (Path Coverage, PC)

- Forderung: Jeden möglichen Pfad vom Eingangs- zum Ausgangsknoten des Kontrollflussgraphen überdecken
- Welche Menge an Testfällen überdecken alle Pfade?
  - Vier Testfälle (für vier Pfade):

manhattan(-1, -1); überdeckt Pfad 0, 1, 2, 3, 4, 5
manhattan(-1, 1); überdeckt Pfad 0, 1, 2, 3, 5
manhattan(1, -1); überdeckt Pfad 0, 1, 3, 4, 5
manhattan(1, 1); überdeckt Pfad 0, 1, 3, 5

```
public int manhattan(int a, int b){
        if (a < 0)
                a = -a;
        if (b < 0)
                b = -b;
        return a+b;
}
```



Pfadüberdeckung (Path Coverage, PC)

- Es kann sein, dass nicht alle Pfade im Kontrollflussgraphen möglich sind
  - Diese müssen dann natürlich nicht abgedeckt werden
- Beispiel:
  - Ein Ausführung des Programms, bei dem „Failed“ und „With distinction“ ausgegeben wird, ist nicht möglich

```
public int score(int a){
        if (a >= 50)
                System.out.println("Passed");
        else
                System.out.println("Failed");
        if (a >= 95)
                System.out.println("With distinction");
}
```

Varianten der Pfadüberdeckung

- Pfadüberdeckung ist ein hohes Ziel
  - Dann werden alle Durchläufe des Programms getestet
- Aber schon sehr kleine Programme können, durch **Schleifen**, **sehr viele** oder **unendlich viele Pfade** besitzen
- Angenommen:
  - Programm hat eine Schleife, die n-mal durchlaufen wird
  - In der while-Schleife ist eine if-then-else-Bedingung
  - Wie viele Pfade hat das Programm mindestens?
  - **Antwort**: **2^n viele Pfade!**

- Daher gibt es z.B. die **strukturierte Pfadüberdeckung**, bei der **für ein gewähltes k** nur alle Pfade überdeckt werden müssen, bei denen **Schleifen mit bis zu k Iterationen durchlaufen** werden

Varianten der Pfadüberdeckung

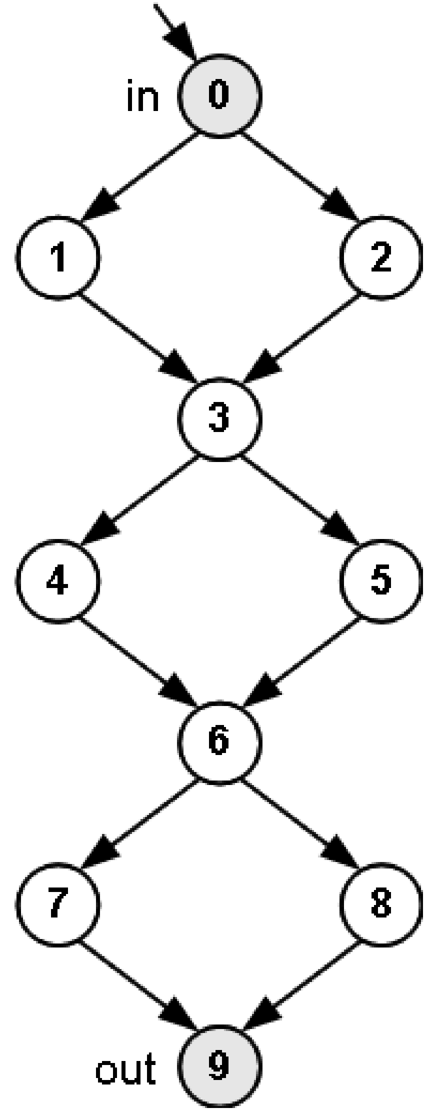
- Eine Mischung aus Kanten- und Pfadüberdeckung ist die Überdeckung von **Kantenpaaren** oder **Teilpfaden**
  - Engl. auch **Edge-Pair-Coverage (EPC)**

- Forderung: Jedes Paar aufeinanderfolgender Kanten im Kontrollflussgraphen muss abgedeckt werden
  - Oder für ein gewähltes k jeder Teilpfad der Länge k

Varianten der Pfadüberdeckung

- Edge-Pair-Coverage Beispiel:
  - Wie viele Pfade gibt es?
    - Es gibt 2*2*2=8 Pfade:

p1 = (0, 1, 3, 4, 6; 7, 9)
p2 = (0, 2, 3, 4, 6; 7, 9)
p3 = (0, 1, 3, 5, 6; 7, 9)
p4 = (0, 2, 3, 5, 6; 7, 9)
p5 = (0, 1, 3, 4, 6; 8, 9)
p6 = (0, 2, 3, 4, 6; 8, 9)
p7 = (0, 1, 3, 5, 6; 8, 9)
p8 = (0, 2, 3, 5, 6; 8, 9)



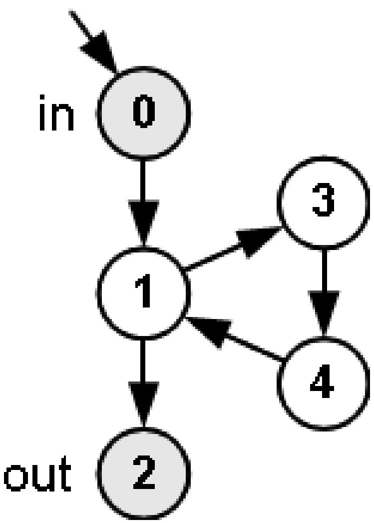
  - Wie groß ist kleinste Pfadmenge, die alle Kanten überdeckt?
    - Antwort: 2, z.B. {p1, p8}
  - …, die alle Paare von Kanten überdeckt?
    - Antwort: 4, z.B. {p1, p8, p3, p6}

Weitere Variante: Primärpfadüberdeckung (Prime Path Coverage, PPC)

- Ein **einfacher Pfad** in einen Kontrollflussgraphen ist ein Pfad zwischen zwei Knoten des Graphen
  - In dem kein Knoten zweimal vorkommt
  - Ausnahme: Start- und Endknoten können identisch sein
- Ein **Primärpfad** ist ein maximaler einfacher Pfad
  - Kann nicht zu einem anderen einfachen Pfad erweitert werden
- **Idee**: Alle Primärpfade überdecken
- Beispiel:
  - Primärpfade?
  - Welche Tests decken alle Primärpfade ab?
    - Tests, die Testpfade p1 und p7 durchlaufen

p1 = (0, 1, 2)
p2 = (0, 1, 3, 4)
p3 = (1, 3, 4, 1)
p4 = (3, 4, 1, 3)
p5 = (4, 1, 3, 4)
p6 = (3, 4, 1, 2)

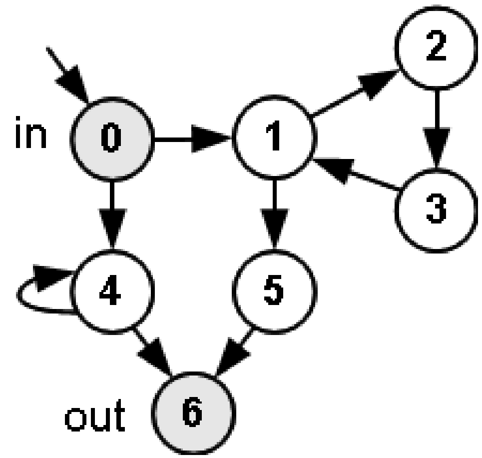
p7 = (0, 1, 3, 4, 1, 3, 4, 1, 2)
deckt p2-p6 ab



aus Ammann, P. and Offutt, J. *Introduction to Software Testing*, Cambridge University Press, 2008

Weitere Variante: Primärpfadüberdeckung (Prime Path Coverage, PPC)

- Berechnung aller Primärpfade
  - Schritt 1: Alle einfachen Pfade der Länge 0 finden, diese Erweitern zu Länge 1, 2, usw.
  - Schritt 2: Maximale auswählen
- Beispiel:



Länge 0	Länge 1	Länge 2	Länge 3	Länge 4
p1 = (0)	p8 = (0, 1)	p17 = (0, 1, 2)	p25 = (0, 1, 2, 3)	p32 = (2, 3, 1, 5, 6)!
p2 = (1)	p9 = (0, 4)	p18 = (0, 1, 5)	p26 = (0, 1, 5, 6)!	
p3 = (2)	p10 = (1, 2)	p19 = (0, 4, 6)!	p27 = (1, 2, 3, 1)*	
p4 = (3)	p11 = (1, 5)	p20 = (1, 2, 3)	p28 = (2, 3, 1, 2)*	
p5 = (4)	p12 = (2, 3)	p21 = (1, 5, 6)!	p29 = (2, 3, 1, 5)	
p6 = (5)	p13 = (3, 1)	p22 = (2, 3, 1)	p30 = (3, 1, 2, 3)*	
p7 = (6)!	p14 = (4, 4)*	p23 = (3, 1, 2)	p31 = (3, 1, 5, 6)!	
	p15 = (4, 6)!	p24 = (3, 1, 5)		
	p16 = (5, 6)!			

*: Kann nicht erweitert werden, da Anfangs- gleich Endknoten

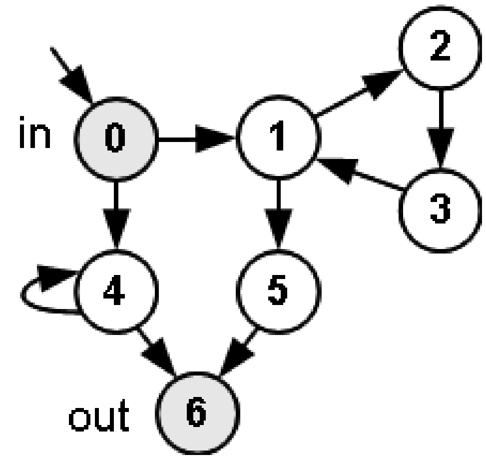
!: … da Endknoten keine ausgehende Kante hat

aus der Vorlesung *Software Testing*, von Itai Segall, Tel Aviv University 2013/14

Weitere Variante: Primärpfadüberdeckung (Prime Path Coverage, PPC)

- Alle Primärpfade
  - maximale auswählen (welcher Pfad ist nicht Teilpfad eines anderen?)

p14 = (4, 4)*
p19 = (0, 4, 6)!
p25 = (0, 1, 2, 3)
p26 = (0, 1, 5, 6)!
p27 = (1, 2, 3, 1)*
p28 = (2, 3, 1, 2)*
p30 = (3, 1, 2, 3)*
p32 = (2, 3, 1, 5, 6)!



p1 = (0)	p8 = (0, 1)	p17 = (0, 1, 2)	p25 = (0, 1, 2, 3)	p32 = (2, 3, 1, 5, 6)!
p2 = (1)	p9 = (0, 4)	p18 = (0, 1, 5)	p26 = (0, 1, 5, 6)!	
p3 = (2)	p10 = (1, 2)	p19 = (0, 4, 6)!	p27 = (1, 2, 3, 1)*	
p4 = (3)	p11 = (1, 5)	p20 = (1, 2, 3)	p28 = (2, 3, 1, 2)*	
p5 = (4)	p12 = (2, 3)	p21 = (1, 5, 6)!	p29 = (2, 3, 1, 5)	
p6 = (5)	p13 = (3, 1)	p22 = (2, 3, 1)	p30 = (3, 1, 2, 3)*	
p7 = (6)!	p14 = (4, 4)*	p23 = (3, 1, 2)	p31 = (3, 1, 5, 6)!	
	p15 = (4, 6)!	p24 = (3, 1, 5)		
	p16 = (5, 6)!			

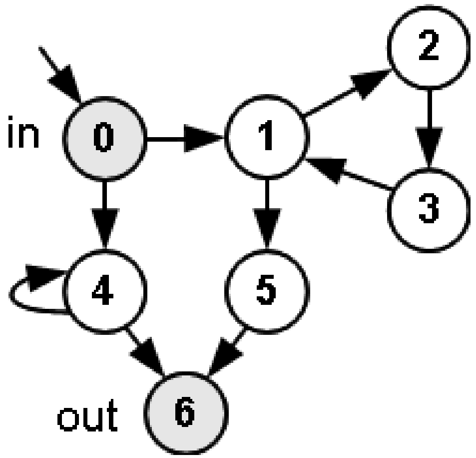
aus der Vorlesung *Software Testing*, von Itai Segall, Tel Aviv University 2013/14

Weitere Variante: Primärpfadüberdeckung (Prime Path Coverage, PPC)

- Schritt 3: Die Primärpfade zu Testpfade erweitern
  - Erweitern zu Start- und Endknoten des Kontrollflussgraphen
  - Mit dem längsten Primärpfad anfangen

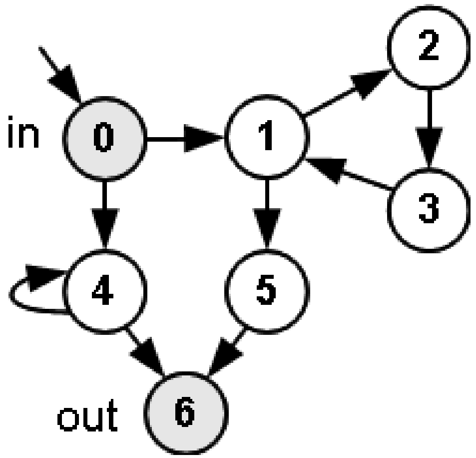
Primärpfade
p14 = (4, 4)*
p19 = (0, 4, 6)!
~~p25 = (0, 1, 2, 3)~~
p26 = (0, 1, 5, 6)!
~~p27 = (1, 2, 3, 1)*~~
p28 = (2, 3, 1, 2)*
p30 = (3, 1, 2, 3)*
~~p32 = (2, 3, 1, 5, 6)!~~

Testpfade
pt1 = (0, 1, 2, 3, 1, 5, 6)



Weitere Variante: Primärpfadüberdeckung (Prime Path Coverage, PPC)

- Schritt 3: Die Primärpfade zu Testpfade erweitern
  - Erweitern zu Start- und Endknoten des Kontrollflussgraphen
  - Mit dem längsten Primärpfad anfangen



Primärpfade

p14 = (4, 4)*
p19 = (0, 4, 6)!
~~p25 = (0, 1, 2, 3)~~
~~p26 = (0, 1, 5, 6)!~~
~~p27 = (1, 2, 3, 1)*~~
p28 = (2, 3, 1, 2)*
p30 = (3, 1, 2, 3)*
~~p32 = (2, 3, 1, 5, 6)!~~

Testpfade

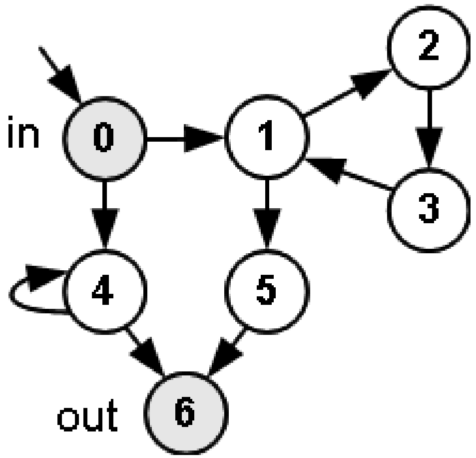
pt1 = (0, 1, 2, 3, 1, 5, 6)
pt2 = (0, 1, 5, 6)

Weitere Variante: Primärpfadüberdeckung (Prime Path Coverage, PPC)

- Schritt 3: Die Primärpfade zu Testpfade erweitern
  - Erweitern zu Start- und Endknoten des Kontrollflussgraphen
  - Mit dem längsten Primärpfad anfangen

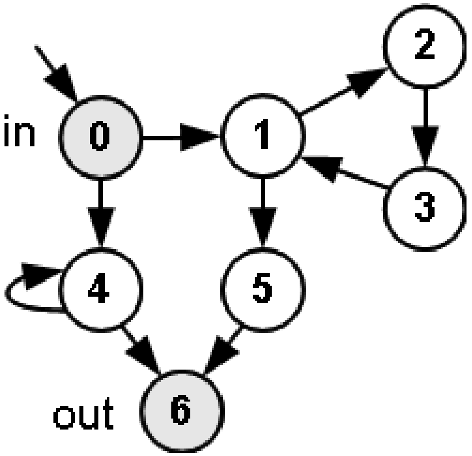
Primärpfade
p14 = (4, 4)*
p19 = (0, 4, 6)!
~~p25 = (0, 1, 2, 3)~~
~~p26 = (0, 1, 5, 6)!~~
~~p27 = (1, 2, 3, 1)*~~
~~p28 = (2, 3, 1, 2)*~~
~~p30 = (3, 1, 2, 3)*~~
~~p32 = (2, 3, 1, 5, 6)!~~

Testpfade
pt1 = (0, 1, 2, 3, 1, 5, 6)
pt2 = (0, 1, 5, 6)
pt3 = (0, 1, 2, 3, 1, 2, 3, 1, 5, 6)



Weitere Variante: Primärpfadüberdeckung (Prime Path Coverage, PPC)

- Schritt 3: Die Primärpfade zu Testpfade erweitern
  - Erweitern zu Start- und Endknoten des Kontrollflussgraphen
  - Mit dem längsten Primärpfad anfangen

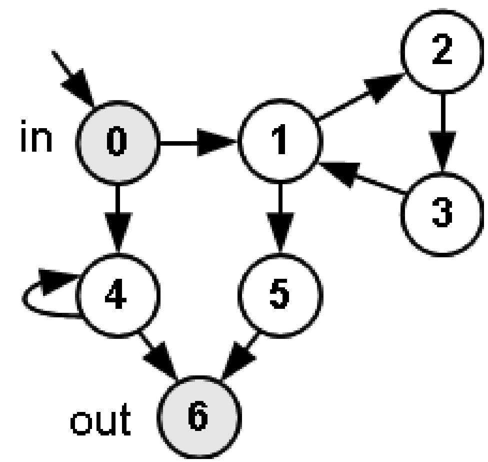


Primärpfade
p14 = (4, 4)*
~~p19 = (0, 4, 6)!~~
~~p25 = (0, 1, 2, 3)~~
~~p26 = (0, 1, 5, 6)!~~
~~p27 = (1, 2, 3, 1)*~~
~~p28 = (2, 3, 1, 2)*~~
~~p30 = (3, 1, 2, 3)*~~
~~p32 = (2, 3, 1, 5, 6)!~~

Testpfade
pt1 = (0, 1, 2, 3, 1, 5, 6)
pt2 = (0, 1, 5, 6)
pt3 = (0, 1, 2, 3, 1, 2, 3, 1, 5, 6)
pt4 = (0, 4, 6)

Weitere Variante: Primärpfadüberdeckung (Prime Path Coverage, PPC)

- Schritt 3: Die Primärpfade zu Testpfade erweitern
  - Erweitern zu Start- und Endknoten des Kontrollflussgraphen
  - Mit dem längsten Primärpfad anfangen



Primärpfade

~~p14 = (4, 4)*~~
~~p19 = (0, 4, 6)!~~
~~p25 = (0, 1, 2, 3)~~
~~p26 = (0, 1, 5, 6)!~~
~~p27 = (1, 2, 3, 1)*~~
~~p28 = (2, 3, 1, 2)*~~
~~p30 = (3, 1, 2, 3)*~~
~~p32 = (2, 3, 1, 5, 6)!~~

Testpfade

pt1 = (0, 1, 2, 3, 1, 5, 6)
pt2 = (0, 1, 5, 6)
pt3 = (0, 1, 2, 3, 1, 2, 3, 1, 5, 6)
pt4 = (0, 4, 6)
pt5 = (0, 4, 4, 6)

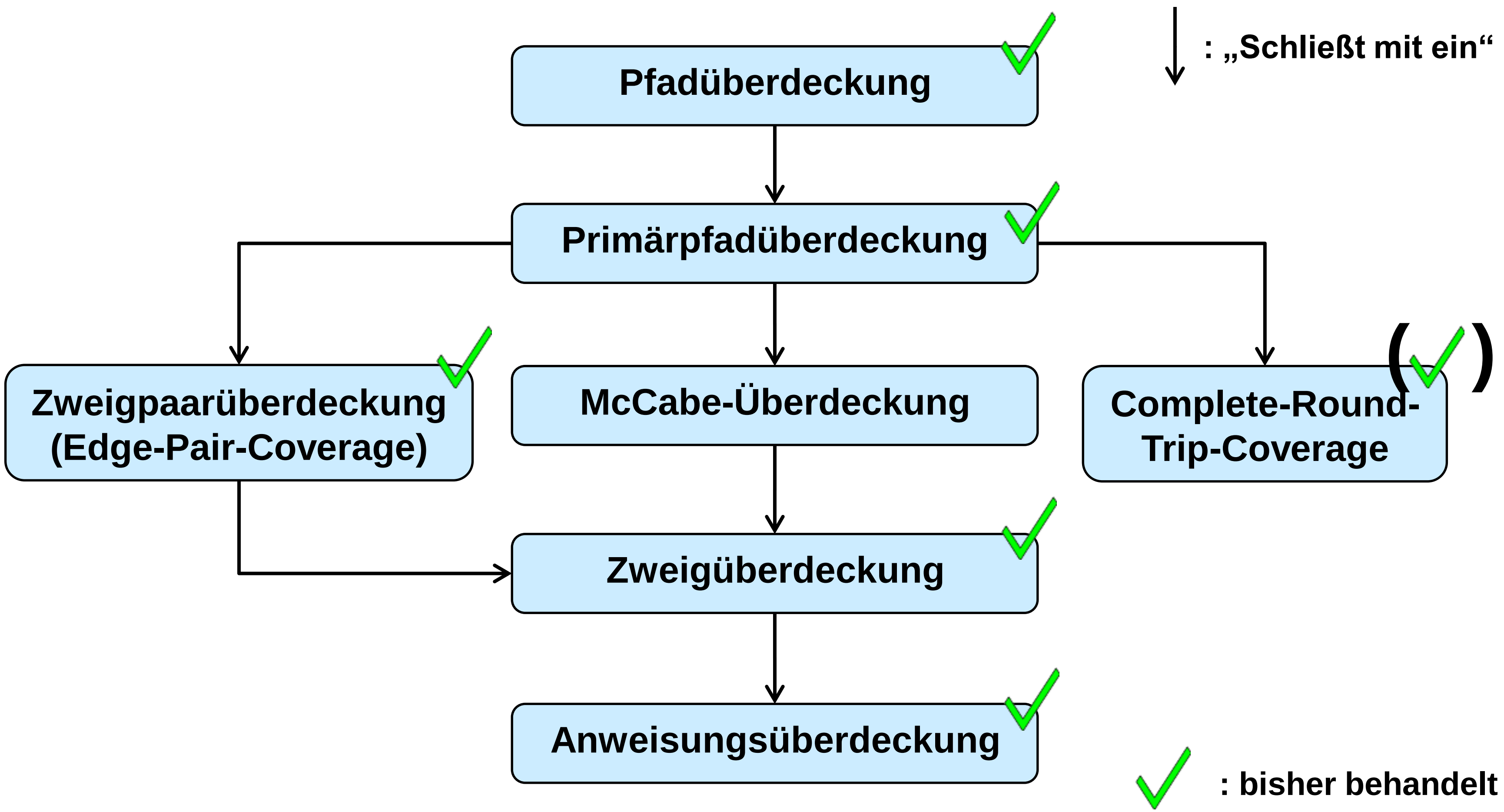
Weitere Variante: Primärpfadüberdeckung (Prime Path Coverage, PPC)

- Vorteil der Primärpfadüberdeckung
  - Garantiert auch **Complete-Round-Trip-Coverage (CRTC)** (Für jeden Knoten wird jeder Zyklus zurück zu diesem Knoten getestet, wenn es einen solchen Zyklus gibt)

- Nachteil
  - Bei vielen Verzweigungen kann die Menge der Primärpfade exponentiell steigen

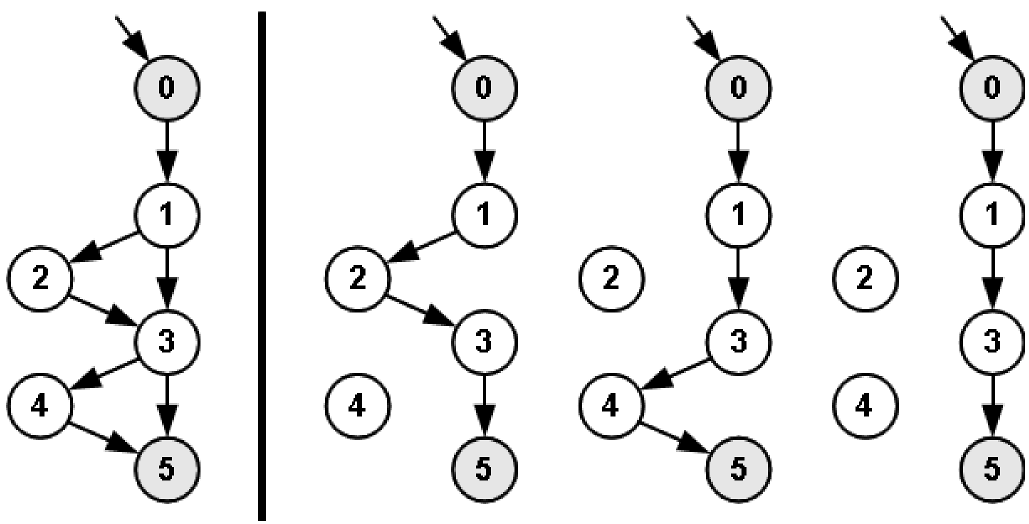
Überdeckungskriterien im Vergleich

- Einige Überdeckungskriterien schließen andere mit ein:



Weitere Variante: McCabe-Überdeckung (Basis Path Coverage, BPC)

- **Idee**: Für jeden Kontrollflussgraphen gibt es eine Menge von vollständigen Pfaden, sodass jeder andere vollständige Pfad aus diesen Pfaden „konstruiert" werden kann
  - Vollständige Pfade: Vom Eingangs- zum Ausgangsknoten
- Diese werden **linear unabhängige Pfade**, **Elementarpfade** oder **Basispfade** genannt
- Beispiel: Drei Basispfade des Manhattan-Beispiels

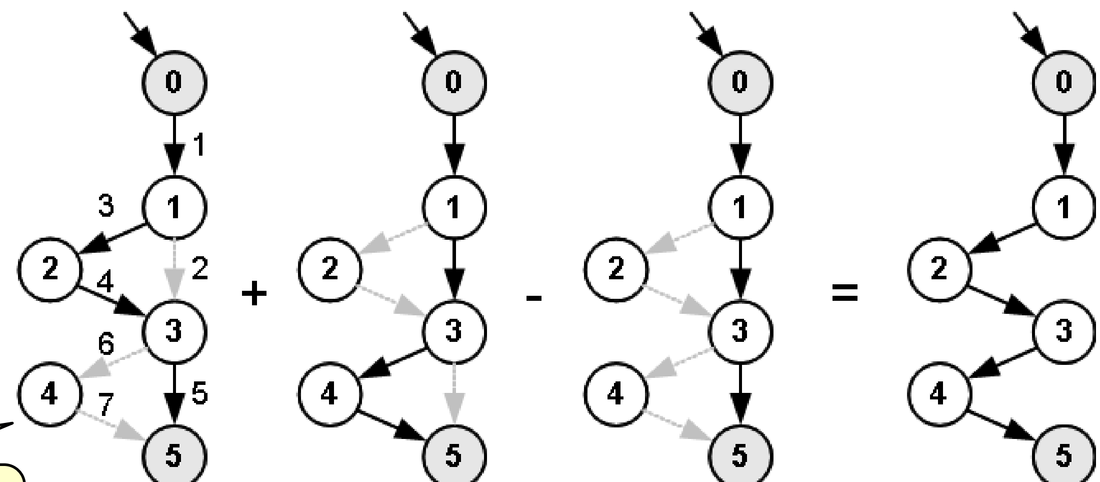


basiert auf Hoffmann, Dirk W. *Software-Qualität*, eXamen.press, 2. Auflage, Springer Berlin Heidelberg, 2013

Weitere Variante: McCabe-Überdeckung (Basis Path Coverage, BPC)

- Was heißt hier „konstruieren“?
  - Pfade lassen sich als **Kantenvektoren** aus $\mathbb{N}^{|E|}$ darstellen, wobei die i-te Stelle des Vektors besagt, wie oft der Pfad die i-te Kante des Kontrollflussgraphen besucht hat.
  - Für eine Menge an Basispfaden gilt, dass der Kantenvektor jedes vollständigen Pfads im Graphen sich aus einer Linearkombination der Vektoren der Basispfade berechnen lässt.
- Beispiel:

$$\begin{bmatrix}1\\0\\1\\1\\1\\0\\0\end{bmatrix} + \begin{bmatrix}1\\1\\0\\0\\0\\1\\1\end{bmatrix} - \begin{bmatrix}1\\1\\0\\0\\1\\0\\0\end{bmatrix} = \begin{bmatrix}1\\0\\1\\1\\0\\1\\1\end{bmatrix} \Leftrightarrow$$

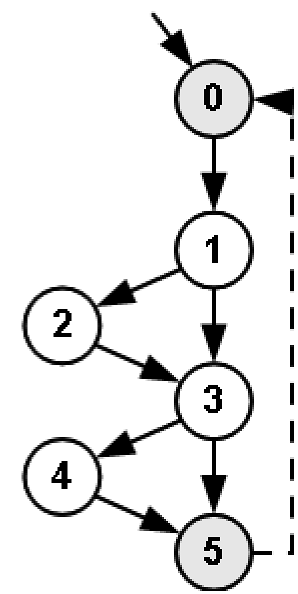


Durchnummerierung der Kanten entspricht Stelle im Vektor

Weitere Variante: McCabe-Überdeckung (Basis Path Coverage, BPC)

- Aus der Graphentheorie ist bekannt, dass ein stark zusammenhängender Graph |E| - |N| + 1 viele Basispfade hat

- Aber ein Kontrollflussgraph ist nicht strak zusammenhängend
  - Wir können ihn aber zusammenhängend machen
  - Annahme: Es gibt immer einen Eingangs- und einen Ausgangsknoten
  - Dann gibt es für einen Kontrollflussgraphen |E| - |N| + 2 viele Basispfade

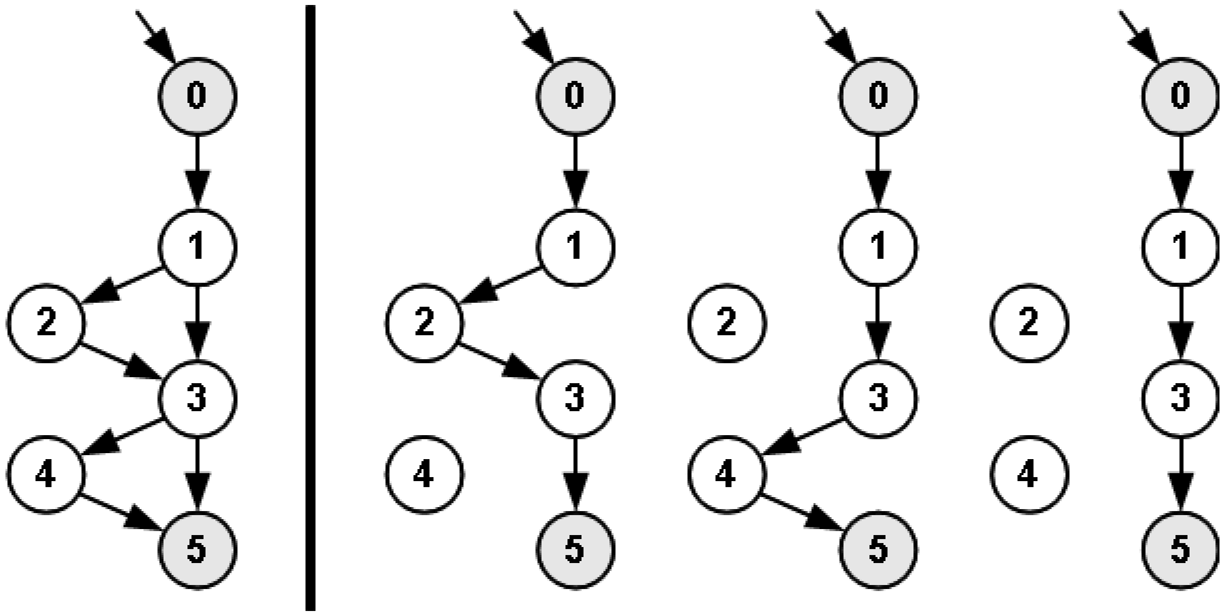


- Diese Zahl heißt auch „zyklomatische Komplexität“
  - Wird auch als Metrik für Fehlerrisiko genutzt
  - Laut McCabe: >10: mittel; >20 hoch; >50 unbeherrschbar

Auch ein Maß für Testbarkeit!

Weitere Variante: McCabe-Überdeckung (Basis Path Coverage, BPC)

- Forderung: Tests müssen alle Basispfade testen
  - Das schließt Anweisungs- und Zweigüberdeckung mit ein
- Ist schwächer als Primärpfadüberdeckung
  - Garantiert z.B. nicht Complete-Round-Trip-Coverage (CRTC)



Überdeckungskriterien im Vergleich

- Einige Überdeckungskriterien schließen andere mit ein: