Kontrol Akışı Analizine (Control Flow Analysis) Dayalı

Saydam Kutu Kapsama Kriterleri

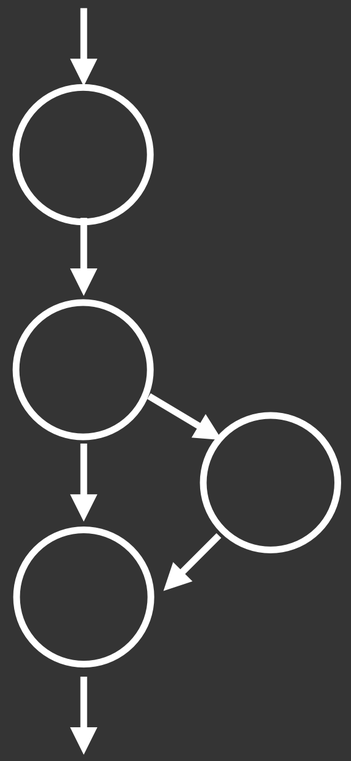
Kontrol Akışı

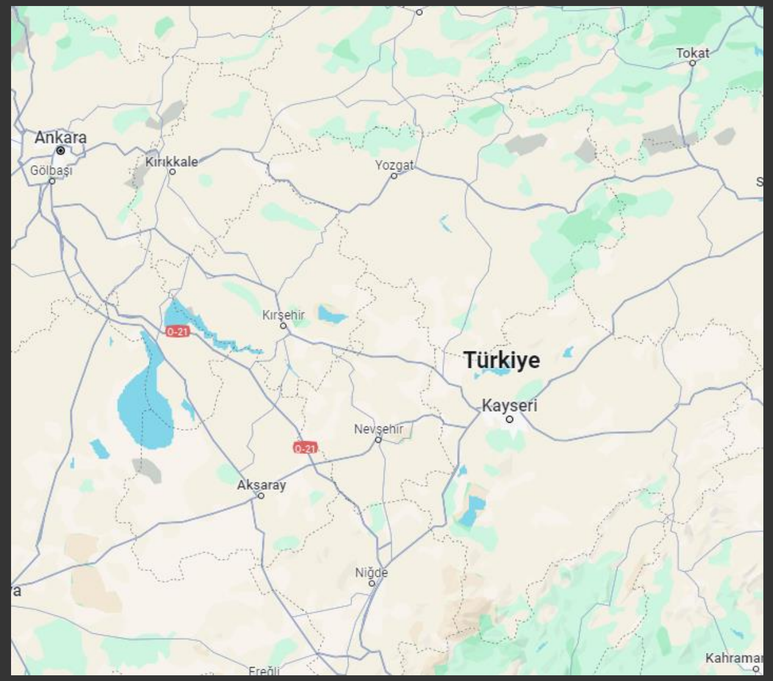
```java
public static Set<Character> duplicateLetters(String s) {
    // lower case the string and remove all characters
    // that are not letters
    s = s.toLowerCase().replaceAll("[^a-z.]", "");

    // initialise the result set
    Set<Character> duplicates = new TreeSet<>();

    // iterate through the string
    for (int i = 0; i < s.length(); i++) {
        char si = s.charAt(i);

        // iterate through the rest of the string,
        // checking for the same letter
        for (int j = i + 1; j < s.length(); j++) {
            char sj = s.charAt(j);

            if (si == sj) {
                // a match has been found, add it to
                // the result set
                duplicates.add(si);
            }
        }
    }

    return duplicates;
}
```

Kontrol Akış Grafı

```java
public static Set<Character> duplicateLetters(String s) {
    // lower case the string and remove all characters
    // that are not letters
    s = s.toLowerCase().replaceAll("[^a-z.]", "");

    // initialise the result set
    Set<Character> duplicates = new TreeSet<>();

    // iterate through the string
    for (int i = 0; i < s.length(); i++) {
        char si = s.charAt(i);

        // iterate through the rest of the string,
        // checking for the same letter
        for (int j = i + 1; j < s.length(); j++) {
            char sj = s.charAt(j);

            if (si == sj) {
                // a match has been found, add it to
                // the result set
                duplicates.add(si);
            }
        }
    }

    return duplicates;
}
```

Control Flow Graph (CFG)

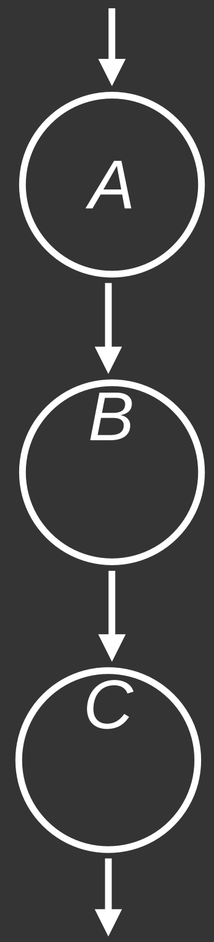





```java
public static Set<Character> duplicateLetters(String s) {
    // lower case the string and remove all characters
    // that are not letters
    s = s.toLowerCase().replaceAll("[^a-z.]", "");

    // initialise the result set
    Set<Character> duplicates = new TreeSet<>();

    // iterate through the string
    for (int i = 0; i < s.length(); i++) {
        char si = s.charAt(i);

        // iterate through the rest of the string,
        // checking for the same letter
        for (int j = i + 1; j < s.length(); j++) {
            char sj = s.charAt(j);

            if (si == sj) {
                // a match has been found, add it to
                // the result set
                duplicates.add(si);
            }
        }
    }

    return duplicates;
}
```

İfadelerin Doğrusal Sırası

```
doSomething(); /* A */
doSomethingElse(); /* B */
doSomethingDifferent(); /* C */
```

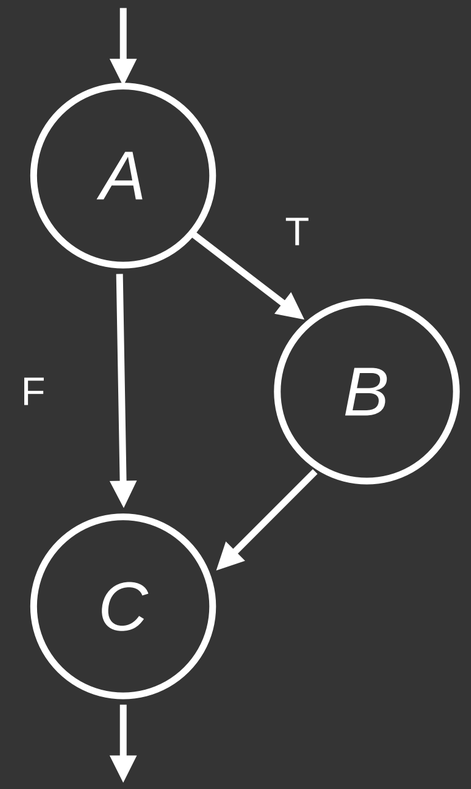


İfadelerin Doğrusal Sırası

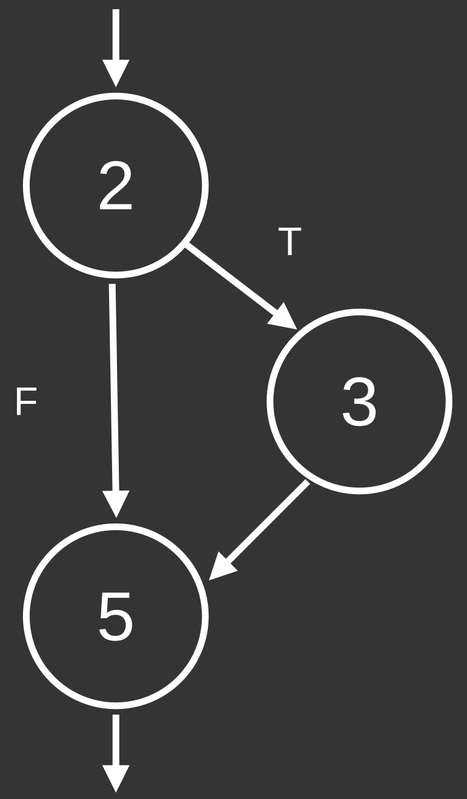
```
doSomething(); /* A */
doSomethingElse(); /* B */
doSomethingDifferent(); /* C */
```

“If” Yapısı

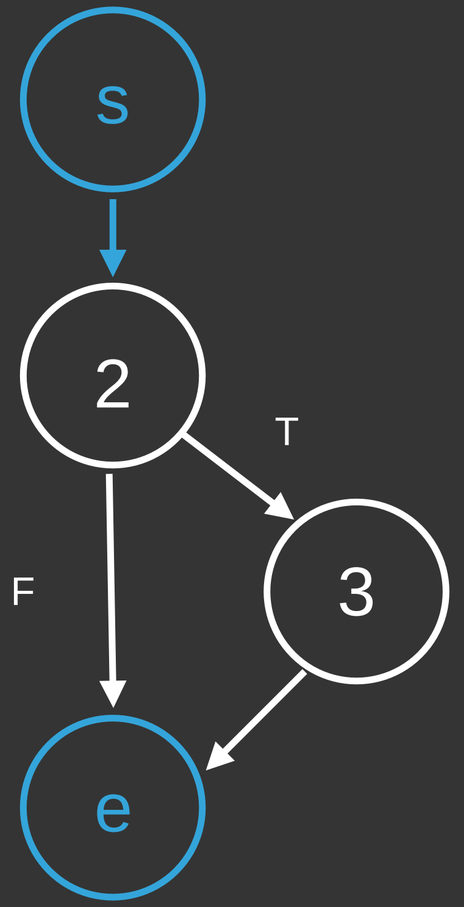
```
if (condition) { /*A*/
  doSomething(); /* B */
}
/* C */
```



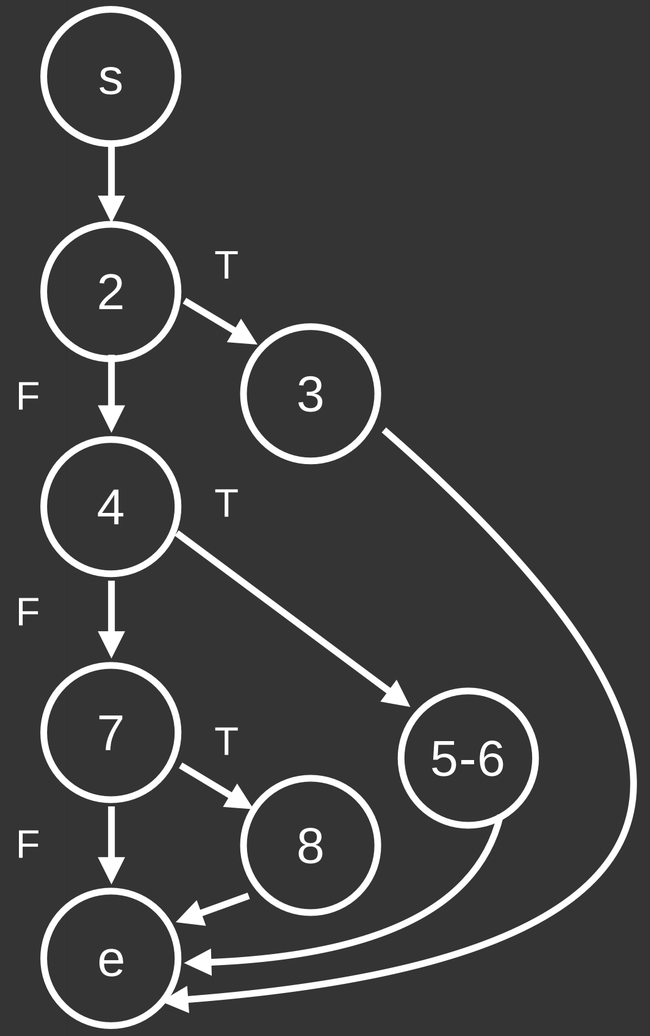
```
1 public void printGreeting() {
2   if (isMorning()) {
3     System.out.println("Good Morning!");
4   }
5 }
```



```java
public void printGreeting() {
  if (isMorning()) {
    System.out.println("Good Morning!");
  }
}
```

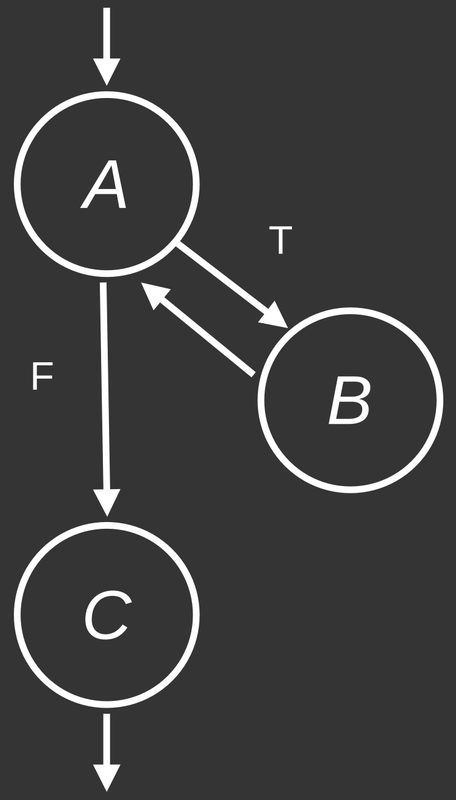


```java
public void printGreeting() {
  if (isMorning()) {
    System.out.println("Good Morning!");
  } else if (isAfternoon()) {
    System.out.println("Good Afternoon!");
    System.out.println("Lovely day for a stroll!");
  } else if (isEvening()) {
    System.out.println("Good Evening!");
  }
}
```



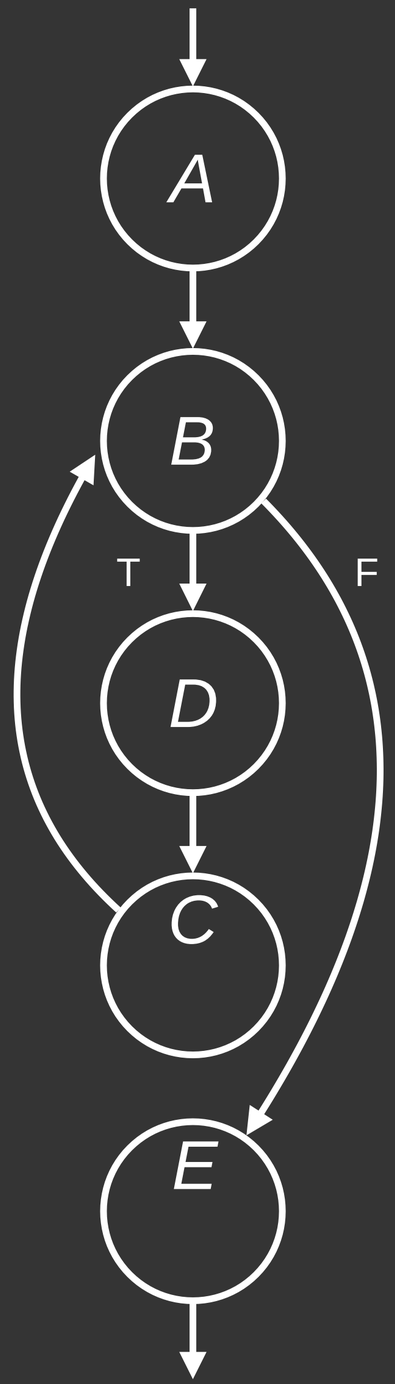
“While” Yapısı

```
while (condition) { /*A*/
  doSomething(); /* B */
}
/* C */
```

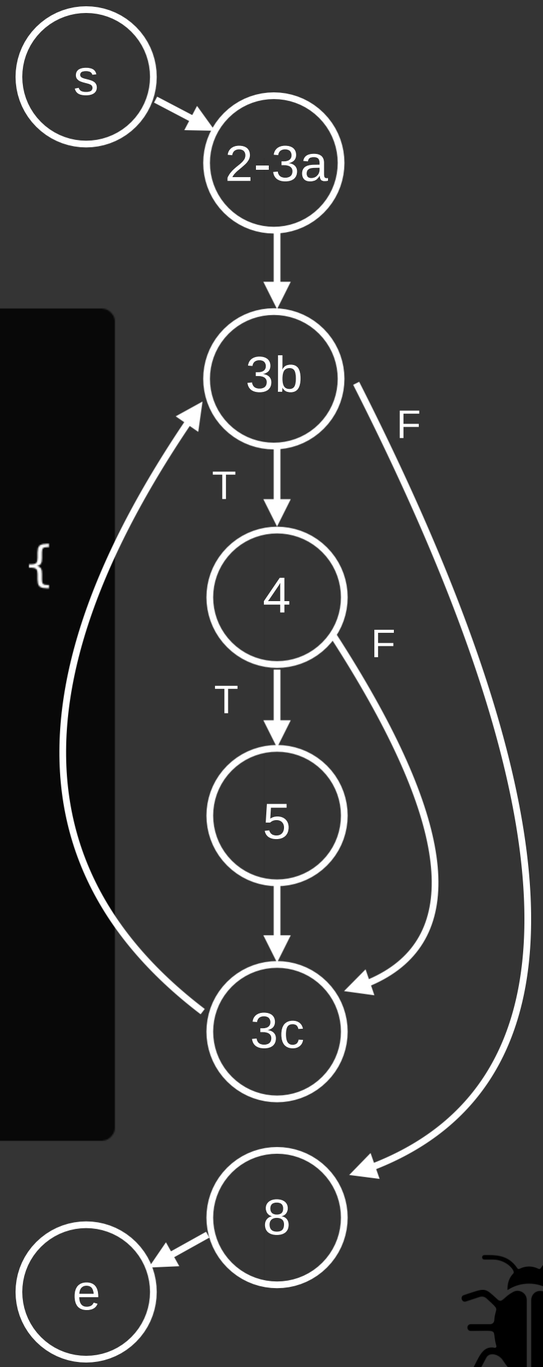


“For” Yapısı

```
for (/*A*/ int i = 0; /*B*/ i < 10; /*C*/ i ++) {
  doSomething(); /* D */
}
// E
```



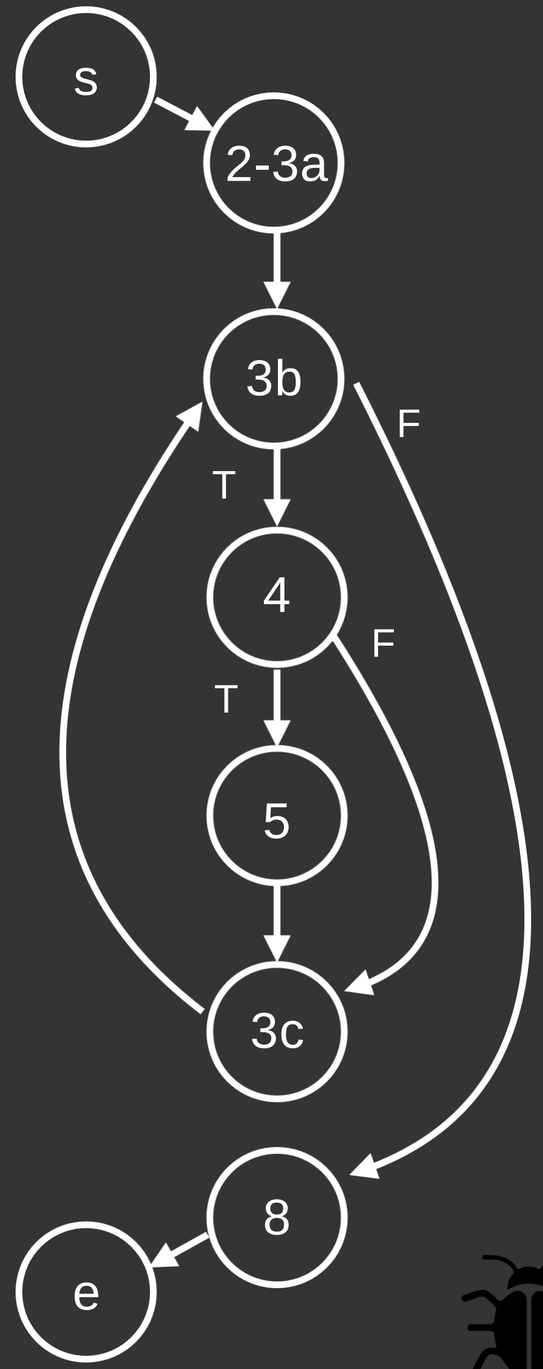
```java
public int countZeros(int[] x) {
  int count = 0;
  for (int i=0 /* 3a */; i < x.length /* 3b */; i++ /* 3c */) {
    if (x[i] == 0) {
        count ++;
    }
  }
  return count;
}
```



İfade Kapsamı

Test paketi CFG'nin tüm düğümlerini çalıştırmalıdır.

```
1 public int countZeros(int[] x) {
2   int count = 0;
3   for (int i=0 /* 3a */; i < x.length /* 3b */; i++ /* 3c */) {
4     if (x[i] == 0) {
5         count ++;
6     }
7   }
8   return count;
9 }
```



İfade Kapsamı

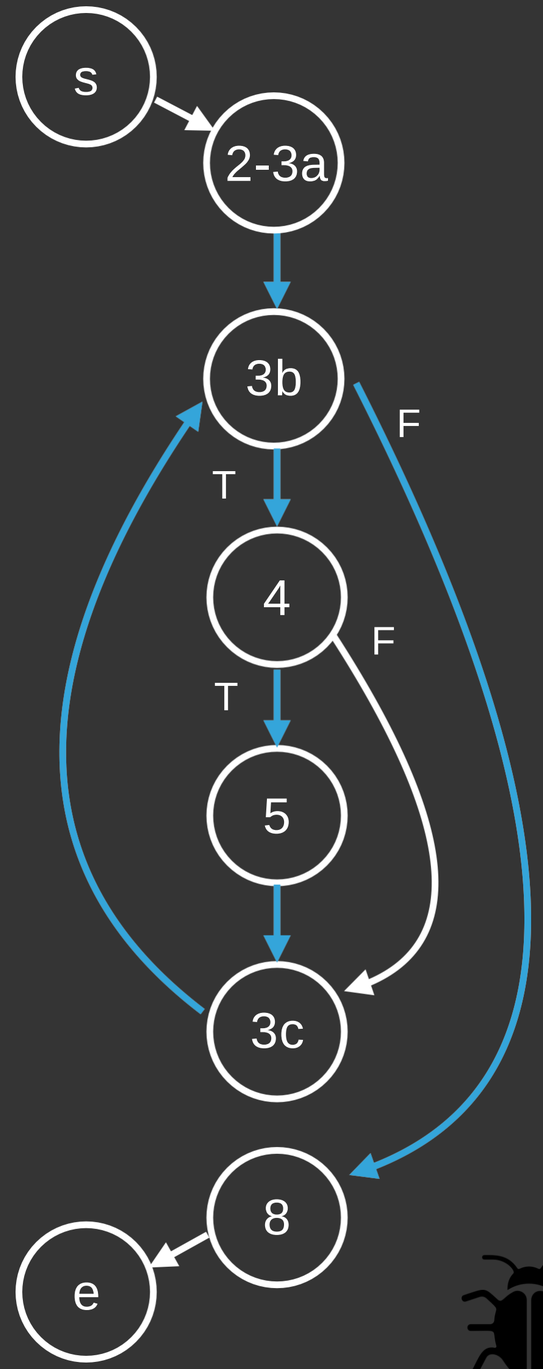
Test paketi CFG'nin tüm düğümlerini çalıştırmalıdır.

Test senaryosu x = [0] bütün düğümleri çalıştırır

Yol (path): s-3a 3b 4 5 3c 3b e

(Pratikte tüm düğümleri kapsayan birkaç test senaryosu gerekebilir)

```
1 public int countZeros(int[] x) {
2   int count = 0;
3   for (int i=0 /* 3a */; i < x.length /* 3b */; i++ /* 3c */) {
4     if (x[i] == 0) {
5         count ++;
6     }
7   }
8   return count;
9 }
```



İfade Kapsamı

Test paketi CFG'nin tüm düğümlerini çalıştırmalıdır.

Ayrıca şöyle de bilinir:

- Satır (Line) Kapsamı
- Düğüm (Node) Kapsamı
- Temel Blok (Basic Block) Kapsamı

İfade Kapsamı

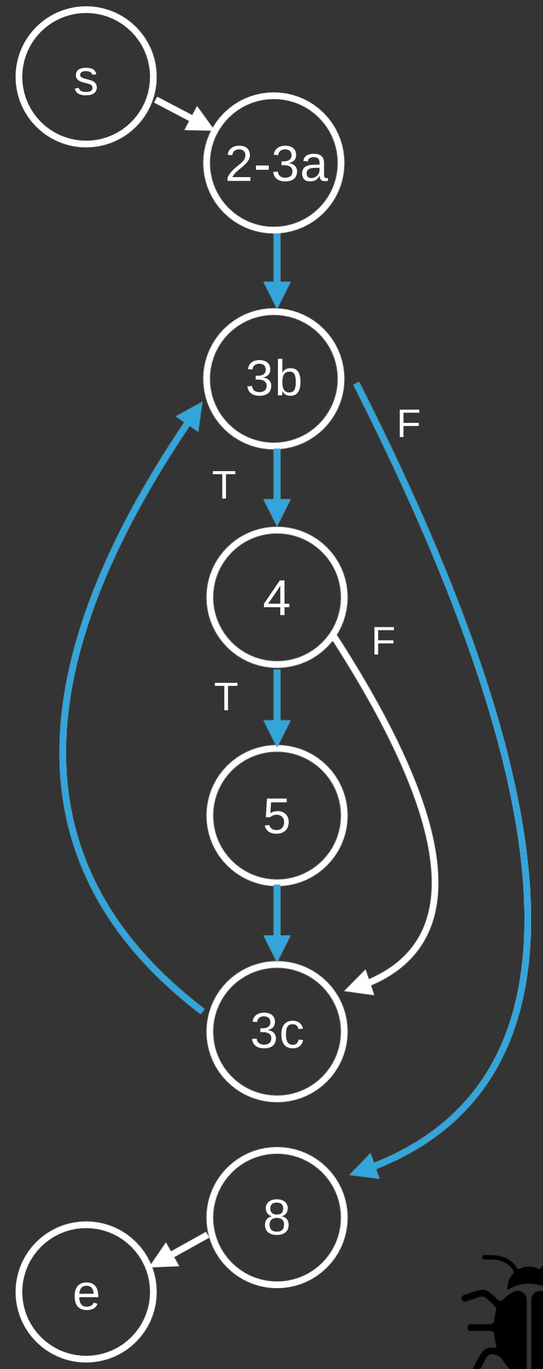
Test paketi CFG'nin tüm düğümlerini çalıştırmalıdır.

Test senaryosu x = [0] bütün düğümleri çalıştırır

Yol (path): s-3a 3b 4 5 3c 3b e

(Pratikte tüm düğümleri kapsayan birkaç test senaryosu gerekebilir)

```
1 public int countZeros(int[] x) {
2   int count = 0;
3   for (int i=0 /* 3a */; i < x.length /* 3b */; i++ /* 3c */) {
4     if (x[i] == 0) {
5         count ++;
6     }
7   }
8   return count;
9 }
```



İfade Kapsamı

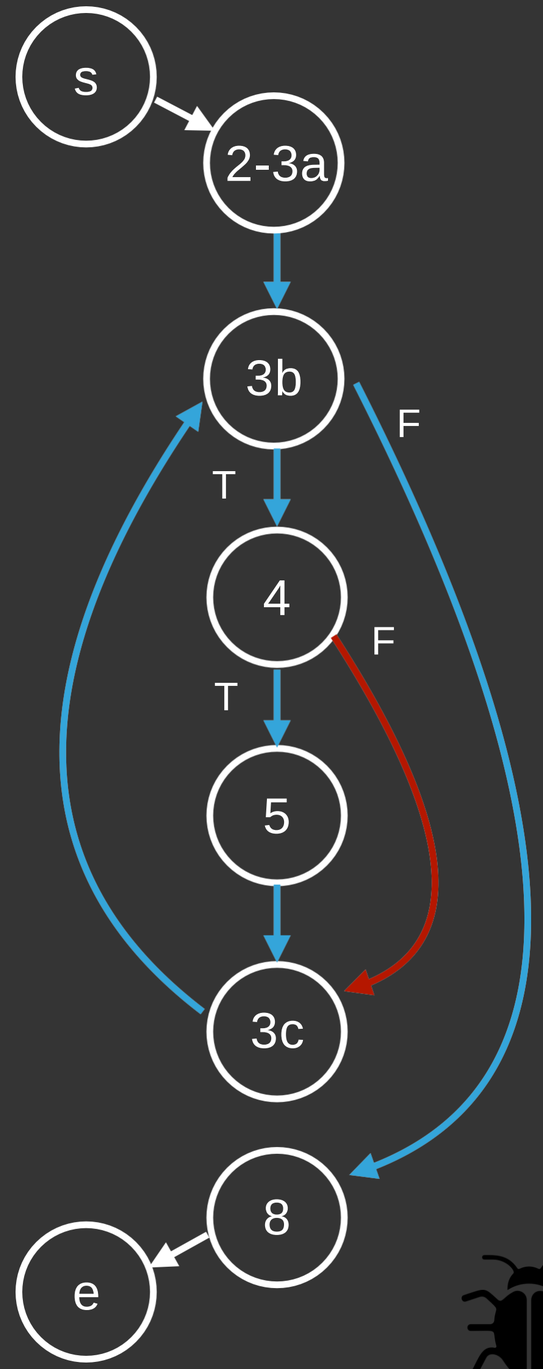
Test paketi CFG'nin tüm düğümlerini çalıştırmalıdır.

Test senaryosu x = [0] bütün düğümleri çalıştırır

Yol (path): s-3a 3b 4 5 3c 3b e

(Pratikte tüm düğümleri kapsayan birkaç test senaryosu gerekebilir)

```
1 public int countZeros(int[] x) {
2   int count = 0;
3   for (int i=0 /* 3a */; i < x.length /* 3b */; i++ /* 3c */) {
4     if (x[i] == 0) {
5         count ++;
6     }
7   }
8   return count;
9 }
```



Dal (Branch) Kapsamı

Test paketi CFG'nin tüm true/false kenarlarını (edge) yürütmelidir.

Dal (Branch) Kapsamı

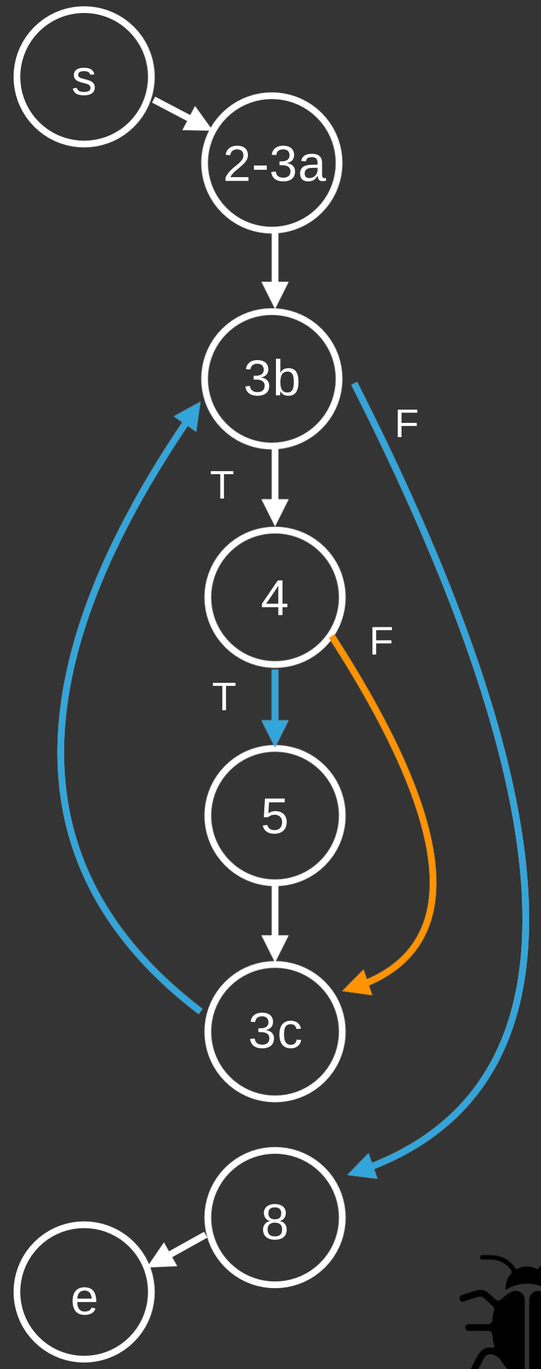
Test paketi CFG'nin tüm true/false kenarlarını (edge) yürütmelidir.

Test senaryosu x = [0] ve x = [1] bütün true/false dallarını yürütecektir.

x = [0], node 4’teki true dalı yürütürken, x = [1] false dalı yürütür

(Not: Test senaryoları x = [0, 1] olarak birleştirilebilir.)

```
1 public int countZeros(int[] x) {
2   int count = 0;
3   for (int i=0 /* 3a */; i < x.length /* 3b */; i++ /* 3c */) {
4     if (x[i] == 0) {
5         count ++;
6     }
7   }
8   return count;
9 }
```



Dal (Branch) Kapsamı

Test paketi CFG'nin tüm true/false kenarlarını (edge) yürütmelidir.

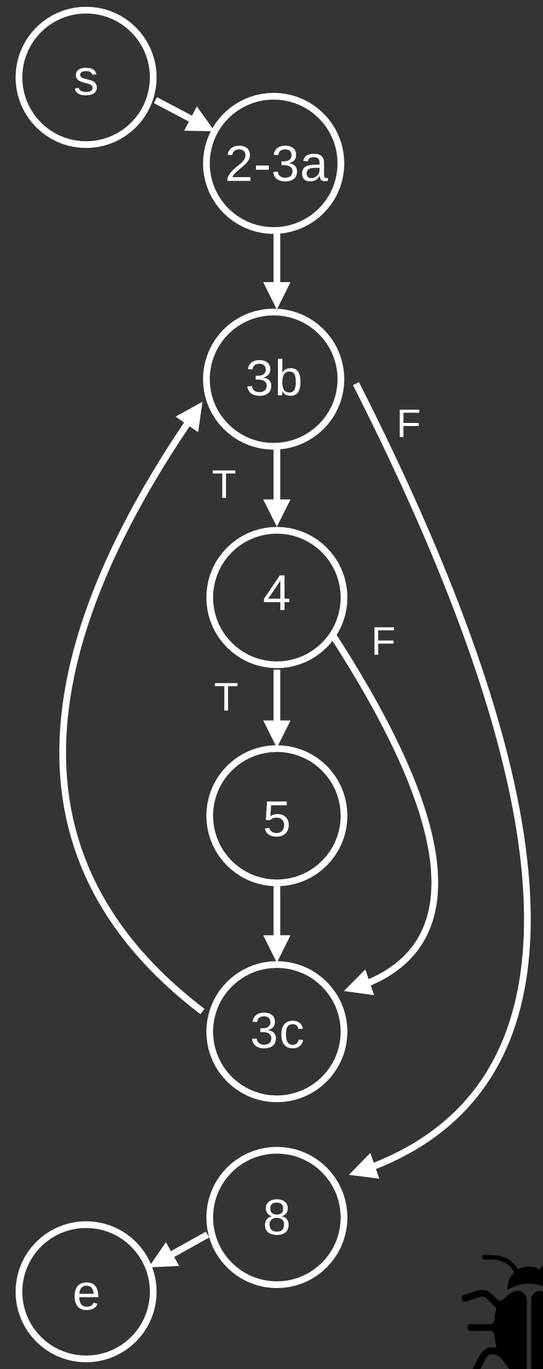
Ayrıca şöyle de bilinir:

- Karar (Decision) Kapsamı
- Öncül (Predicate) Kapsamı
- Kenar (Edge) Kapsamı

Yol (Path) Kapsamı

Test paketi CFG'deki tüm yolları yürütmelidir.

```
1 public int countZeros(int[] x) {
2   int count = 0;
3   for (int i=0 /* 3a */; i < x.length /* 3b */; i++ /* 3c */) {
4     if (x[i] == 0) {
5         count ++;
6     }
7   }
8   return count;
9 }
```



Yol (Path) Kapsamı

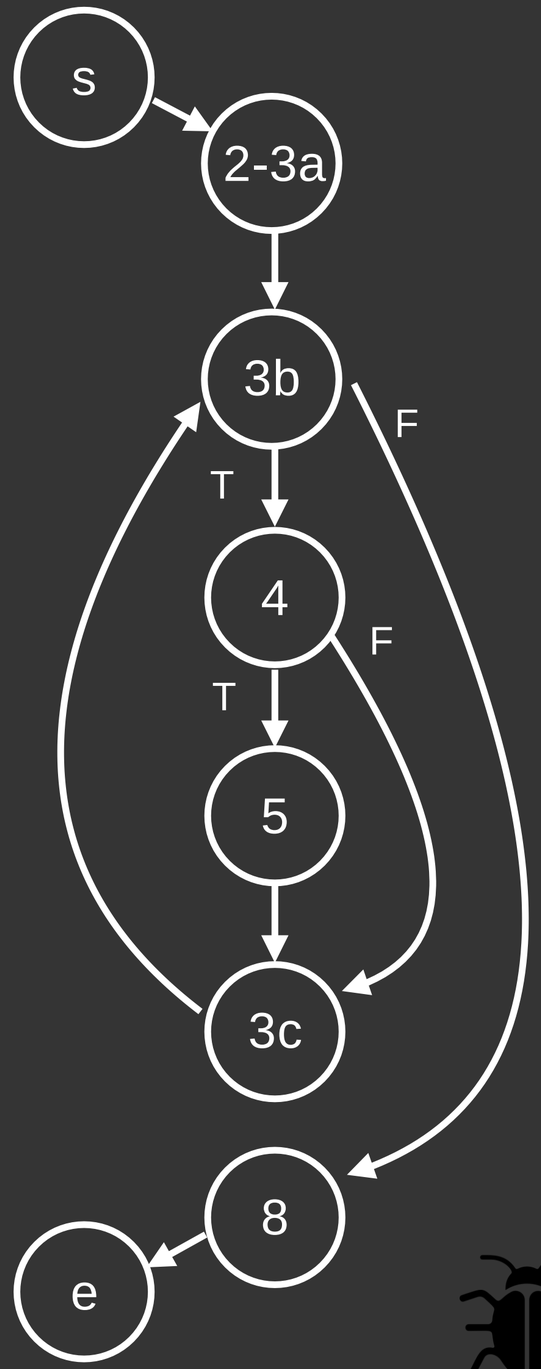
Test paketi CFG'deki tüm yolları yürütmelidir

Genellikle pratikte pek mümkün değildir.

`CountZeros'taki` olası yolların sayısı `x'in` uzunluğuna bağlıdır

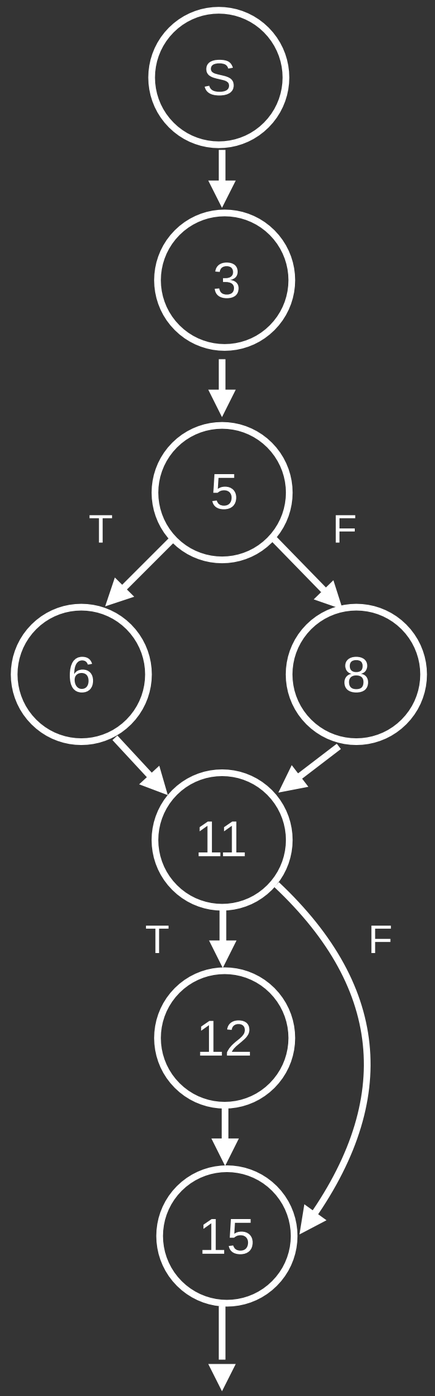
Yol Kapsamının bazı sürümleri, potansiyel olarak sonsuz sayıda yolu azaltmak için her döngünün 0, 1 veya daha fazla yürütülmesine odaklanır.

```
1 public int countZeros(int[] x) {
2   int count = 0;
3   for (int i=0 /* 3a */; i < x.length /* 3b */; i++ /* 3c */) {
4     if (x[i] == 0) {
5         count ++;
6     }
7   }
8   return count;
9 }
```



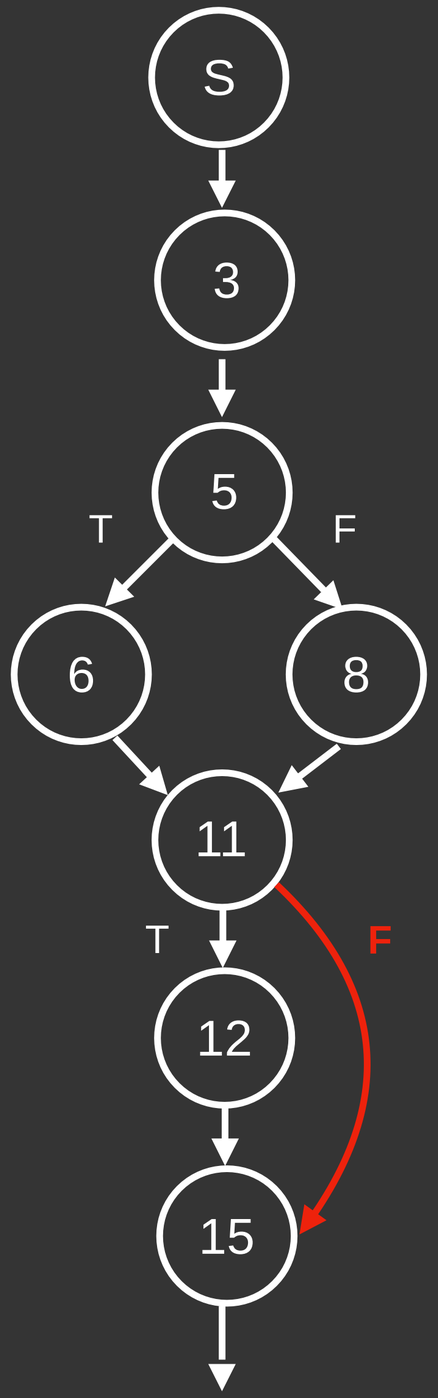
Dal Kapsamı konusunda burada gizlenen bir sorun var. Nedir?

```
1  public void testMe(int x) {
2
3    int y = 0;
4
5    if (x > 0) {
6        y = y + 1;
7    } else {
8        y = y + 2;
9    }
10
11   if (y > 0) {
12       y = y + 1;
13   }
14
15   // ...
```



Dal Kapsamı konusunda burada gizlenen bir sorun var. Nedir?

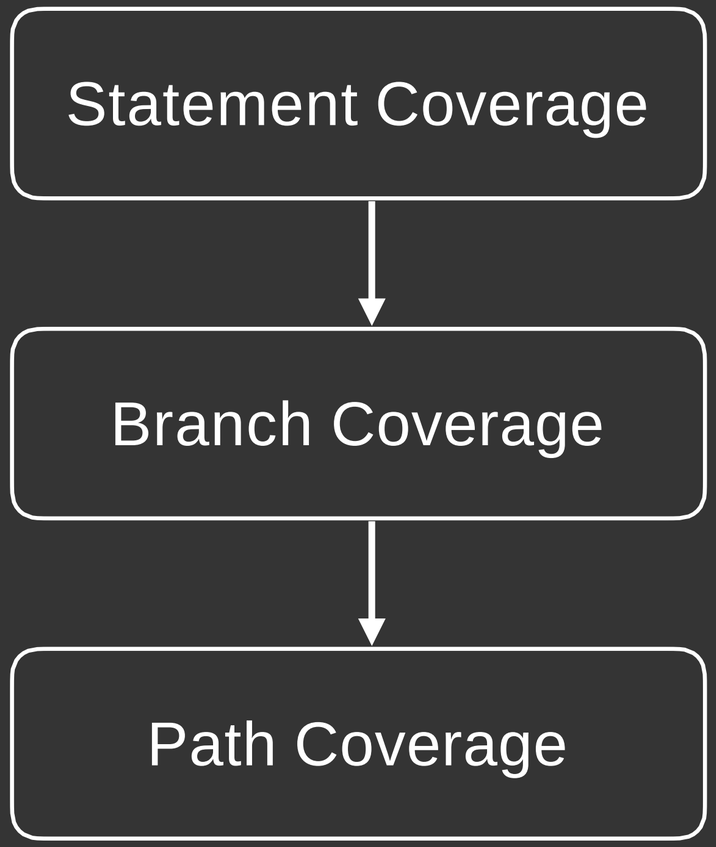
```
1  public void testMe(int x) {
2
3    int y = 0;
4
5    if (x > 0) {
6        y = y + 1;
7    } else {
8        y = y + 2;
9    }
10
11   if (y > 0) {
12       y = y + 1;
13   }
14
15   // ...
```



Kapsam Kriterleri Uygulanamaz Test Gereksinimlerine Tabidir

- Bu aynı zamanda İfade Kapsamı ile de *gerçekleşebilir*
  - Mümkün olmayan ifadeler ölü koda işaret eder.
- Dal Kapsamında gerçekleşme olasılığı *daha yüksektir*
  - Mümkün olmayan dallar, koddaki gereksiz *kararlara işaret eder*
- Yol Kapsamı ile büyük olasılıkla *gerçekleşir*
  - Uygun olmayan yollar mutlaka bir şeylerin fazlalığının sonucu değildir
  - Gerçek kodda CFG'den geçen tüm yollar yasal olarak mümkün değildir.

Yapısal (Structural) Kapsam Kriterlerinin Sınıflandırılması



Line Coverage
Node Coverage
Basic Block Coverage

Decision Coverage
Predicate Coverage
Edge Coverage

Yapısal Kapsama Ne Zaman Kullanılmalı?

- Yapısal kapsam düzeyi, kodunuzun ne kadarının test paketiniz tarafından yürütüldüğünü anlamak için yararlı bir metriktir.
- Ortak Gerekçe: kodunuzun en az bir testte en az bir kez uygulanmamış bölümlerini yayınlamak istemezsiniz
  - Bu nedenle İfade/Satır Kapsamı yaygın olarak kullanılan bir metriktir
  - Ancak Şube Kapsamı daha güçlüdür ve çok fazla ek çaba gerektirmeden elde edilebilir
  - Yol Kapsamı daha az yaygındır ve çoğu zaman inatçıdır