UNIVERSITÄT
LEIPZIG

Einführung in die Objekt-Orientierte Modellierung und Programmierung

Wintersemester 2025/2026

Dirk Zeckzer

Institut für Informatik

Teil XX

Rekursion

Rekursive Funktionen: Fakultät

Definition: $f(n) = n!$

- Berechnung iterativ:

$$f(n) = \prod_{i=1}^{n} i$$

- Beispiel

$$\begin{aligned} f(5) = \prod_{i=1}^{5} i &= 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \\ &= 2 \cdot 3 \cdot 4 \cdot 5 \\ &= 6 \cdot 4 \cdot 5 \\ &= 24 \cdot 5 \\ &= 120 \end{aligned}$$

- Berechnung rekursiv:

$$f(n) = \begin{cases} 1 & n = 1 \\ n \cdot f(n-1) & n > 1 \end{cases}$$

- Beispiel

$$\begin{aligned} f(5) &= 5 \cdot f(4) \\ &= 5 \cdot (4 \cdot f(3)) \\ &= 5 \cdot (4 \cdot (3 \cdot f(2))) \\ &= 5 \cdot (4 \cdot (3 \cdot (2 \cdot f(1)))) \\ &= 5 \cdot (4 \cdot (3 \cdot (2 \cdot 1))) \\ &= 120 \end{aligned}$$

Rekursive Funktionen: Fakultät

Definition: $f(n) = n!$

- Berechnung iterativ:

$$f(n) = \prod_{i=1}^{n} i$$

```java
public static long fakultaetForLoop(
  final long n
) {
  // Initialisierung
  long result = 1;

  // Berechnung
  for (long i = 1; i <= n; ++i) {
    result *= i;
  }

  // Ergebnis
  return result;
}
```

- Berechnung rekursiv:

$$f(n) = \begin{cases} 1 & n = 1 \\ n \cdot f(n-1) & n > 1 \end{cases}$$

```java
public static long fakultaetRecursive(
  final long n
) {
  // Fallunterscheidung
  if (n > 1) {
    // rekursiver Aufruf
    return n * fakultaetRecursive(n - 1);
  } else {
    // Basisfall
    return 1;
  }
}
```

Odd–Even

$$odd(n) = \begin{cases} true & n = 1 \\ even(n-1) & n > 1 \end{cases} \qquad even(n) = \begin{cases} false & n = 1 \\ odd(n-1) & n > 1 \end{cases}$$

Beispiel:

$$\begin{aligned} odd(5) &= even(4) \\ &= odd(3) \\ &= even(2) \\ &= odd(1) \\ &= true \end{aligned}$$

Odd–Even

$$odd(n) = \begin{cases} true & n = 1 \\ even(n-1) & n > 1 \end{cases}$$

$$even(n) = \begin{cases} false & n = 1 \\ odd(n-1) & n > 1 \end{cases}$$

```
1  public static boolean odd(
2    final int n
3  ) {
4    if (n > 1) {
5      return even(n - 1);
6    } else {
7      return true;
8    }
9  }
```

```
1  public static boolean even(
2    final int n
3  ) {
4    if (n > 1) {
5      return odd(n - 1);
6    } else {
7      return false;
8    }
9  }
```

Fibonacci-Folge

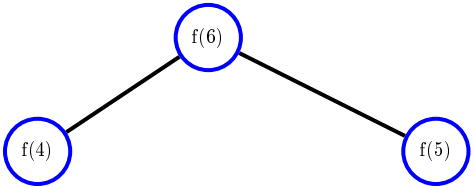
1 1 2 3 5 8 13 21 34 55 ...

$$f(n) = \begin{cases} 1 & n = 1 \\ 1 & n = 2 \\ f(n-1) + f(n-2) & n > 2 \end{cases}$$

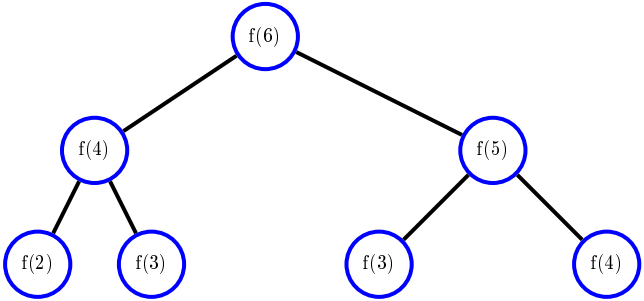
```java
public static long fibonacci(
  final long n
) {
  if (n == 1) {
    return 1;
  } else if (n == 2) {
    return 1;
  } else {
    return fibonacci(n - 1) + fibonacci(n - 2);
  }
}
```

Fibonacci-Folge

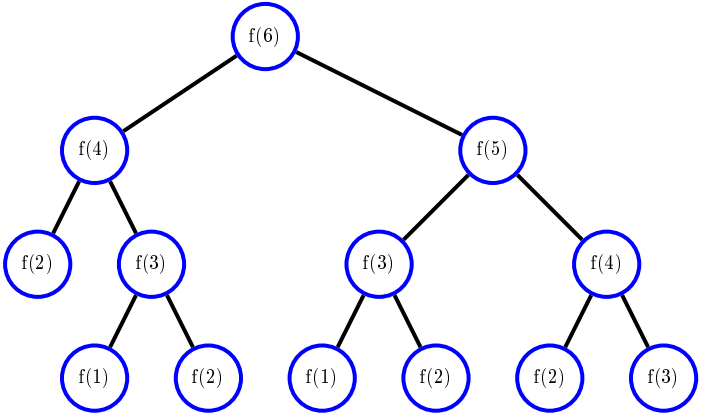
Fibonacci-Folge



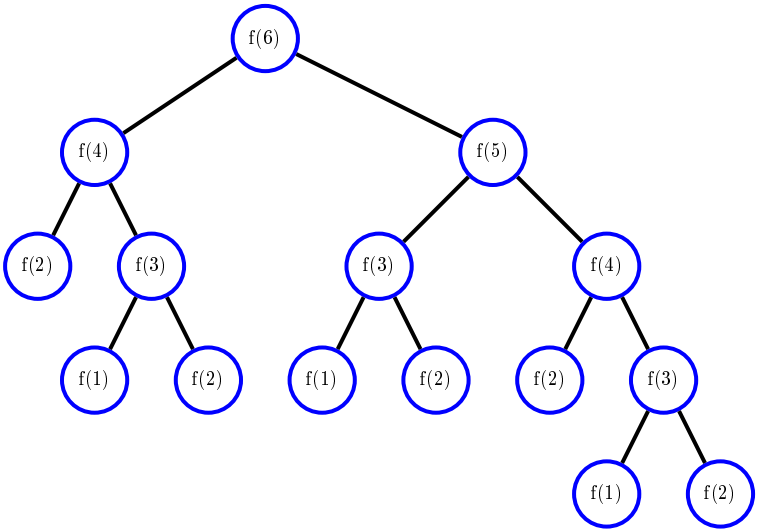
Fibonacci-Folge



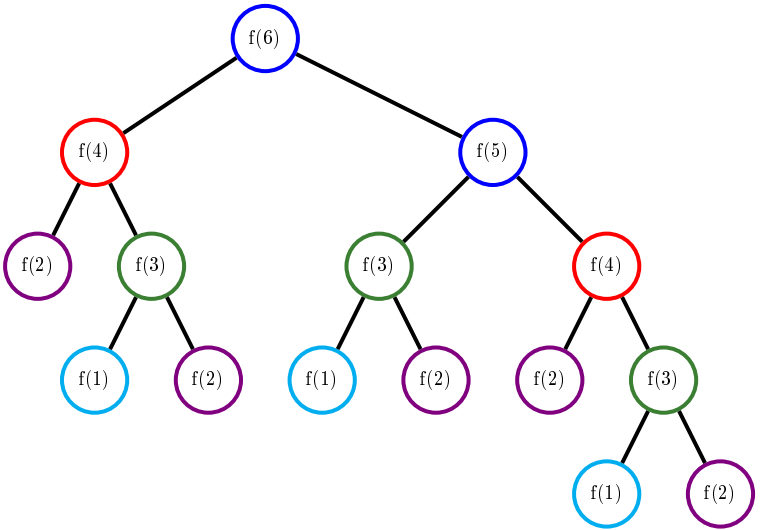
Fibonacci-Folge



Fibonacci-Folge



Fibonacci-Folge



Fibonacci-Folge

$$f(n) = \begin{cases} 1 & n = 1 \\ 1 & n = 2 \\ f(n-1) + f(n-2) & n > 2 \end{cases}$$

```java
public static long fibonacciIterative(
  final long n
) {
  if (n == 1) {
    return 1;
  }

  if (n == 2) {
    return 1;
  }

  long n1 = 1;
  long n2 = 1;
  long current = 2;
  long result = 0;

  while (current < n) {
    result = n1 + n2;
    n1 = n2;
    n2 = result;
    ++current;
  }

  return result;
}
```

Ackermannfunktion

$$
\begin{aligned}
a(0,m) &= m+1 \\
a(n+1,0) &= a(n,1) \\
a(n+1,m+1) &= a(n,a(n+1,m))
\end{aligned}
$$

```
1  public static int ackermann(
2    final int n,
3    final int m
4  ) {
5    if (n == 0) {
6      return m + 1;
7    } else if (m == 0) {
8      return ackermann(n - 1, 1);
9    } else {
10     return ackermann(n - 1, ackermann(n, m - 1));
11   }
12 }
```

Ackermannfunktion

$$
\begin{aligned}
a(0,m) &= m+1 \\
a(n+1,0) &= a(n,1) \\
a(n+1,m+1) &= a(n,a(n+1,m))
\end{aligned}
$$

Werte:

- $a(0,m) = m+1$
- $a(1,m) = ?$

$$
\begin{aligned}
a(1,0) &= a(0,1) \\
&= 2
\end{aligned}
\qquad
\begin{aligned}
a(1,1) &= a(0,a(1,0)) \\
&= a(0,2) \\
&= 3
\end{aligned}
\qquad
\begin{aligned}
a(1,2) &= a(0,a(1,1)) \\
&= a(0,3) \\
&= 4
\end{aligned}
$$

Ackermannfunktion

$$
\begin{aligned}
a(0,m) &= m+1 \\
a(n+1,0) &= a(n,1) \\
a(n+1,m+1) &= a(n,a(n+1,m))
\end{aligned}
$$

Werte:

$$
\begin{aligned}
a(2,0) &= a(1,1) \\
&= 3
\end{aligned}
\qquad
\begin{aligned}
a(3,0) &= a(2,1) \\
&= a(1,a(2,0)) \\
&= a(1,3) \\
&= 5
\end{aligned}
\qquad
\begin{aligned}
a(4,0) &= a(3,1) \\
&= a(2,a(3,0)) \\
&= a(2,5) \\
&= a(1,a(2,4)) \\
&= \ldots \\
&= a(1,11) \\
&= 13
\end{aligned}
$$

Ackermannfunktion

$$
\begin{aligned}
a(0,m) &= m+1 \\
a(n+1,0) &= a(n,1) \\
a(n+1,m+1) &= a(n,a(n+1,m))
\end{aligned}
$$

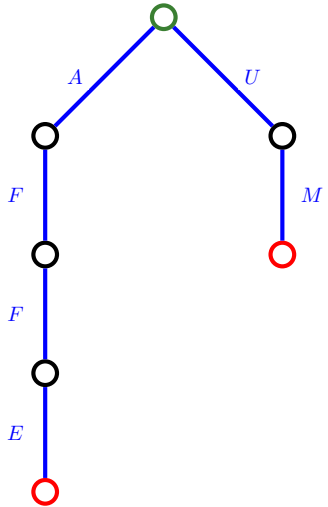
Tabelle: Werte für $a(n,m)$ [Wikipedia]

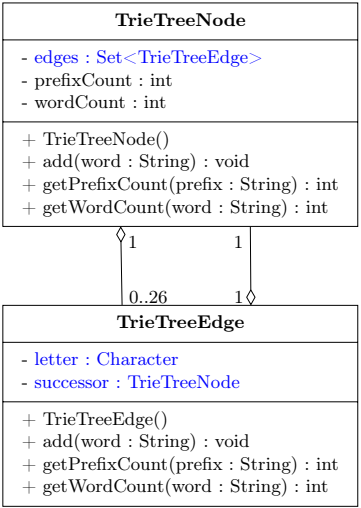
n	0	1	2	3	4	...	m
0	1	2	3	4	5	...	$m+1$
1	2	3	4	5	6	...	$m+2$
2	3	5	7	9	11	...	$2m+3$
3	5	13	29	61	125	...	$8 \cdot 2^m - 3$
4	13	65533	$2^{65536}-3$	$a(3,2^{65536}-3)$	$a(3,a(4,3))$	...	$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{}-3$
			$\approx 2 \cdot 10^{19728}$				(Potenzturm mit m+3 Zahlen)
5	65533	$a(4,65533)$	$a(4,a(5,1))$	$a(4,a(5,2))$	$a(4,a(5,3))$	...	$a(4,a(5,m-1))$
6	$a(5,1)$	$a(5,a(6,0))$	$a(5,a(6,1))$	$a(5,a(6,2))$	$a(5,a(6,3))$	...	$a(5,a(6,m-1))$

Datenstruktur Trie

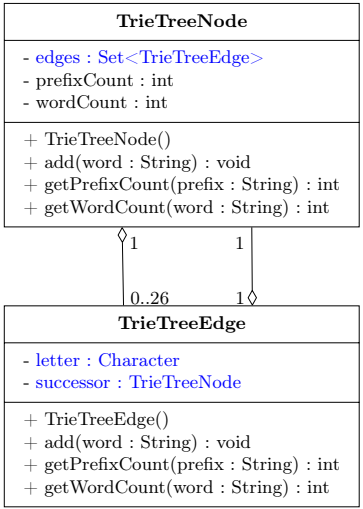
- Alphabet: $\Sigma = \{A, B, C, \ldots, Z\}$
- $\Sigma^+ = \{A, AA, \ldots, AB, ABA, \ldots, B, BA, \ldots\}$
- ϵ: das **Leere Wort**, ein Wort ohne Buchstaben
- Worte: $W = \Sigma^* = \{\epsilon, A, AA, \ldots, AB, ABA, \ldots, B, BA, \ldots\}$

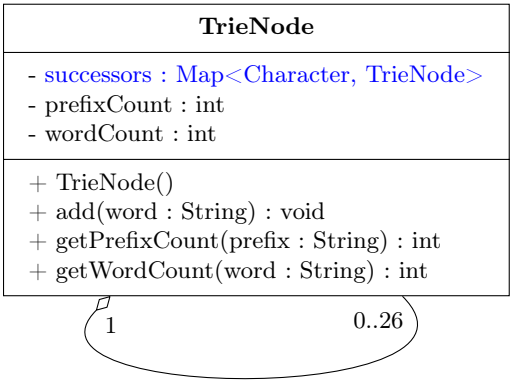
Datenstruktur Trie



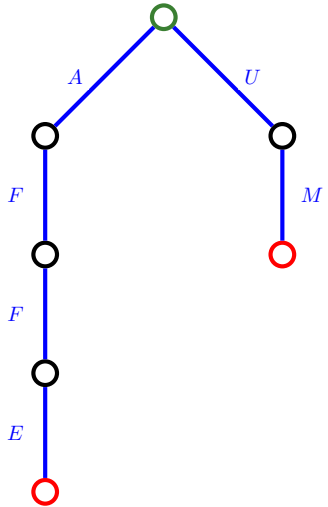


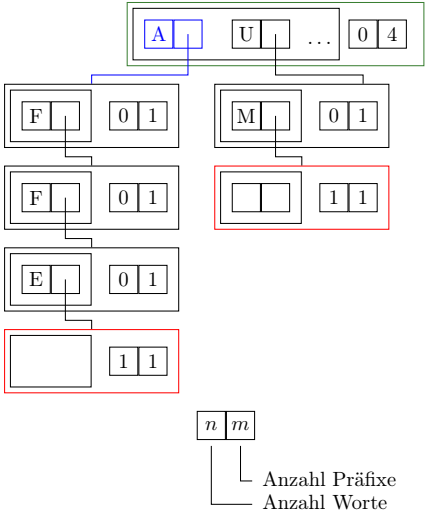
Datenstruktur Trie





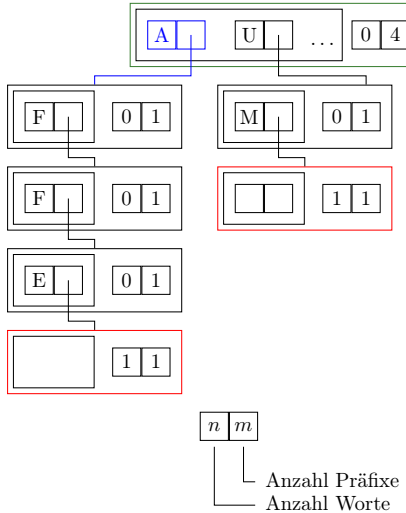
Datenstruktur Trie





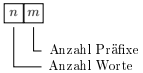
Datenstruktur Trie

- Datenstruktur zur Speicherung von Worten
- Ergebnis: Baum
- Ein Knoten im Baum entspricht einem Zeichen des Wortes



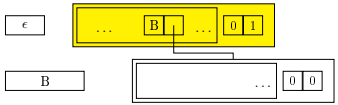
Datenstruktur Trie

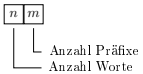
Einfügen des Wortes BANANE



Datenstruktur Trie

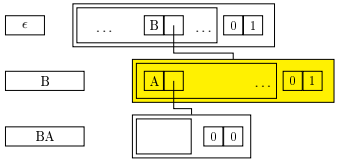
Einfügen des Wortes BANANE

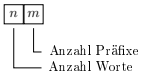




Datenstruktur Trie

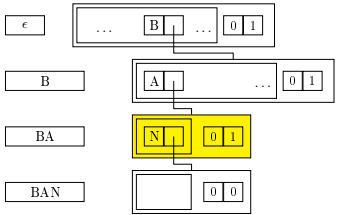
Einfügen des Wortes BANANE

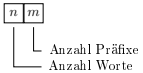




Datenstruktur Trie

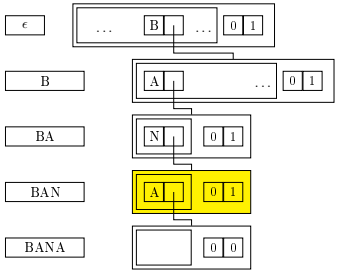
Einfügen des Wortes BANANE

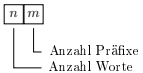




Datenstruktur Trie

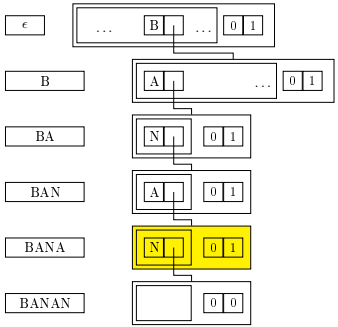
Einfügen des Wortes BANANE

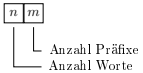




Datenstruktur Trie

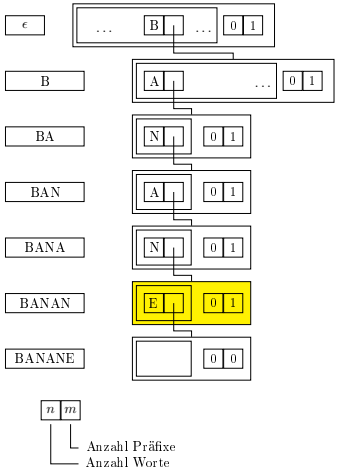
Einfügen des Wortes BANANE





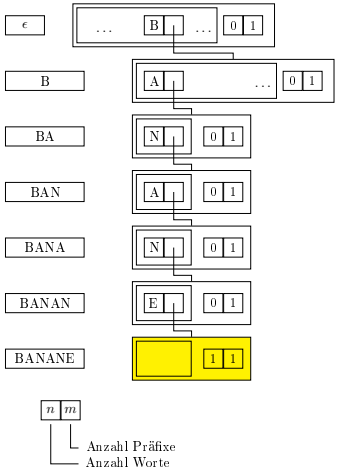
Datenstruktur Trie

Einfügen des Wortes BANANE



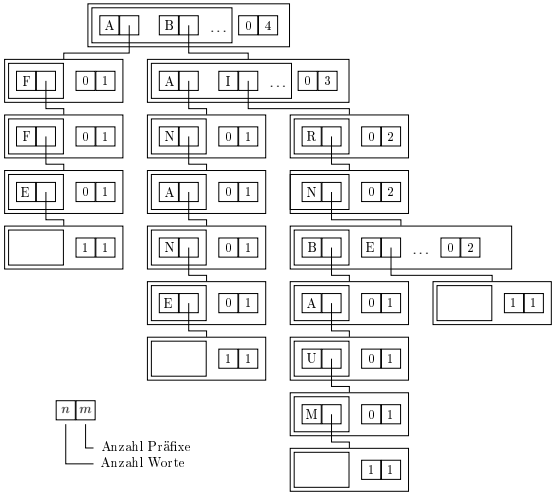
Datenstruktur Trie

Einfügen des Wortes BANANE

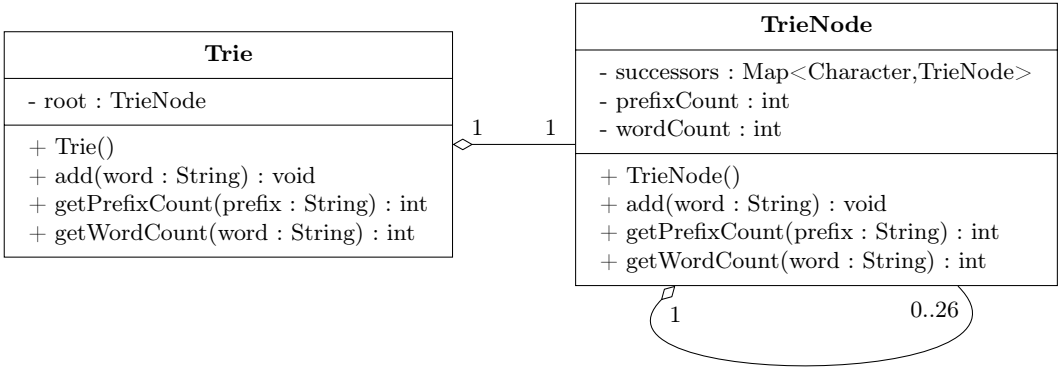


Datenstruktur Trie

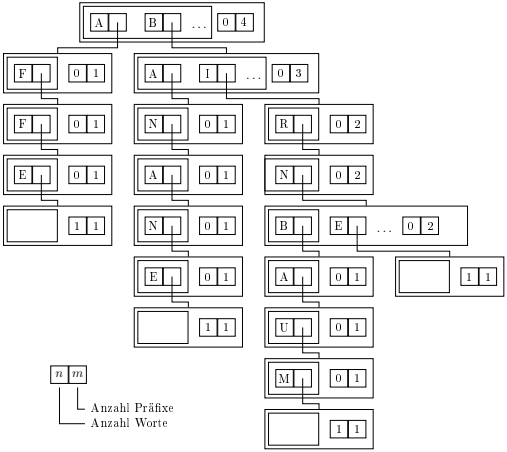
Trie nach dem Einfügen von BANANE, AFFE, BIRNE, BIRNBAUM



Datenstruktur Trie: Modell

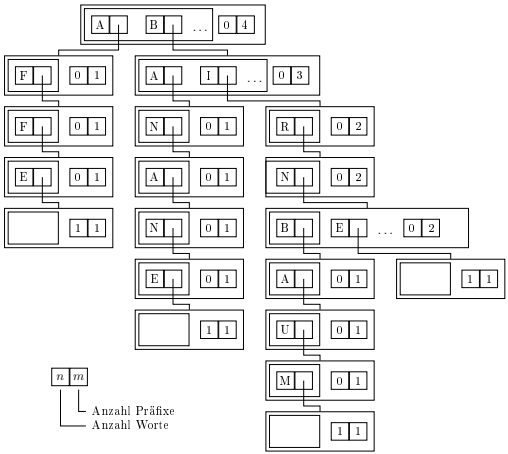


Trie



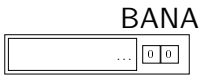

```
1   public class Trie {
2
3     private final TrieNode
4       root = new TrieNode();
5
6     public Trie() {
7     }
8
9     public void add(
10      final String word
11    ) {
12      root.add(word);
13    }
14
15    public int getPrefixCount(
16      final String prefix
17    ) {
18      return root.getPrefixCount(prefix);
19    }
20
21    public int getWordCount(
22      final String word
23    ) {
24      return root.getWordCount(word);
25    }
26  }
```

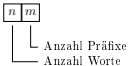
TrieNode: Attribute




```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5
6     private int prefixCount = 0;
7
8     private int wordCount = 0;
9
10    public TrieNode() {
11    }
12
13    ...
14  }
```

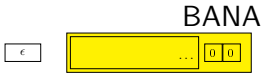
TrieNode: add





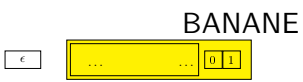

```
1   public class Trie {
2
3     private final TrieNode
4       root = new TrieNode();
5
6     public Trie() {
7     }
8
9     public void add(
10      final String word
11    ) {
12      root.add(word);
13    }
14
15    public int getPrefixCount(
16      final String prefix
17    ) {
18      return root.getPrefixCount(prefix);
19    }
20
21    public int getWordCount(
22      final String word
23    ) {
24      return root.getWordCount(word);
25    }
26  }
```

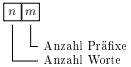
TrieNode: add




```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8   public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

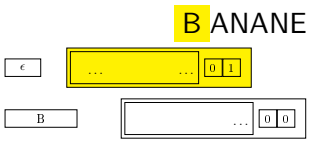
TrieNode: add

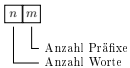





```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

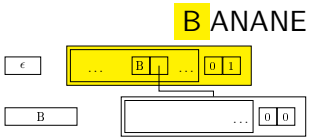
TrieNode: add

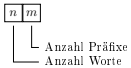





```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

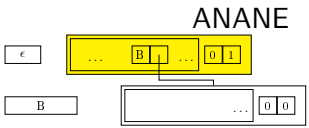
TrieNode: add





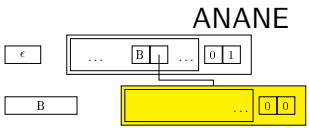

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

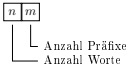
TrieNode: add




```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

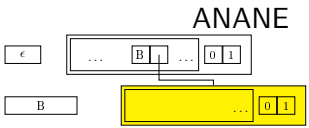
TrieNode: add

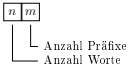





```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
8
9     public void add(
10      final String word
11    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

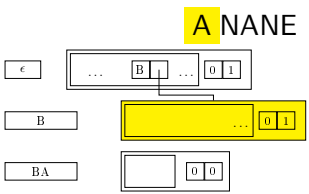
TrieNode: add





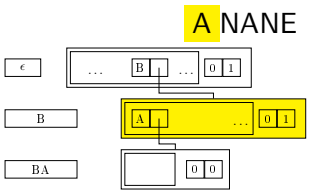

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

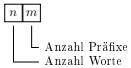
TrieNode: add




```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

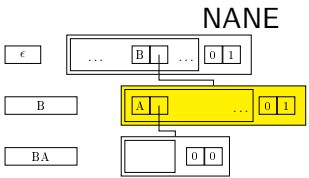
TrieNode: add

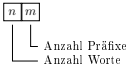





```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

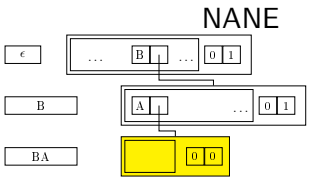
TrieNode: add

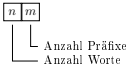





```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

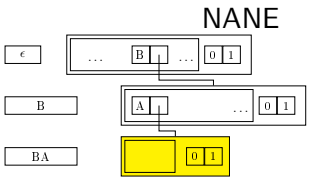
TrieNode: add

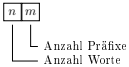





```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

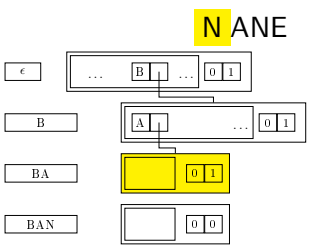
TrieNode: add





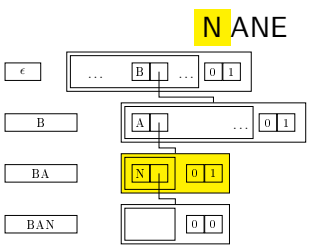

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: add



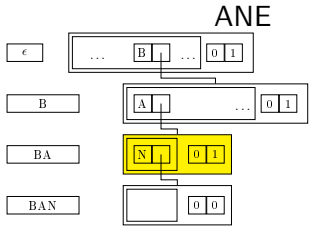

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: add



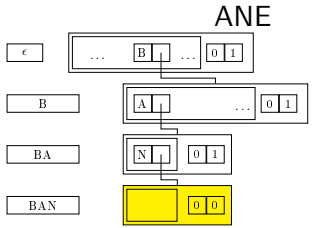

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

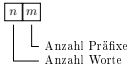
TrieNode: add




```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

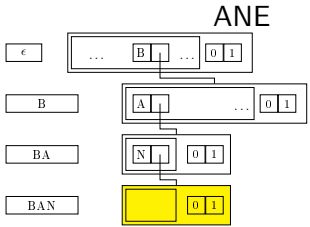
TrieNode: add

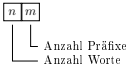





```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

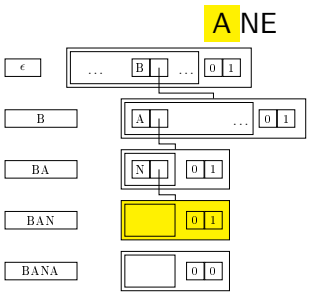
TrieNode: add





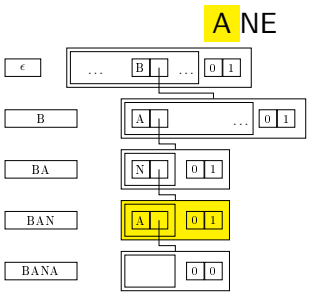

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: add



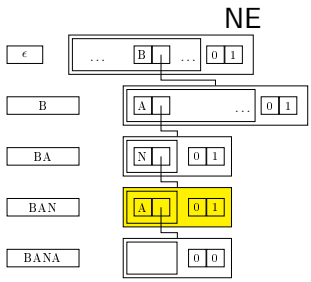

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: add



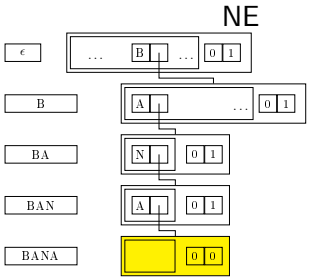

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: add



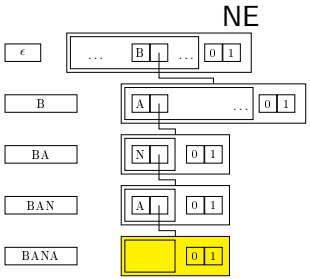

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: add



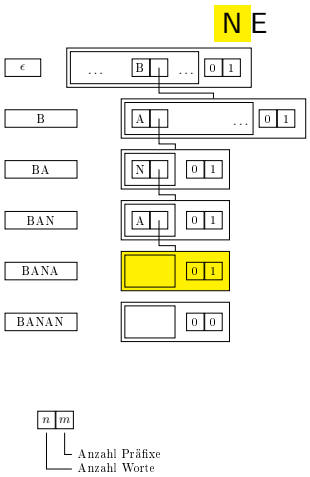

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: add



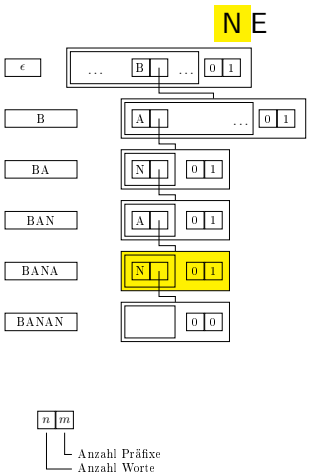

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: add



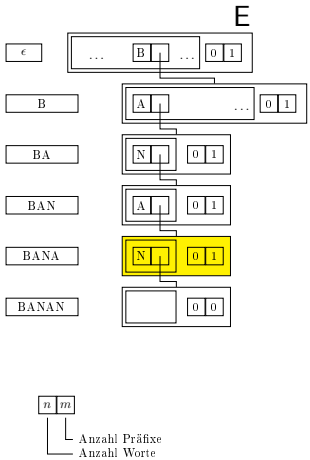

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: add



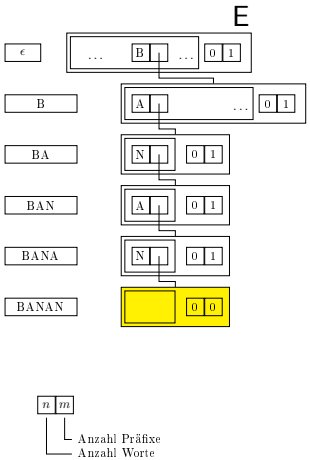

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: add



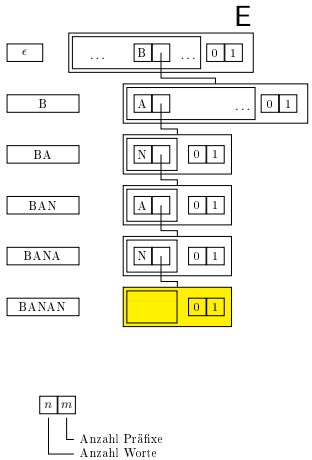

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: add



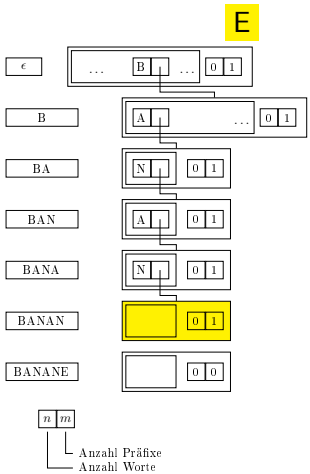

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: add



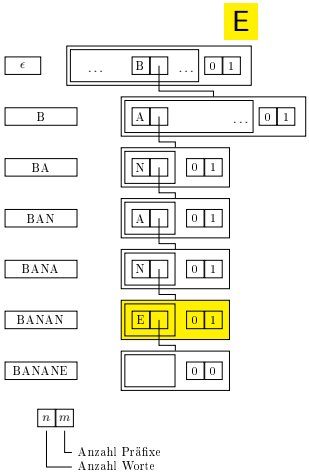

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: add



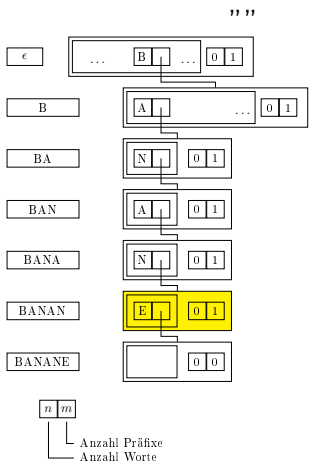

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: add



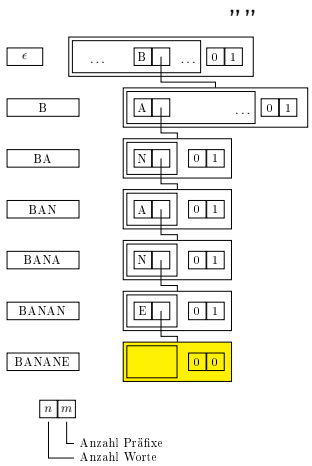

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: add



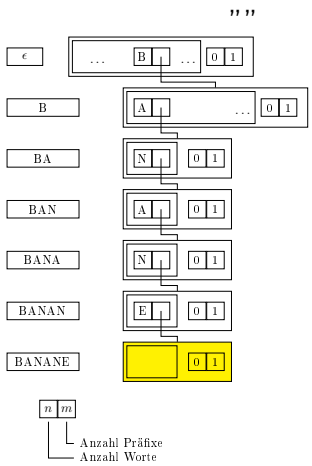

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: add



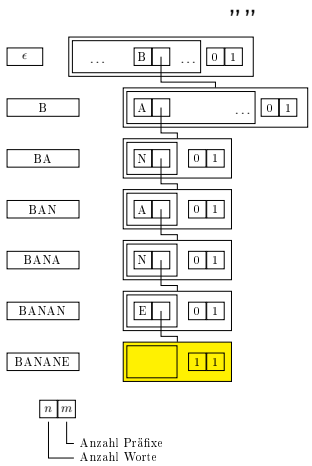

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: add



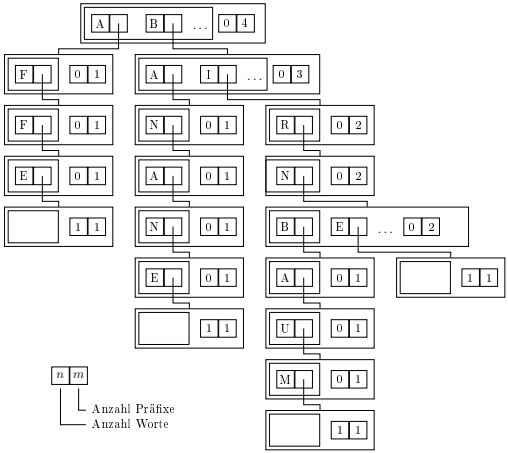

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: add



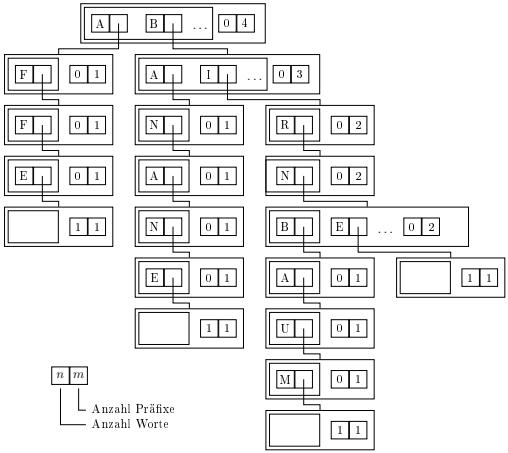

```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public void add(
9       final String word
10    ) {
11      ++prefixCount;
12      if (word.isEmpty()) {
13        ++wordCount;
14      } else {
15        final Character currentChar
16          = word.charAt(0);
17        TrieNode trieNode
18          = successors.get(currentChar);
19        if (trieNode == null) {
20          trieNode = new TrieNode();
21          successors.put(currentChar,
22                         trieNode);
23        }
24        trieNode.add(word.substring(1));
25      }
26    }
27  }
```

TrieNode: getPrefixCount




```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public int getPrefixCount(
9       final String prefix
10    ) {
11      if (prefix.isEmpty()) {
12        return prefixCount;
13      } else {
14        final Character currentChar
15          = prefix.charAt(0);
16        final TrieNode trieNode
17          = successors.get(currentChar);
18        if (trieNode == null) {
19          return 0;
20        }
21        return trieNode.getPrefixCount(
22                 prefix.substring(1)
23               );
24      }
25    }
26  }
```

TrieNode: getWordCount




```
1   public class TrieNode {
2
3     private final Map<Character, TrieNode>
4       successors = new TreeMap<>();
5     private int prefixCount = 0;
6     private int wordCount = 0;
7
8     public int getWordCount(
9       final String word
10    ) {
11      if (word.isEmpty()) {
12        return wordCount;
13      } else {
14        final Character currentChar
15          = word.charAt(0);
16        final TrieNode trieNode
17          = successors.get(currentChar);
18        if (trieNode == null) {
19          return 0;
20        }
21        return trieNode.getWordCount(
22                 word.substring(1)
23               );
24      }
25    }
26  }
```