# Teil X

## Rekursion

# Rekursive Funktionen: Fakultät

$$f(n) = n!$$

# Rekursive Funktionen: Fakultät

$$f(n) = n!$$

- iterativ:

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

# Rekursive Funktionen: Fakultät

$$f(n) = n!$$

- iterativ:

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

- rekursiv:

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

# Rekursive Funktionen: Fakultät

$$f(n) = n!$$

- iterative:

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

- rekursiv:

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

```java
public static long fakultaetForLoop(
  long n
) {
  long result = 1;
  for (long i = 1; i <= n; ++i) {
    result *= i;
  }

  return result;
}
```

# Rekursive Funktionen: Fakultät

$$f(n) = n!$$

- iterativ:

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

- rekursiv:

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

```java
public static long fakultaetForLoop(
  long n
) {
  long result = 1;
  for (long i = 1; i <= n; ++i) {
    result *= i;
  }

  return result;
}
```

```java
public static long fakultaetWhileLoop(
  long n
) {
  if (n > 0) {
    long result = n;
    long factor = n - 1;

    while (factor > 0) {
      result *= factor;
      - -factor;
    }
    return result;
  } else {
    return 1;
  }
}
```

# Rekursive Funktionen: Fakultät

$$f(n) = n!$$

- iterativ:

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

- rekursiv:

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

```java
public static long fakultaetForLoop(
  long n
) {
  long result = 1;
  for (long i = 1; i <= n; ++i) {
    result *= i;
  }

  return result;
}
```

```java
public static long fakultaetRecursive(
  long n
) {
  if (n > 1) {
    return n * fakultaetRecursive(n - 1);
  } else {
    return 1;
  }
}
```

# Odd–Even

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

# Odd–Even

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

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

```java
public static boolean odd(
  int n
) {
  if (n == 0) {
    return false;
  } else {
    return even(n - 1);
  }
}
```

```java
public static boolean even(
  int n
) {
  if (n == 0) {
    return true;
  } else {
    return odd(n - 1);
  }
}
```

# Fibonacci-Folge

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

# 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 fibonacci(
  long n
) {
  if (n == 1) {
    return 1;
  } else if (n == 2) {
    return 1;
  } else {
    return fibonacci(n - 1) + fibonacci(n - 2);
  }
}
```

# 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(
  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

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

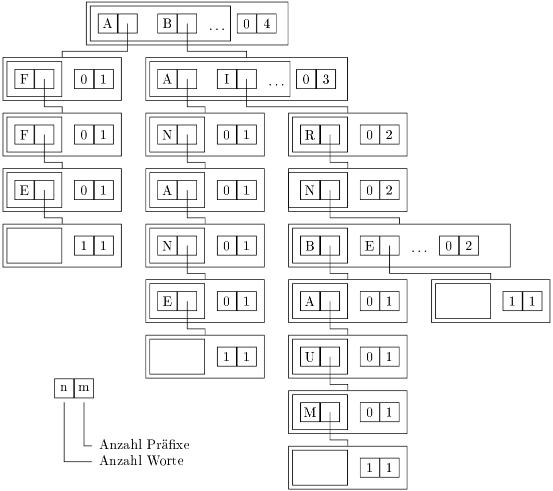
# Ackermannfunktion

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

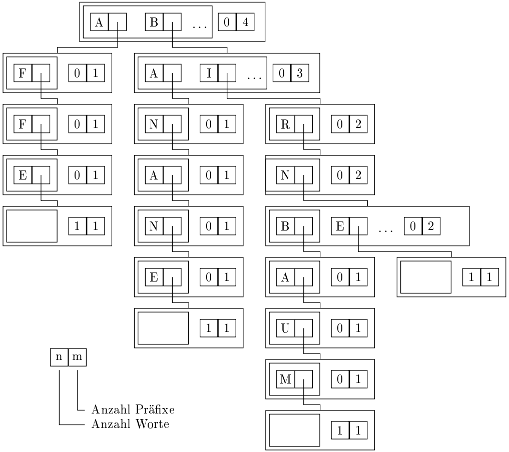
```
public static int ackermann(
  int n,
  int m
) {
  if (n == 0) {
    return m + 1;
  } else if (m == 0) {
    return ackermann(n - 1, 1);
  } else {
    return ackermann(n - 1, ackermann(n, m - 1));
  }
}
```

# Datenstruktur Trie

Beispiel eines Tries:

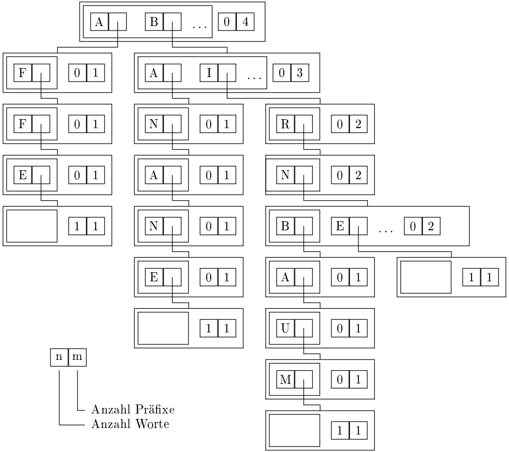


# Trie



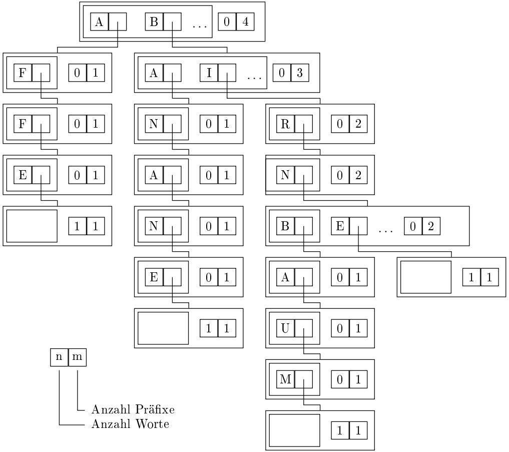
```java
public class Trie {

  private final TrieNode
    root = new TrieNode();

  public void add(
    String word
  ) {
    root.add(word);
  }

  public int getPrefixCount(
    String prefix
  ) {
    return root.getPrefixCount(prefix);
  }

  public int getWordCount(
    String word
  ) {
      return root.getWordCount(word);
  }
}
```

# TrieNode: Attribute



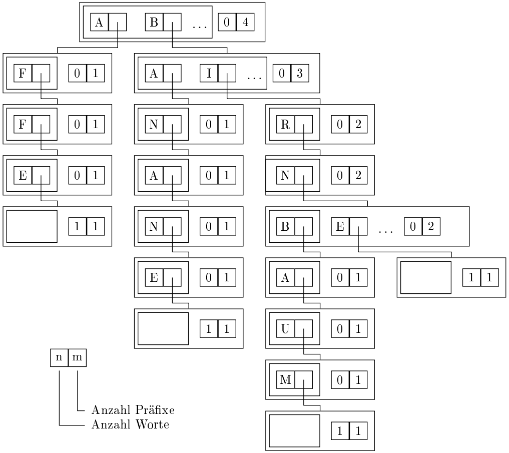
```java
public class TrieNode {

  private final Map<Character, TrieNode>
    successors = new TreeMap<>();

  private int prefixCount = 0;

  private int wordCount = 0;

  public TrieNode() {
  }

  ...
}
```

# TrieNode: add



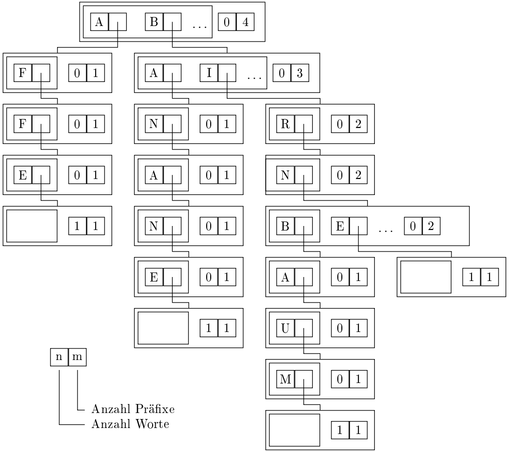
```java
public class TrieNode {

 private final Map<Character, TrieNode>
  successors = new TreeMap<>();
 private int prefixCount = 0;
 private int wordCount = 0;

 public void add(
  String word
 ) {
  ++prefixCount;
  if (word.isEmpty()) {
   ++wordCount;
  } else {
   Character currentChar = word.charAt
   (0);
   TrieNode trieNode
    = successors.get(currentChar);
   if (trieNode == null) {
    trieNode = new TrieNode();
    successors.put(currentChar,
   trieNode);
   }
   trieNode.add(word.substring(1));
  }
 }
}
```

# TrieNode: getPrefixCount



```java
public class TrieNode {

  private final Map<Character, TrieNode>
    successors = new TreeMap<>();
  private int prefixCount = 0;
  private int wordCount = 0;

  public int getPrefixCount(
    String prefix
  ) {
    if (prefix.isEmpty()) {
      return prefixCount;
    } else {
      Character currentChar = prefix.
     charAt(0);
      TrieNode trieNode
        = successors.get(currentChar);
      if (trieNode == null) {
        return 0;
      }
      return trieNode.getPrefixCount(
               prefix.substring(1)
             );
    }
  }
```

# TrieNode: getWordCount



```
public class TrieNode {

  private final Map<Character, TrieNode>
    successors = new TreeMap<>();
  private int prefixCount = 0;
  private int wordCount = 0;

  public int getWordCount(
    String word
  ) {
    if (word.isEmpty()) {
      return wordCount;
    } else {
      Character currentChar = word.charAt
     (0);
      TrieNode trieNode
        = successors.get(currentChar);
      if (trieNode == null) {
        return 0;
      }
      return trieNode.getWordCount(
               word.substring(1)
             );
    }
  }
}
```