2. Übung

Einführung in die Bioinformatik I, 1. Teil
Wintersemester 2019/2020

Aufgabe 1:

Betrachten Sie die Algorithmen ggT_1 und ggT_2 zur Berechnung des größten gemeinsamen Teilers zweier natürlicher Zahlen a,$b > 0$:

```
ggT_1(a, b)
  WHILE a ≠ 0 DO        (*)
      IF a < b THEN
         swap(a,b);
      END IF;
      a ← a MOD b;
  END WHILE;
  RETURN b;
END.
```

```
ggT_2(a, b)
  IF a = b                    (*)
     THEN RETURN a;
  END IF;
  IF a < b
     THEN RETURN ggT_2(a, b-a);
     ELSE RETURN ggT_2(a-b, b);
  END IF;
END.
```

Dabei bedeutet `swap(a,b)`, dass die Inhalte der Variablen a und b getauscht werden.

Wie oft wird der Vergleich in der Zeile `(*)` beim Aufruf der Algorithmen mit den Werten $(21, 8)$, $(64, 28)$ und $(238, 68)$ ausgeführt?

	(21,8)	(64,28)	(238,68)
ggT_1			
ggT_2			

Aufgabe 1:

Betrachten Sie die Algorithmen ggT_1 und ggT_2 zur Berechnung des größten gemeinsamen Teilers zweier natürlicher Zahlen a,$b > 0$:

```
ggT_1(a, b)
  WHILE a ≠ 0 DO        (*)
      IF a < b THEN
         swap(a,b);
      END IF;
      a ← a MOD b;
  END WHILE;
  RETURN b;
END.
```

```
ggT_2(a, b)
  IF a = b                    (*)
     THEN RETURN a;
  END IF;
  IF a < b
     THEN RETURN ggT_2(a, b-a);
     ELSE RETURN ggT_2(a-b, b);
  END IF;
END.
```

Dabei bedeutet `swap(a,b)`, dass die Inhalte der Variablen a und b getauscht werden.

Wie oft wird der Vergleich in der Zeile `(*)` beim Aufruf der Algorithmen mit den Werten $(21, 8)$, $(64, 28)$ und $(238, 68)$ ausgeführt?

	(21,8)	(64,28)	(238,68)
ggT_1	6	4	3
ggT_2	7	7	5

Aufgabe 1:

```
ggT_1(a, b)
  WHILE a ≠ 0 DO        (*)
      IF a < b THEN
         swap(a,b);
      END IF;
      a ← a MOD b;
  END WHILE;
  RETURN b;
END.
```

1. Durchlauf

a = 238
b = 68

WHILE 238 ≠ 0 DO ✓

IF 238 < 68 THEN ✗

a ← 238 MOD 68
a ← 34

2. Durchlauf

a = 34
b = 68

WHILE 34 ≠ 0 DO ✓

IF 34 < 68 THEN ✓
swap(a,b)

a ← 68 MOD 34
a ← 0

3. Durchlauf

a = 0
b = 34

WHILE 0 ≠ 0 DO ✗

RETURN 34

Aufgabe 1:

```
ggT_2(a, b)
  IF a = b                    (*)
     THEN RETURN a;
  END IF;
  IF a < b
     THEN RETURN ggT_2(a, b-a);
     ELSE RETURN ggT_2(a-b, b);
  END IF;
END.
```

1. Durchlauf	2. Durchlauf	3. Durchlauf	4. Durchlauf	5. Durchlauf
a = 238	a = 170	a = 102	a = 34	a = 34
b = 68	b = 68	b = 68	b = 68	b = 34
IF 238 = 68 ✗	IF 170 = 68 ✗	IF 102 = 68 ✗	IF 34 = 68 ✗	IF 34 = 34 ✓
IF 238 < 68 ✗	IF 170 < 68 ✗	IF 102 < 68 ✗	IF 34 < 68 ✓	RETURN 34
RETURN ggT_2(238-68, 68)	RETURN ggT_2(170-68, 68)	RETURN ggT_2(102-68, 68)	RETURN ggT_2(34, 68-34)	

Aufgabe 2:

Wenden Sie den “einfachen Textsuche”-Algorithmus auf die Pattern (Muster) `issip`, `ssi`, `sis` und den Text `Mississippi` an!
Wie viele Buchstabenvergleiche werden jeweils benötigt?

Aufgabe 2:

Wenden Sie den “einfachen Textsuche”-Algorithmus auf die Pattern (Muster) `issip`, `ssi`, `sis` und den Text `Mississippi` an!
Wie viele Buchstabenvergleiche werden jeweils benötigt?

Es werden für alle drei Patterns jeweils 15 Vergleiche von Einzelbuchstaben gemacht.

```
                    Text =  M  i  s  s  i  s  s  i  p  p  i
                            x
#Vergleiche =  1            i  s  s  i  p
                               *  *  *  *  x
#Vergleiche =  5               i  s  s  i  p
                                  x
#Vergleiche =  1               i  s  s  i  p
                                     x
#Vergleiche =  1               i  s  s  i  p
                                     *  *  *  *  *
#Vergleiche =  5               i  s  s  i  p
                                        x
#Vergleiche =  1               i  s  s  i  p
                                           x
#Vergleiche =  1               i  s  s  i  p
```

Aufgabe 3:

Geben Sie *alle* Substrings von `Mississippi` an!
Wie viele sind es?

Aufgabe 3:

Geben Sie *alle* Substrings von `Mississippi` an!
Wie viele sind es?

Substrings der Länge 0: ε

Substrings der Länge 1: `M, i, s, p`

Substrings der Länge 2: `Mi, is, ss, si, ip, pp, pi`

Substrings der Länge 3: `Mis, iss, ssi, sis, sip, ipp, ppi`

Substrings der Länge 4: `Miss, issi, ssis, siss, ssip, sipp, ippi`

Substrings der Länge 5: `Missi, issis, ssiss, sissi, issip, ssipp, sippi`

Substrings der Länge 6: `Missis, ississ, ssissi, sissip, issipp, ssippi`

Substrings der Länge 7: `Mississ, ississi, ssissip, sissipp, issippi`

Substrings der Länge 8: `Mississi, ississip, ssissipp, sissippi`

Substrings der Länge 9: `Mississip, ississipp, ssissippi`

Substrings der Länge 10: `Mississipp, ississippi`

Substrings der Länge 11: `Mississippi`

Insgesamt gibt es
54 Substrings.

Aufgabe 4:

Bei der Anwendung eines Textsuche-Algorithmus auf einen Text $T = t_1 \dots t_n$ und ein Pattern $P = p_1 \dots p_m$, $m < n$, passiere folgendes: Der Algorithmus untersucht, ob P in T ab Position i enthalten ist, und stellt das erste Mismatch an Position j in P fest, d. h. $t_{i+j-1} \neq p_j$, wobei $j > 1$. Daraufhin wird P um eine Position nach rechts geschoben und festgestellt, dass das Pattern an dieser Stelle im Text vorkommt. Was können Sie über das Pattern P sagen?

- P hat eine Länge von 2 oder mehr.
- Die Zeichen p_1, …, p_{j-1} sind alle identisch, jedoch nicht mit p_j.

Aufgabe 4:

Bei der Anwendung eines Textsuche-Algorithmus auf einen Text $T = t_1 \dots t_n$ und ein Pattern $P = p_1 \dots p_m$, $m < n$, passiere folgendes: Der Algorithmus untersucht, ob P in T ab Position i enthalten ist, und stellt das erste Mismatch an Position j in P fest, d. h. $t_{i+j-1} \neq p_j$, wobei $j > 1$. Daraufhin wird P um eine Position nach rechts geschoben und festgestellt, dass das Pattern an dieser Stelle im Text vorkommt. Was können Sie über das Pattern P sagen?

- P hat eine Länge von 2 oder mehr.
- Die Zeichen p_1, …, p_{j-1} sind alle identisch, jedoch nicht mit p_j.

Situation beim Mismatch an Position j in *P*.

$T =$	t_1	t_2	t_3	…	t_{i-1}	t_i	t_{i+1}	t_{i+2}	…	t_{i+j-1}	…	t_n	
						*	*	*	*	x	?		
$P =$						p_1	p_2	p_3	…	p_j	…	p_m	

Situation nach dem Verschieben.

$T =$	t_1	t_2	t_3	…	t_{i-1}	t_i	t_{i+1}	t_{i+2}	t_{i+3}	…	t_{i+j-1}	t_{i+j}	…	…	t_n
							*	*	*	*	*	*	*	*	
$P =$							p_1	p_2	p_3	…	p_{j-1}	p_j	…	p_m	

Die Fibonacci-Zahlenfolge

Die Fibonacci-Folge ist die unendliche Folge natürlicher Zahlen, die mit zweimal der Zahl 1 beginnt. Im Anschluss ergibt jeweils die Summe zweier aufeinanderfolgender Zahlen die unmittelbar danach folgende Zahl: 1, 1, 2, 3, 5, 8, 13, 21, 34, …

Der folgende Algorithmus kann eine beliebige *n*-te Stelle der Fibonacci-Folge mit n > 0 berechnen:

```
fibo(n)
   IF n = 1 OR n = 2 THEN      (*)
      ERGEBNIS ← 1;
   ELSE
      ERGEBNIS ← fibo(n-1) + fibo(n-2);
   END IF;

   RETURN ERGEBNIS;
```

Wie oft wird der Vergleich in der Zeile (*) beim Aufruf des Algorithmus mit dem Wert (5) ausgeführt?

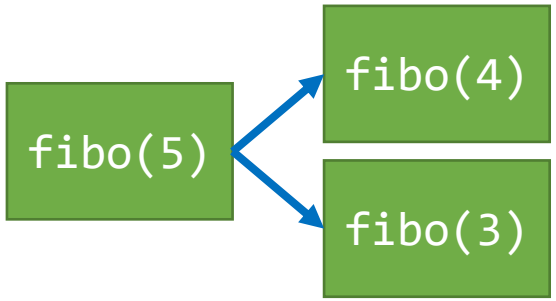
Die Fibonacci-Zahlenfolge

```
fibo(n)
   IF n = 1 OR n = 2 THEN      (*)
      ERGEBNIS ← 1;
   ELSE
      ERGEBNIS ← fibo(n-1) + fibo(n-2);
   END IF;

   RETURN ERGEBNIS;
```

fibo(5)

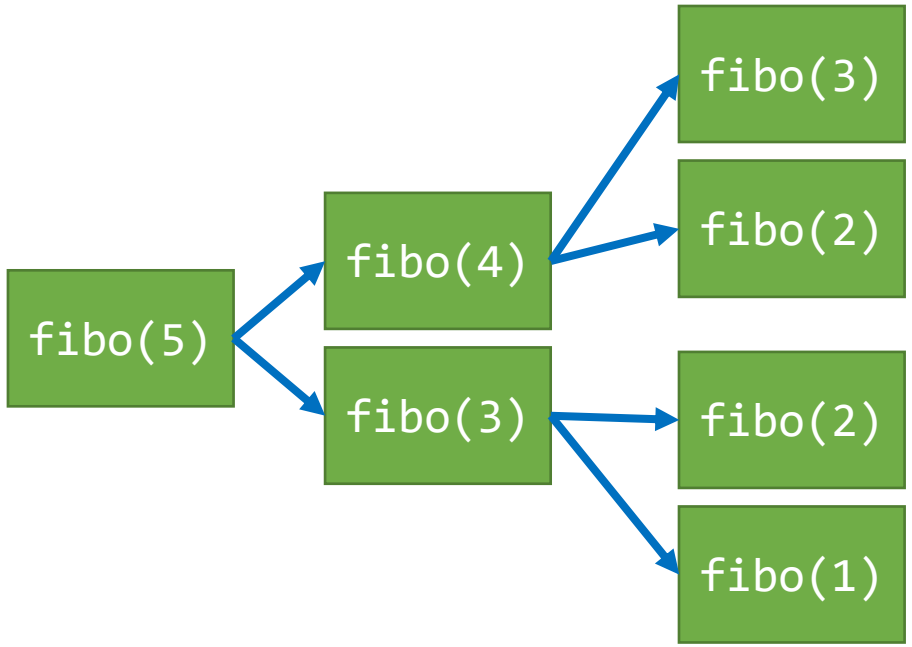
Die Fibonacci-Zahlenfolge

```
fibo(n)
   IF n = 1 OR n = 2 THEN      (*)
      ERGEBNIS ← 1;
   ELSE
      ERGEBNIS ← fibo(n-1) + fibo(n-2);
   END IF;

   RETURN ERGEBNIS;
```



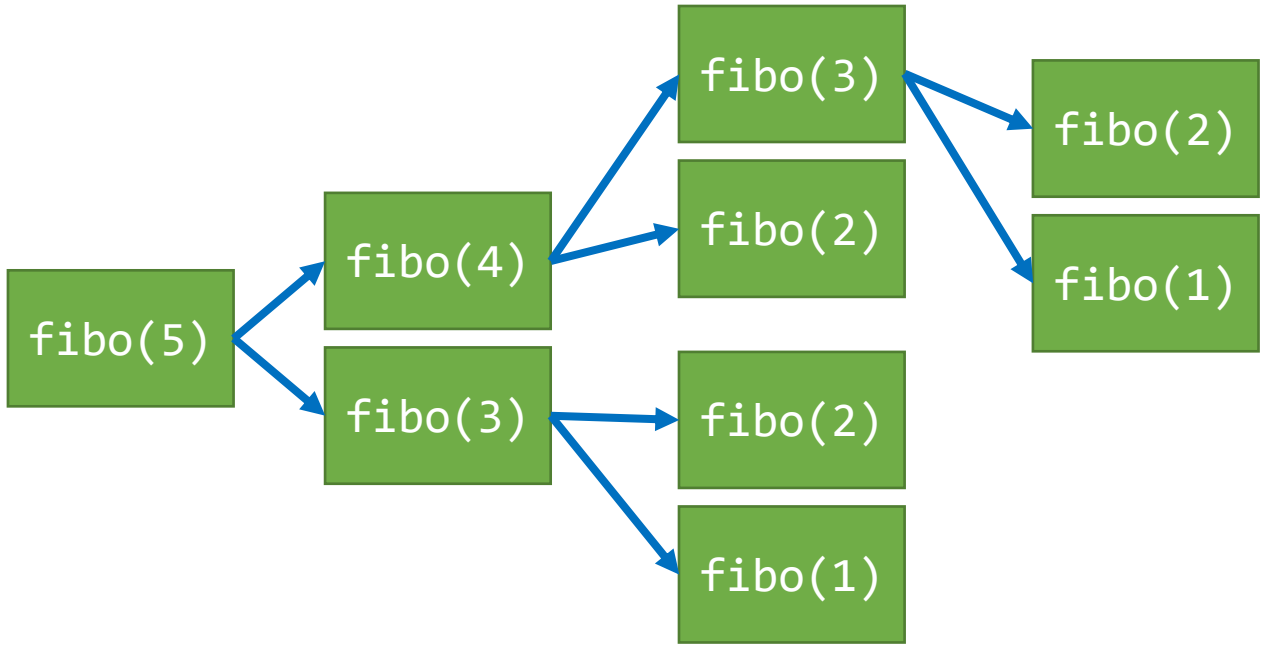
Die Fibonacci-Zahlenfolge

```
fibo(n)
   IF n = 1 OR n = 2 THEN      (*)
      ERGEBNIS ← 1;
   ELSE
      ERGEBNIS ← fibo(n-1) + fibo(n-2);
   END IF;

   RETURN ERGEBNIS;
```



Die Fibonacci-Zahlenfolge

```
fibo(n)
   IF n = 1 OR n = 2 THEN      (*)
      ERGEBNIS ← 1;
   ELSE
      ERGEBNIS ← fibo(n-1) + fibo(n-2);
   END IF;

   RETURN ERGEBNIS;
```



Die naive Textsuche

Die naive Textsuche hat eine theoretische Laufzeit von `O(n*m)` wenn `m` die Länge des Patterns und `n` die Länge des Textes ist.
Ihr habt schon in der Vorlesung gelernt, dass mithilfe der Vorverarbeitung des Patterns oder des Textes, sich eine deutlich schnellere Laufzeit erreichen lässt.

Welche Möglichkeiten gibt es, die naive Textsuche auch ohne Vorverarbeitung zu verbessern?

Welche Möglichkeiten gibt es, die naive Textsuche ohne Vorverarbeitung so zu verbessern, dass möglichst schnell *ein* Vorkommen des Patterns im Text gefunden wird?