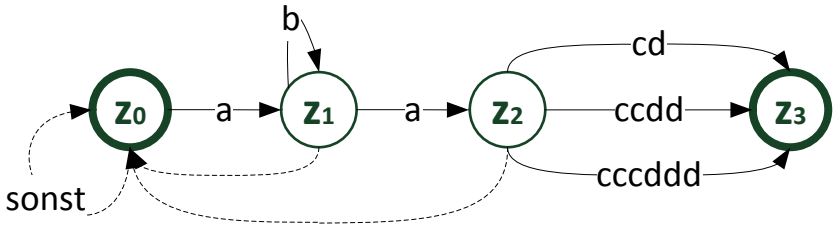
Aufgaben:

1. Zeichnen Sie einen *endlichen Automaten* auf, mit dessen Hilfe man folgende Sprache L bzw. folgende Wörter erzeugen kann:
   - Alle Wörter beginnen mit "aba", wobei darin kein, ein, zwei oder beliebig viele "b" vorkommen dürfen, also z.B. auch "aa", "abba" oder "abbbbbbbbbbbbba".
   - Alle Wörter enden mit "cd", "ccdd" oder "cccddd".

   Exemplarisch gilt somit: L = {aacd, aaccdd, aacccddd, abacd, abbbbbbbbbbbbbbacd, ...}

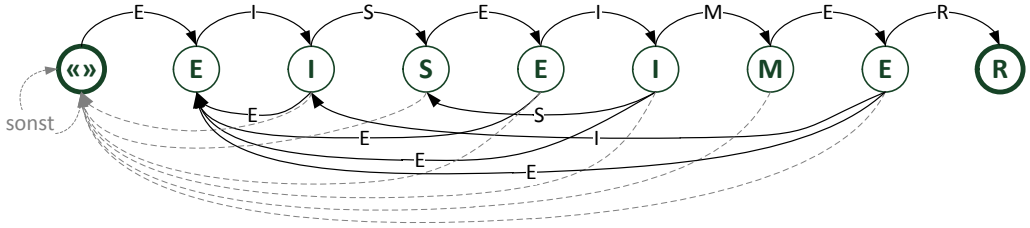


2. Mit der "einfachen Suche" durchsucht man einen Text *brute force* nach einem Muster. Wie viele Zeichenvergleiche sind im besten und wie viele im schlechtesten Fall erforderlich, wenn der zu durchsuchende Text n Zeichen lang ist und das Muster m Zeichen beinhaltet?
   Es gelte: n > m.

   Bester Fall: *m Vergleiche*

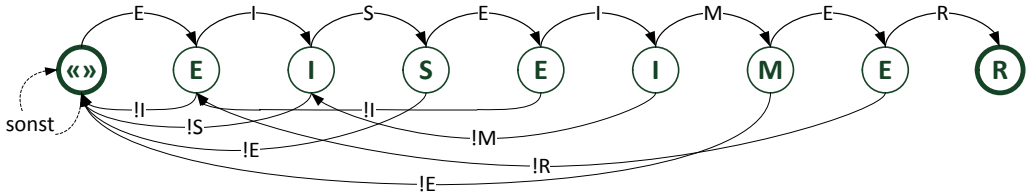
   Schlechtester Fall: *m ·(n – m + 1) Vergleiche*

3. Zeichnen Sie einen *endlichen Automaten* auf, um das Muster "EISEIMER" zu suchen.



4. Zeichnen Sie einen *Musterautomaten* auf, um das Muster "EISEIMER" zu suchen.

Teilwort	Rand	Länge des Randes
E	∅	*0*
EI	∅	*0*
EIS	∅	*0*
EISE	*E*	*1*
EISEI	*EI*	*2*
EISEIM	∅	*0*
EISEIME	*E*	*1*



5. Wir wollen mit dem KMP-Algorithmus nach "EISEIMER" suchen. Was liefert in diesem Zusammenhang die Methode `initNext(…)` als Ergebnis zurück?
-1, 0, 0, 0, 1, 2, 0, 1

6. Es gilt folgendes Alphabet: A = {a, b, c, d, e, f, g, h, i}. Wir wollen mit Quicksearch nach dem Muster "gaga" suchen. Wie welchen Werten muss dafür das Array `shift` initialisiert werden?
1, 5, 5, 5, 5, 5, 2, 5, 5

7. Welche Ordnung bezüglich Rechenzeit besitzen nachfolgende Suchalgorithmen im praktischen Gebrauch (also nicht theoretisch)? Der zu durchsuchende Text beinhalte n Zeichen und das Muster besitze m Zeichen. Es gelte: n > m. Ordnen Sie bestmöglich je mit einem Pfeil (->) zu:
O(n * m) | O(n + m) | O(n) | O(n / m)
Einfache Suche
Suche mit endlichem Automat
Suche mit KMP-Algorithmus
Quicksearch
   - *Einfache Suche* *O(n * m)*
   - *Suche mit endlichem Automat* *O(n)*
   - *Suche mit KMP-Algorithmus* *O(n + m)*
   - *Quicksearch* *O(n / m)*

8. In den Unterlagen finden Sie zu allen Suchalgorithmen der letzten Aufgabe den Source-Code. Überprüfen Ihre Zuordnungen der letzten Aufgabe, indem Sie den Source-Code begutachten.
Bemerkung: Wie Sie wissen, schlägt sich z.B. eine Ordnung $O(n^2)$ typisch in zwei verschachtelten Schleifen-Strukturen nieder, welche beide von n abhängig sind.
   - *Einfache Suche* *O(n * m)* *ALG6, Folie 32 (2 for-Schleifen, aussen „n“ innen „m“)*
   - *Suche mit endlichem Automat* *O(n)* *ALG 7, Folie 7&8 (1 do-while-Schleife, von„n“)*
   - *Suche mit KMP-Algorithmus* *O(n + m)* *ALG7, Folie 17 & 19 (17: do-while „m“, 19: do-while „n“)*
   - *Quicksearch* *O(n / m)* *ALG7, Folie 31 (do-while-Schleif, von „n“, plus „jump“)*

9. Beschreiben Sie kurz mit Ihren eigenen Worten, inwiefern sich Optimal-Mismatch und Quicksearch unterscheiden.
Beim Optimal-Mismatch versucht man immer zuerst solche Zeichenvergleiche zu machen, bei denen die Wahrscheinlichkeit für einen Mismatch gross ist. Im Falle eines Mismatches kann sofort vorwärts gesprungen werden, was den Algorithmus beschleunigt.

10. Markieren Sie im Source-Code zum Optimal-Mismatch Algorithmus jene Zeilen, wo man das Pattern zur Laufzeit reorganisiert.

```
if (j > 0) {          // swap is possible; j <-> (j-1)
    int temp1 = pPos[j];
    pPos[j] = pPos[j - 1]; pPos[j - 1] = temp1;
    char temp2 = pChar[j];
    pChar[j] = pChar[j - 1]; pChar[j - 1] = temp2;
}
```