

☐ Gr. 1, Dr. G. Kronberger

Name Andreas Roither Aufwand in h 6 h

☐ Gr. 2, Dr. H. Gruber



Gr. 3, Dr. D. Auer

Punkte _____ Kurzzeichen Tutor / Übungsleiter _____ / _____

1. *m*-Ketten-Problem

(4 + 6 Punkte)

- a) *Definition*: Eine Zeichenkette ist eine *m*-Kette, wenn sie höchstens *m* unterschiedliche Zeichen enthält.

Beispiele: Die drei Zeichenketten *a*, *ab* und *abcbaac* sind 3-Ketten, die Zeichenkette *abcd* ist aber keine 3-Kette mehr, sondern eine 4-Kette. Eine Zeichenkette *s* der Länge *n* ist natürlich eine *n*-Kette, von Interesse ist aber das kleinste *m* für welches die Bedingung aus der Definition oben für *s* noch gilt. Entwickeln Sie daher eine möglichst effiziente Funktion

```
FUNCTION MinM(s: STRING): INTEGER;
```

zur Ermittlung des kleinsten *m* für eine Zeichenkette *s*.

- b) Gegeben sei eine nichtleere Zeichenkette *s* und eine ganze Zahl *m* mit $1 \leq m \leq \text{Length}(s)$. Entwickeln Sie eine möglichst effiziente Funktion

```
FUNCTION MaxMStringLen(s: STRING, m: INTEGER): INTEGER;
```

welche die Länge der längsten *m*-Kette, die als Teilkette in *s* enthalten ist, liefert.

2. Wildcard Pattern Matching

(4 + (5 + 5) Punkte)

Viele Programme, z. B. Texteditoren und Kommandozeilen-Interpreter diverser Betriebssysteme (engl. *shells*), verwenden eine spezielle Variante der Zeichenkettensuche, die in der Musterkette Platzhalter (auch Jokerzeichen, engl. *wildcards*, genannt) zulässt. Denken Sie z. B. an den MS-DOS/Windows-Befehl *del *.** bzw. an das äquivalente UNIX-Kommando *rm **. Hier muss festgestellt werden, ob die Musterkette (**.** bzw. ***) zu einem Dateinamen im aktuellen Verzeichnis passt.

Jokerzeichen dürfen nur in Musterketten vorkommen: Dabei steht das Jokerzeichen *?* für *ein* beliebiges Zeichen in der Zeichenkette und das Jokerzeichen *** für eine *beliebige Anzahl* (null oder mehr) beliebiger Zeichen in der Zeichenkette. Jokerzeichen können auch gemischt und mehrfach in einer Musterkette vorkommen.

Nehmen Sie an, dass sowohl die Muster- als auch die Zeichenkette durch das spezielle Endezeichen *\$* abgeschlossen ist, welches innerhalb der Ketten nicht vorkommt. Folgende Tabelle zeigt einige einfache *Beispiele*:

Musterkette <i>p</i>	Zeichenkette <i>s</i>	<i>p</i> und <i>s</i> passen zusammen?
ABC\$	ABC\$	ja
AB <i>C</i> \$	AB\$	nein
ABC\$	ABCD\$	nein
A? <i>C</i> \$	A <i>X</i> C\$	ja
*\$	<i>beliebige auch leere Kette</i>	ja
A* <i>C</i> \$	AC\$	ja
A* <i>C</i> \$	A <i>XYZC</i> \$	ja

- a) Erweitern/ändern Sie den *BruteForce*-Algorithmus für die Zeichenkettensuche so, dass er obige Aufgabenstellung bewältigt, jedoch als Jokerzeichen nur ? (auch mehrfach) in der Musterkette vorkommen darf.
 - b) Die zusätzliche Behandlung des Jokerzeichens * ist mit den Standard-Algorithmen leider nicht mehr so einfach möglich. Allerdings lässt sich das Problem relativ einfach mittels Rekursion lösen:
 - 1. Definieren Sie zuerst ein rekursives Prädikat *Matching*(p, s), das *true* liefert, wenn p und s zusammenpassen, sonst *false*. Zerlegen Sie dabei sowohl p als auch s "geschickt" in zwei Teile: in das erste Zeichen und den Rest der Kette.
 - 2. Implementieren Sie das Prädikat *Matching* in Form einer rekursiven Funktion und testen Sie diese ausführlich.
-

Für besonders Interessierte: Implementieren Sie eine iterative Variante des Prädikats *Matching* aus 2.b.1, testen Sie diese ausführlich und stellen Sie Laufzeitvergleiche der rekursiven und iterativen Variante für lange Eingabeketten an. Als „**Belohnung**“ gibt es bis zu vier Zusatzpunkte.

Übung 2

Aufgabe 1

Lösungsidee

Bei MinM wird in eine Liste eingefügt falls ein Zeichen darin noch nicht enthalten ist. Am Schluss wird gezählt wie viele Zeichen in der Liste enthalten sind. Diese Anzahl gibt MinM zurück. Bei MaxMStringLen wird solange in eine Liste eingefügt bis m erreicht wird. Falls die Anzahl unterschiedlicher Zeichen in der Liste überschritten wurde, löscht RemoveFirst das erste Zeichen in der Liste. Falls die aktuelle Länge größer ist als die vorher gespeicherte Länge wird die aktuelle Länge gespeichert.

```

1 PROGRAM kette;
  (* Implementation with lists *)
3
  TYPE
5     nodePtr = ^listElement;
     listElement = RECORD
7         next: nodePtr;
         c: Char;
9     END; (* RECORD *)

11  (* Creates a new node*)
    FUNCTION NewNode(c : Char): nodePtr;
13  VAR node : nodePtr;
    BEGIN
15  New(node);
     node^.next := NIL;
17  node^.c := c;
     NewNode := node;
19  END;

21  (* Appends a Node to a List *)
    PROCEDURE Append(var list : nodePtr; element : nodePtr);
23  VAR tmp : nodePtr;
    BEGIN
25  if list = NIL THEN list := element ELSE
    BEGIN
27      tmp := list;
        WHILE tmp^.next <> NIL DO tmp := tmp^.next;
29
        tmp^.next := element;
31  END;
    END;

33
    (* recursive; disposes every node in a list *)
35  PROCEDURE ClearList(var list : nodePtr);
    BEGIN
37  IF list <> NIL THEN
    BEGIN
39      IF list^.next = NIL THEN dispose(list) ELSE ClearList(list^.next);
        END;
41  END;

43  (* Counts nodes in a list *)

```

```

45 FUNCTION CountNodes(n : nodePtr) : INTEGER;
46 VAR
47   count : INTEGER;
48 BEGIN
49   count := 0;
50   WHILE n <> NIL DO BEGIN
51     Inc(count);
52     n := n^.next;
53   END;
54   CountNodes := count;
55 END;
56
57 PROCEDURE RemoveFirst(var list : nodePtr);
58 VAR
59   temp : nodePtr;
60 BEGIN
61   IF list <> NIL THEN BEGIN
62     temp := list;
63     list := list^.next;
64     Dispose(temp);
65   END;
66 END;
67
68 (* Check if char exists in the list;
69    Returns TRUE OR FALSE *)
70 FUNCTION CharExists(list : nodePtr; c : Char): Boolean;
71 VAR
72   n : nodePtr;
73 BEGIN
74   n := list;
75
76   WHILE n <> NIL DO BEGIN
77     IF n^.c = c THEN BEGIN
78       CharExists := TRUE;
79       break;
80     END;
81     n := n^.next;
82   END;
83
84   IF n = NIL THEN CharExists := FALSE;
85 END;
86
87 (* Counts different chars in a list;
88    RETURNS 0 if empty*)
89 FUNCTION CountDistinct(list: nodePtr): Integer;
90 VAR
91   temp, temp2 : nodePtr;
92 BEGIN
93   IF list <> NIL THEN
94     BEGIN
95       temp := list;
96       temp2 := NIL;
97
98       WHILE temp <> NIL DO BEGIN
99         IF NOT CharExists(temp2, temp^.c) THEN Append(temp2, NewNode(temp^.c));
100         temp := temp^.next;
101       END;

```

```

    CountDistinct := CountNodes(temp2);
103   ClearList(temp2);
END
105 ELSE
    CountDistinct := 0;
107 END;

(* Prints out list *)
PROCEDURE PrintList(list : nodePtr);
111 VAR
    n : nodePtr;
113 BEGIN
    n := list;
115
    WHILE n <> NIL DO BEGIN
117         Write(n^.c);
            n := n^.next;
119     END;
    WriteLn;
121 END;

(* Insert to string from a list RETURNS STRING*)
FUNCTION InsertinString(list : nodePtr): STRING;
125 VAR
    n : nodePtr;
127 s : STRING;
    BEGIN
129     n := list;
        s := '';
131
    WHILE n <> NIL DO BEGIN
133         s := Concat(s,n^.c);
            n := n^.next;
135     END;
    InsertinString := s;
137 END;

(* Implementation with Single Linked List *)
FUNCTION MinM(s: STRING) : Integer;
141 VAR
    i : Integer;
143 list : nodePtr;
    BEGIN
145     list := NIL;

147     FOR i := 1 TO Length(s) DO BEGIN
        IF NOT CharExists(list ,s[i]) THEN Append(list , NewNode(s[i]));
149     END;

151     MinM := CountNodes(list);
    END;
153

(* Implementation with Single Linked List *)
FUNCTION MaxMStringLen(var longestS: STRING; s: STRING; m: Integer): Integer;
155 VAR
    i, count, tempCount, maxLength : Integer;
157 list : nodePtr;
159 BEGIN

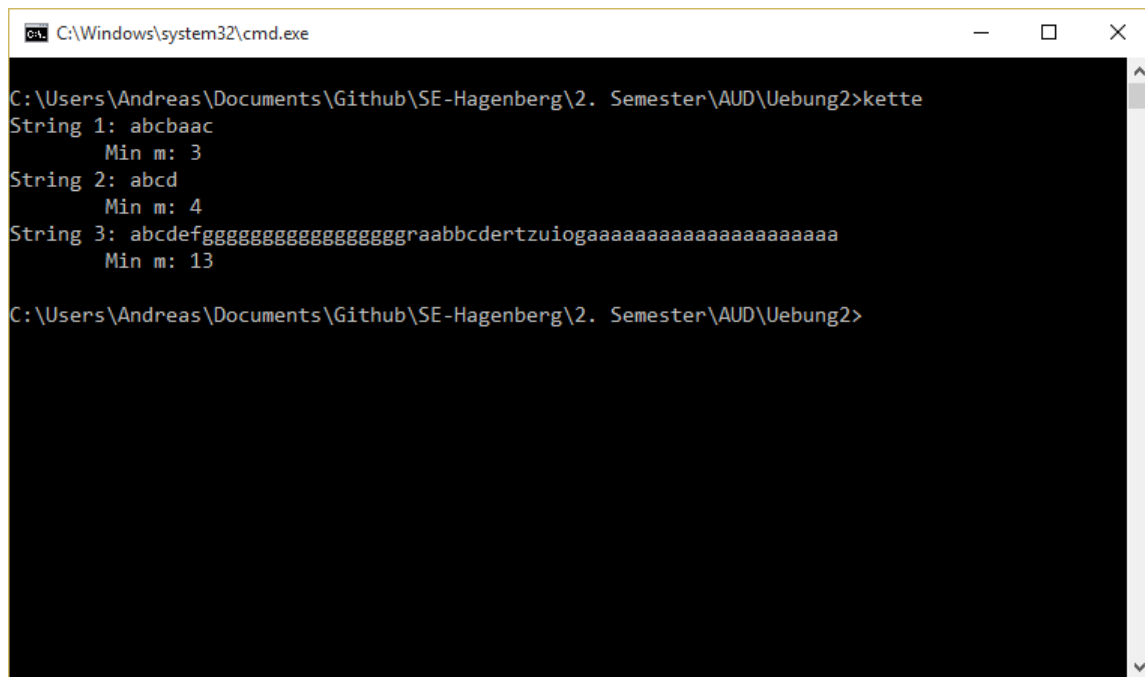
```

```

161 list := NIL;
count := 0;
maxLength := 0;
163
165 FOR i := 1 TO Length(s) DO BEGIN
167     Append(list , NewNode(s[i]));
tempCount := CountDistinct(list);
169
    IF tempCount > m THEN
171     BEGIN
        RemoveFirst(list);
173     END
    ELSE
        count := count + 1;
175
        IF count > maxLength THEN BEGIN
177             maxLength := count;
            longestS := InsertinString(list);
179
        END;
181 END;
MaxMStringLen := maxLength;
183 END;
185
VAR
s1, s2, s3, longest : String;
187 BEGIN
s1 := 'abcbaac';
189 s2 := 'abcd';
s3 := 'abcdefggggggggggggggggggraabbdertzuioaaaaaaaaaaaaaaaaaaaaa';
191 s3 := 'abcdefggggggggggggggggggraabbdertzuioaaaaaaaaaaaaaaaaaaaaa';
longest := '';
193
WriteLn('String 1: ', s1, #13#10#9, 'Min m: ', MinM(s1));
195 WriteLn('String 2: ', s2, #13#10#9, 'Min m: ', MinM(s2));
WriteLn('String 3: ', s3, #13#10#9, 'Min m: ', MinM(s3));
197
WriteLn('_____');
199 WriteLn('String 1 mit m 2: ', s1, #13#10#9, 'MaxM: ', MaxMStringLen(longest, s1
, 2),
#13#10#9, 'Longest substring: ', longest, #13#10);
201 WriteLn('String 2 mit m 4: ', s2, #13#10#9, 'MaxM: ', MaxMStringLen(longest, s2
, 4),
#13#10#9, 'Longest substring: ', longest, #13#10);
203 WriteLn('String 3 mit m 5: ', s3, #13#10#9, 'MaxM: ', MaxMStringLen(longest, s3
, 5),
#13#10#9, 'Longest substring: ', longest, #13#10);
205 WriteLn('String 3 mit m 7: ', s3, #13#10#9, 'MaxM: ', MaxMStringLen(longest, s3
, 7),
#13#10#9, 'Longest substring: ', longest, #13#10);
207
END.

```

Kette.pas



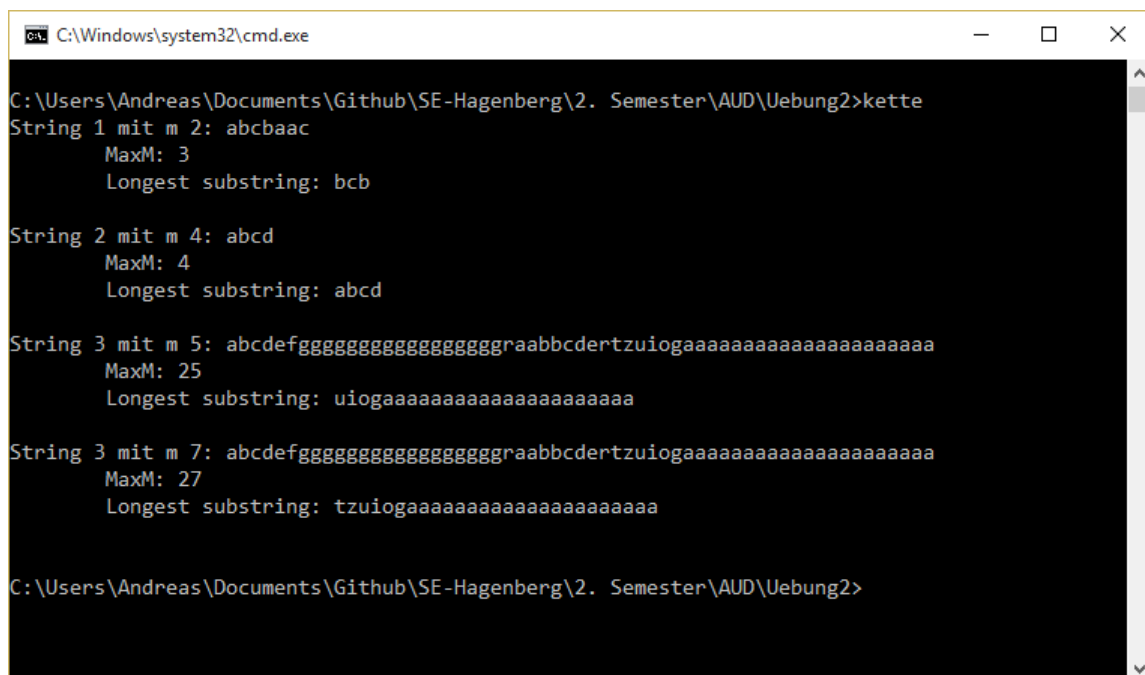
```
C:\Windows\system32\cmd.exe

C:\Users\Andreas\Documents\Github\SE-Hagenberg\2. Semester\AUD\Uebung2>kette
String 1: abcbaac
      Min m: 3
String 2: abcd
      Min m: 4
String 3: abcdefgggggggggggggggggggggraabbcdertzuiogaaaaaaaaaaaaaaaaaaaaa
      Min m: 13

C:\Users\Andreas\Documents\Github\SE-Hagenberg\2. Semester\AUD\Uebung2>
```

Abbildung 1: Ausgabe 1a

Ausgegeben wird die Anzahl der unterschiedlichen Zeichen der oben angegebenen Zeichenketten.



```
C:\Windows\system32\cmd.exe

C:\Users\Andreas\Documents\Github\SE-Hagenberg\2. Semester\AUD\Uebung2>kette
String 1 mit m 2: abcbaac
      MaxM: 3
      Longest substring: bcb

String 2 mit m 4: abcd
      MaxM: 4
      Longest substring: abcd

String 3 mit m 5: abcdefgggggggggggggggggggggraabbcdertzuiogaaaaaaaaaaaaaaaaaaaaa
      MaxM: 25
      Longest substring: uiogaaaaaaaaaaaaaaaaaaaaa

String 3 mit m 7: abcdefgggggggggggggggggggggraabbcdertzuiogaaaaaaaaaaaaaaaaaaaaa
      MaxM: 27
      Longest substring: tzuiogaaaaaaaaaaaaaaaaaaaaa

C:\Users\Andreas\Documents\Github\SE-Hagenberg\2. Semester\AUD\Uebung2>
```

Abbildung 2: Ausgabe 1b

Hier wird der längste Substring mit maximalen m eines Strings ausgegeben.

Aufgabe 2

Lösungsidee

Das Matching funktioniert ähnlich wie die BruteForce Methode. Es wird von links nach rechts durch den string gegangen. Dabei wird bei jedem Aufruf ein Teil des string nicht mehr mit übergeben, für die Zeichen “*” und “?” werden dabei spezielle Aktionen durchgeführt. Falls die Zeichenfolge nicht mehr übereinstimmt wird False zurückgegeben, andernfalls wird True zurückgegeben.

```

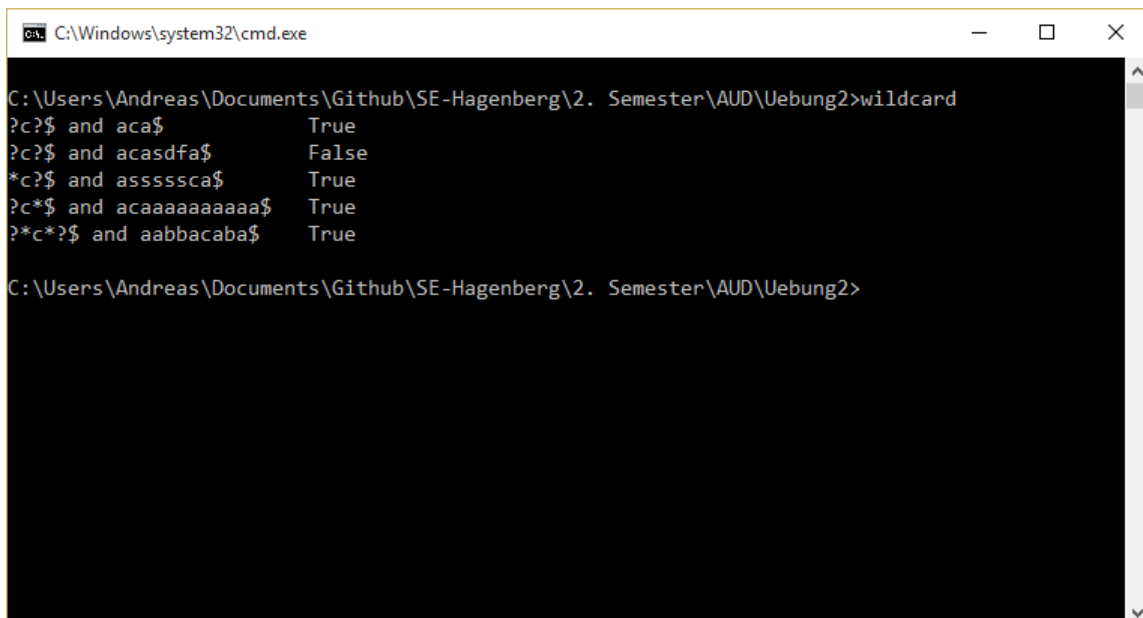
PROGRAM wildcard;
2
(* Matching Going from the left to right
3   recursive *)
4 FUNCTION Matching(p, s : STRING): Boolean;
5
6 VAR
7     i, j : Integer;
8 BEGIN
9     i := 1;
10    j := 1;
11
12    (* *)
13    WHILE (p[j] <> '*' ) AND (j <= length(p)) AND ((s[i] = p[j]) OR
14      (p[j] = '?')) DO BEGIN
15        i := i + 1;
16        j := j + 1;
17      END;
18
19      IF (p[j] <> '*' ) AND (i <= length(s)) THEN BEGIN
20        Matching := Matching(p, Copy(s, 2, length(s)))
21      END
22      ELSE IF (j <= length(p)) AND (i <= length(s)) THEN BEGIN
23        Matching := Matching(Copy(p, j + 1, Length(p)), Copy(s, 2, length(s)));
24      END
25      ELSE IF ((j >= length(p)) AND (i >= length(s))) OR
26        ((j = length(p)) AND (p[j] = '*')) THEN BEGIN
27        Matching := True;
28      END
29      ELSE Begin
30        Matching := False;
31      END;
32    END;
33    VAR
34      s, p : STRING;
35    BEGIN
36      s := '?c?$';
37      p := 'aca$';
38
39      IF Matching(s, p) THEN WriteLn(s, ' and ', p, #9#9, ' True')
40      ELSE WriteLn(s, ' and ', p, #9, ' False');
41
42      s := '?c?$';
43      p := 'acasdfa$';
44      IF Matching(s, p) THEN WriteLn(s, ' and ', p, #9, ' True')
45      ELSE WriteLn(s, ' and ', p, #9, ' False');
46
47      s := '*c?$';

```



```
48  p := 'assssca$';  
    IF Matching(s, p) THEN WriteLn(s, ' and ', p, #9, ' True')  
50  ELSE WriteLn(s, ' and ', p, #9, ' False');  
  
52  s := '?c*$';  
    p := 'aaaaaaaaaaaa$';  
54  IF Matching(s, p) THEN WriteLn(s, ' and ', p, #9, ' True')  
    ELSE WriteLn(s, ' and ', p, #9, ' False');  
56  
    s := '?*c*?$';  
58  p := 'aabbacaba$';  
    IF Matching(s, p) THEN WriteLn(s, ' and ', p, #9, ' True')  
60  ELSE WriteLn(s, ' and ', p, #9, ' False');  
END.
```

wildcard.pas



```
C:\Windows\system32\cmd.exe  
  
C:\Users\Andreas\Documents\Github\SE-Hagenberg\2. Semester\AUD\Uebung2>wildcard  
?c*$ and aca$      True  
?c*$ and acasdfa$  False  
*c*$ and assssca$  True  
?c*$ and aaaaaaaaa$ True  
?*c*?$ and aabbacaba$ True  
  
C:\Users\Andreas\Documents\Github\SE-Hagenberg\2. Semester\AUD\Uebung2>
```

Abbildung 3: Ausgabe Matching

Ausgabe für die jeweiligen Zeichenketten.