HvA

Sorting and Searching

Practicum 3

Martin Bouma (500695988) en Angela van der Veer (500692573)
7-1-2015
IS203

Inleiding

Dit document beschrijft de uitwerking van opdracht 3 van Sorting and Searching. In opgave a wordt de substring methode van twee programmeertalen besproken. In opgave b worden de algoritmes Knuth-Morris-Pratt en Boyer-Moore vergeleken. In opdracht c wordt met reguliere expressies getest of een telefoonboek voldoet aan bepaalde regels.

a. Implementatie van substring in programmeertalen

In dit hoofdstuk bekijken we substring methoden van de programmeertalen Java en C. De broncode van beide methoden wordt weergegeven waarna we verklaren om welke variant van het algoritme het gaat (brute force, Knuth-Morris-Pratt, Boyer-Moore, Rabin-Karp…). In Java heet de substringmethode “indexOf()”. In C heet de substringmethode “strstr”.

Substring in Java

Dit is de broncode van de door Java aangeleverde substring methode “indexOf()”

```
/**
     * Returns the index within this string of the first occurrence of the
     * specified substring. The integer returned is the smallest value
     * <i>k</i> such that:
     * <blockquote><pre>
     * this.startsWith(str, <i>k</i>)
     * </pre></blockquote>
     * is <code>true</code>.
     *
     * @param   str   any string.
     * @return  if the string argument occurs as a substring within this
     *          object, then the index of the first character of the first
     *          such substring is returned; if it does not occur as a
     *          substring, <code>-1</code> is returned.
     */
    public int indexOf(String str) {
    return indexOf(str, 0);
    }

    /**
     * Returns the index within this string of the first occurrence of the
     * specified substring, starting at the specified index.  The integer
     * returned is the smallest value <tt>k</tt> for which:
     * <blockquote><pre>
     *     k &gt;= Math.min(fromIndex, this.length()) && this.startsWith(str,
k)
     * </pre></blockquote>
     * If no such value of <i>k</i> exists, then -1 is returned.
     *
     * @param   str         the substring for which to search.
     * @param   fromIndex   the index from which to start the search.
     * @return  the index within this string of the first occurrence of the
     *          specified substring, starting at the specified index.
     */
    public int indexOf(String str, int fromIndex) {
        return indexOf(value, offset, count,
                       str.value, str.offset, str.count, fromIndex);
    }

    /**
     * Code shared by String and StringBuffer to do searches. The
     * source is the character array being searched, and the target
```

```
     * is the string being searched for.
     *
     * @param   source       the characters being searched.
     * @param   sourceOffset offset of the source string.
     * @param   sourceCount  count of the source string.
     * @param   target       the characters being searched for.
     * @param   targetOffset offset of the target string.
     * @param   targetCount  count of the target string.
     * @param   fromIndex    the index to begin searching from.
     */
    static int indexOf(char[] source, int sourceOffset, int sourceCount,
                       char[] target, int targetOffset, int targetCount,
                       int fromIndex) {
    if (fromIndex >= sourceCount) {
            return (targetCount == 0 ? sourceCount : -1);
    }
        if (fromIndex < 0) {
            fromIndex = 0;
        }
    if (targetCount == 0) {
        return fromIndex;
    }

        char first  = target[targetOffset];
        int max = sourceOffset + (sourceCount - targetCount);

        for (int i = sourceOffset + fromIndex; i <= max; i++) {
            /* Look for first character. */
            if (source[i] != first) {
                while (++i <= max && source[i] != first);
            }

            /* Found first character, now look at the rest of v2 */
            if (i <= max) {
                int j = i + 1;
                int end = j + targetCount - 1;
                for (int k = targetOffset + 1; j < end && source[j] ==
                        target[k]; j++, k++);

                if (j == end) {
                    /* Found whole string. */
                    return i - sourceOffset;
                }
            }
        }
        return -1;
    }
```

Bron: Netbeans, Java source code

Welke variant?

De methode indexOf() werkt volgens het “Brute force” principe. Deze variant zoekt in de string stapsgewijs van links naar rechts naar de eerste letter van het patroon. Als deze match gevonden is worden de daaropvolgende letters uit het patroon vergeleken met de daaropvolgende letters uit de string. Als alle letters matchen is het patroon (de substring) gevonden in de string. Als er een mismatch optreedt, dan gaat de code opnieuw zoeken naar de eerste letter van het patroon vanaf de plek waar de mismatch plaats heeft gevonden.

Substring in C

Dit is de broncode van de door C aangeleverde substring methode “strstr”

```
/*
 * Copyright (c) 2001 Apple Computer, Inc. All rights reserved.
 *
 * @APPLE_LICENSE_OSREFERENCE_HEADER_START@
 *
 * This file contains Original Code and/or Modifications of Original Code
 * as defined in and that are subject to the Apple Public Source License
 * Version 2.0 (the 'License'). You may not use this file except in
 * compliance with the License.  The rights granted to you under the
 * License may not be used to create, or enable the creation or
 * redistribution of, unlawful or unlicensed copies of an Apple operating
 * system, or to circumvent, violate, or enable the circumvention or
 * violation of, any terms of an Apple operating system software license
 * agreement.
 *
 * Please obtain a copy of the License at
 * http://www.opensource.apple.com/apsl/ and read it before using this
 * file.
 *
 * The Original Code and all software distributed under the License are
 * distributed on an 'AS IS' basis, WITHOUT WARRANTY OF ANY KIND, EITHER
 * EXPRESS OR IMPLIED, AND APPLE HEREBY DISCLAIMS ALL SUCH WARRANTIES,
 * INCLUDING WITHOUT LIMITATION, ANY WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE, QUIET ENJOYMENT OR NON-INFRINGEMENT.
 * Please see the License for the specific language governing rights and
 * limitations under the License.
 *
 * @APPLE_LICENSE_OSREFERENCE_HEADER_END@
 */
/*
 * Copyright (c) 1988, 1993
 *      The Regents of the University of California.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *      This product includes software developed by the University of
 *      California, Berkeley and its contributors.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
```

```
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */
/*
 * History:
 *  2002-01-24 gvdl    Initial implementation of strstr
 */

#include <string.h>
#include <libsa/stdlib.h>

__private_extern__ char *
strstr(const char *in, const char *str)
{
    char c;
    size_t len;

    c = *str++;
    if (!c)
        return (char *) in;    // Trivial empty string case

    len = strlen(str);
    do {
        char sc;

        do {
            sc = *in++;
            if (!sc)
                return (char *) 0;
        } while (sc != c);
    } while (strncmp(in, str, len) != 0);

    return (char *) (in - 1);
}
```

Bron: http://www.opensource.apple.com/source/xnu/xnu-792.13.8/libsa/strstr.c

Verklarende woordenlijst:
size_t is an unsigned integer type of at least 16 bit
* is een pointer naar een plek in het geheugen waar een variabele staat opgeslagen.

Welke variant?

Deze substring zoekmethode werkt ook volgens het brute force principe. Er wordt in deze methode wel gebruik gemaakt van een andere substring methode die ook volgens dat zelfde principe werkt. De code vergelijkt stapsgewijs de eerste letter uit de substring met elk karakter uit de string, van links naar rechts. Als de karakters overeenkomen wordt de tweede substring methode aangeroepen, namelijk “strncmp”
Deze kijkt of de rest van de karakters in de substring overeenkomen met de karakters in de string ná het gevonden eerste overeenkomende karakter. Dit is dus ook volgens het brute force principe. Komt er een karakter niet overeen dan gaat de methode “strstr” verder met zoeken één plek na de zojuist gevonden overeenkomst (die dus geen match opleverde tussen de substring en de string).

b. Vergelijken van algoritmes

Voor deze opdracht moesten we door een tekst heen zoeken. Hiervoor hebben we de tekst in een file gezet en daar doorheen gezocht. Hiervoor hebben we deze code gebruikt.

```
public static String readFile(String fileName) {
    int aantalCharsInFile;
    char[] arrayCharsUitFile = new char[145580];
    final StringBuffer buffer = new StringBuffer();
    final FileReader reader;
    try {
       reader = new FileReader(fileName);
       while ((aantalCharsInFile = reader.read(arrayCharsUitFile)) > 0) {
          buffer.append(arrayCharsUitFile, 0, aantalCharsInFile);
       }
    } catch (Exception e) {
       String s = e.getMessage();
    }
    return buffer.toString();
 }
```

Bron:http://www.java2s.com/Tutorials/Java/IO/Text_File/Load_a_text_file_contents_as_a_String_in_Java.htm

Voor de algoritme Knuth-Morris-Pratt hebben we de code van het boek gebruikt. Hieronder staat wat we hebben gebruikt. In de comments staat wat we hebben toegevoegd en verandert.

```
private String pat;
private int[][] dfa;
public int vergelijkingen;                    // toegevoegd

public KMP(String pat) {
   this.pat = pat;
   int M = pat.length();
   int R = 145580;                            // aantal karakters in de tekst
   dfa = new int[R][M];
   dfa[pat.charAt(0)][0] = 1;
   for (int X = 0, j = 1; j < M; j++) {
      for (int c = 0; c < R; c++) {
         dfa[c][j] = dfa[c][X];
      }
      dfa[pat.charAt(j)][j] = j + 1;
      X = dfa[pat.charAt(j)][X];
   }
}

public int search(int pos, String txt) { // pos toegevoegd
   int i, j, N = txt.length(), M = pat.length();
   for (i = pos, j = 0; i < N && j < M; i++) // pos toegevoegd
   {
      vergelijkingen++;                        // toegevoegd
      j = dfa[txt.charAt(i)][j];
   }
   if (j == M) {
      return i - M;
   } else {
      return N;
   }
}
```

Voor de algoritme Boyer-Moore hebben we ook de code uit het boek gebruikt. Ook hier hebben we wat aangepast.

```
private int[] right;
private String pat;
public int vergelijkingen;                  // toegevoegd

public BoyerMoore(String pat) {
   this.pat = pat;
   int M = pat.length();
   int R = 145580;                          // aantal karakters in tekst
   right = new int[R];
   for (int c = 0; c < R; c++) {
      right[c] = -1;
   }
   for (int j = 0; j < M; j++) {
      right[pat.charAt(j)] = j;
   }
}

public int search(int pos, String txt) { // pos toegevoegd
   int N = txt.length();
   int M = pat.length();
   int skip;
   for (int i = pos; i <= N - M; i += skip) { // pos toegevoegd
      vergelijkingen++;                     // toegevoegd
      skip = 0;
      for (int j = M - 1; j >= 0; j--) {
         if (pat.charAt(j) != txt.charAt(i + j)) {
            skip = j - right[txt.charAt(i + j)];
            if (skip < 1) {
               skip = 1;
            }
            break;
         }
      }
      if (skip == 0) {
         return i;
      }
   }
   return N;
}
```

Voor het gebruik van de algoritmes hebben we deze methoden geschreven.

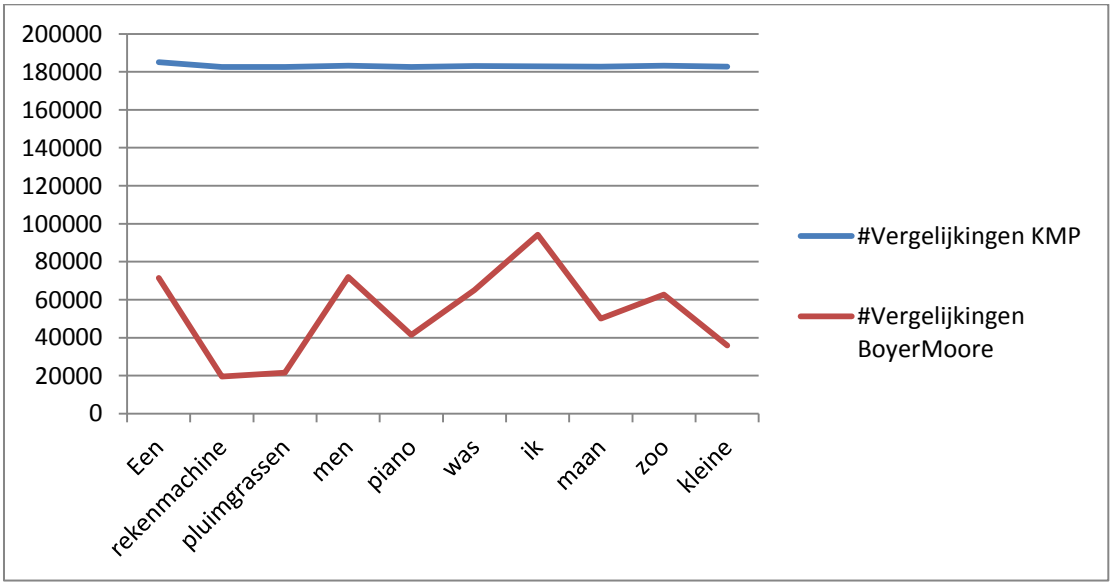
```
public static void searchKMP(String text, String searchWord) {
    int aantal = 0;
    //extra: het zoekwoord wordt volledig naar kleine letters gezet
    KMP kmp = new KMP(searchWord.toLowerCase());

    int positie = 0;
    //145580 is het aantal characters in de tektst
    for (int i = 0; i < 145580; i++) {
      //extra: de hele tektst wordt naar kleine letters gezet
      positie = kmp.search(positie + 1, text.toLowerCase());
      if (positie == text.length()) {
        System.out.println("Aantal: " + aantal);
        System.out.println("Vergelijkingen: " + kmp.vergelijkingen);
        return;
      } else {
        aantal++;
      }
    }
  }

public static void searchBoyerMoore(String text, String searchWord) {
    int aantal = 0;
    //extra: de hele tektst wordt naar kleine letters gezet
    BoyerMoore boyerMoore = new BoyerMoore(searchWord.toLowerCase());

    int positie = 0;
    //145580 is het aantal characters in de tektst
    for (int i = 0; i < 145580; i++) {
      //extra: de hele tektst wordt naar kleine letters gezet
      positie = boyerMoore.search(positie + 1, text.toLowerCase());
      if (positie == text.length()) {
        System.out.println("Aantal: " + aantal);
        System.out.println("Vergelijkingen: " + boyerMoore.vergelijkingen);
        return;
      } else {
        aantal++;
      }
    }
  }
```

We houden bij hoeveel keren het woord voorkomt door aantal en de positie van waar woord is gevonden met positie.
We zorgen er eerst voor dat het woord dat gezocht word allemaal kleine letters zijn. Vervolgens lopen we door de hele tekst om te zoeken. Bij iedere character gaan we een character verder en gebruiken we de methode van KMP om te vergelijken en te zoeken. Zodra het einde van de tekst is bereikt, wordt het aantal overeenkomsten en vergelijkingen uitgeprint.

Deze tabel volgt uit het uitvoeren van het programma:

Woord	Een	rekenmachine	pluimgrassen	men	piano	was	ik	maan	zoo	kleine
#Vergelijkingen KMP	185078	182582	182593	183264	182586	183102	182965	182732	183236	182677
#Vergelijkingen BoyerMoore	71437	19515	21614	72006	41507	65104	94196	50054	62714	35996
#Matches	1248	0	1	341	1	260	383	50	327	19

Om het vergelijken van de twee methoden makkelijker te maken hebben we ook een grafiek gemaakt:



Nu kijken we of onze resultaten in lijn zijn met de tabel "Cost summary for substring search implementation" uit het boek.

#Vergelijkingen KMP	185078	182582	182593	183264	182586	183102	182965	182732	183236	182677
#Karakters	3	12	12	3	5	3	2	4	3	6
Guarantee (3N)	436740	436740	436740	436740	436740	436740	436740	436740	436740	436740
Typical (1.1N)	160138	160138	160138	160138	160138	160138	160138	160138	160138	160138
#Vergelijkingen BoyerMoore	71437	19515	21614	72006	41507	65104	94196	50054	62714	35996
#Karakters	3	12	12	3	5	3	2	4	3	6
Guarantee (M*N)	214311	1746960	1746960	214311	727900	214311	291160	582320	214311	873480
Typical (N/M)	48527	12132	12132	48527	29116	48527	72790	36395	48527	24263

N = tekstlengte
M = lengte substring

Conclusie
Het aantal vergelijkingen die onze test heeft opgeleverd vallen ruimschoots binnen de "guarantee". Ze zijn wat hoger dan de "typical", maar wij vinden ze er dicht genoeg bij liggen.

c. Reguliere expressies

Om te bekijken of een string geschikt is voor een telefoonnummer hebben we deze code geschreven.

```
public static boolean checkNummer(String telNr) {
     //netnummer van 3 cijfers
     //Hij checkt of het nummer begint met een 0 of +31 of 0031
     //Dan of er 2 cijfers zijn, deze kunnen alles zijn (0 tm 9)
     //Dan mogelijk een streepje
     //Dan een cijfer dat hoger is dan 0
     //Dan volgen nog 6 cijfers
      Pattern pattern = Pattern.compile("((0|\\+31|0031)\\d{2}-?[^0]\\d{6})");
      boolean netThreeNumber = pattern.matcher(telNr).matches();

      //netnummer van 4 cijfers
     //Hij checkt of het nummer begint met een 0 of +31 of 0031
     // Dan twee cijfers die hoger zijn dan 0
     //Dan 1 cijfer
     //Dan mogelijk een streepje
     //Dan een cijfer dat hoger is dan 0
     //Dan nog 5 cijfers
      pattern = Pattern.compile("(0|\\+31|0031)[^0][^0]\\d{1}-?[^0]\\d{5}");
      boolean netFourNumber = pattern.matcher(telNr).matches();

     //mobiele nummer
     //Hij checkt of het nummer begint met een 0 of +31 of 0031
     //Dan verplicht een 6
     //Dan mogelijk een streepje
     //Dan 8 cijfers
      pattern = Pattern.compile("(0|\\+31|0031)6-?[^0]\\d{7}");
      boolean mobileNumber = pattern.matcher(telNr).matches();

      //uitzondering nummers
     //Óf alleen 112
     //Óf verplicht 0900 met mogelijk een streepje
     //Dan 4 cijfers
     //Óf verplicht 0800 met mogelijk streepje
     //Dan 4 cijfers
      pattern = Pattern.compile("112|0900-?[^0]\\d{3}|0800-?[^0]\\d{3}");
      boolean exceptionNumber = pattern.matcher(telNr).matches();

     // return true als er een van de 4 mogelijke soorten nummers klopt, anders false
     if (netThreeNumber || netFourNumber || mobileNumber || exceptionNumber) {
        return true;
     }
     return false;
  }
```

Hierna hebben we een aantal telefoonnummers bekeken of die kloppen of niet. Bij de comments bij de foute nummers staat aangegeven precies welke fout we gemaakt hebben.

```
//klopt:
System.out.println("003172-5117049  " + ReguliereExpressies.checkNummer("003172-5117049"));
System.out.println("0720-117049  " + ReguliereExpressies.checkNummer("0720-117049"));
System.out.println("+3172-5117049  " + ReguliereExpressies.checkNummer("+3172-5117049"));
System.out.println("06-52154073  " + ReguliereExpressies.checkNummer("06-52154073"));
System.out.println("+316-52154073  " + ReguliereExpressies.checkNummer("+316-52154073"));
System.out.println("112  " + ReguliereExpressies.checkNummer("112"));
System.out.println("0900-1234  " + ReguliereExpressies.checkNummer("0900-1234"));
System.out.println("09001234  " + ReguliereExpressies.checkNummer("09001234"));

//fout:
//dit nummer begtin met een 1 ipv een 0
System.out.println("1720-117049  " + ReguliereExpressies.checkNummer("1720-117049"));
//dit nummer heeft maar drie cijfers na de 0800
System.out.println("0800534  " + ReguliereExpressies.checkNummer("0800534"));
//dit nummer heeft twee streepjes
System.out.println("+316--87654321  " + ReguliereExpressies.checkNummer("+316--87654321"));
//het americaanse alarmnummer ipv nederlandse
System.out.println("911  " + ReguliereExpressies.checkNummer("911"));
//ipv 0031 "perongeluk" 0032
System.out.println("003256-9256705  " + ReguliereExpressies.checkNummer("003256-9256705"));
//nummer met netnummer van 3 cijfers; abonneenummer dat begint met 0
System.out.println("+3172-0123456  " + ReguliereExpressies.checkNummer("+3172-0123456"));
//nummer met netnummer van 4 cijfers; abonneenummer dat begint met 0
System.out.println("+31569-012345  " + ReguliereExpressies.checkNummer("+31569-012345"));
```

Na runnen is de output:

```
Output:
run:
003172-5117049  true
0720-117049  true
+3172-5117049  true
06-52154073  true
+316-52154073  true
112  true
0900-1234  true
09001234  true
1720-117049  false
0800534  false
+316--87654321  false
911  false
003256-9256705  false
+3172-0123456  false
+31569-012345  false
BUILD SUCCESSFUL (total time: 0 seconds)
```