Efficient Algorithms

Task 04 - Suffix Array - LCP - Search

Bericht

Es wurden alle vier Kapitel des Buches „Hound of Baskerville“ eingelesen. Umbrüche wurden als Leerzeichen und größere Abstände als ein Leerzeichen umgeschrieben. Jedes Zeichen wird lower case geschrieben. Danach wurde aus diesem String ein Array erstellt das jeweils suffix und index enthält. In die Suchfunktion werden Suffix Array, Text und der Suchbegriff („Sherlock“, „Watson“, „Hound“) übergeben. Nun wird die erste Stelle ermittelt wo das Suchwort erscheint. Mit Hilfe des Textes und dem Suffix Array wird ein LCP Array erstellt. Nun wird über alle Suffixe iteriert ab der Stelle wo das Suchwort das erste mal auftaucht. Wir zählen einen Counter hoch für jede Iteration. Sobald das die Stelle im LCP Array geringer ist als das Suchwort lang ist retournieren wir die Anzahl des Counters.

Search Term	Execution Time (s)	Comparison	Appearance
Sherlock	0.145788908005	66	12
Watson	0.13347697258	81	32
Hound	0.142812013626	64	14

Code Suffix Array

```
# create a suffix array with suffix returns array of tuples
def suffix(text):
    suffix_array = []
    for i in range(0, len(text)):
        suffix_array.append((text[i:len(text)], i))

    suffix_array = sorted(suffix_array)

    return suffix_array


# creates the suffix array out of the array with suffix array tuples
def create_idx_list(suffix_array_tuples):
    suffix_array_idx = []
    for key, value in suffix_array_tuples:
        suffix_array_idx.append(value)

    return suffix_array_idx
```

Code LCP

```
# calculates the longest common prefix of two terms
def lcp(term1, term2):
    for i in xrange(min(len(term1), len(term2))):
        if term1[i] != term2[i]:
            return i

    return min(len(term1), len(term2))


# creates an lcp array of a given string
def create_lcp_array(str, suffix_array):
    size = len(str)
    reverse_suffix_array = range(0,size)

    for x in range(0,size):
        reverse_suffix_array[suffix_array[x]] = x

    lcp = size * [None]
    h = 0
    for i in range(size):
        if reverse_suffix_array[i] > 0:
            j = suffix_array[reverse_suffix_array[i] - 1]
            while i != size - h and j != size - h and str[i + h] == str[j + h]:
                h += 1
            lcp[reverse_suffix_array[i]] = h
            if h > 0:
                h -= 1
    if size > 0:
        lcp.pop(0)
    return lcp
```

Code Search

```
# simple binary search returns the first appearance in the suffix array
def simple_binary_search(suffix_array, text, searchterm):
    global COUNTER

    term_length = len(searchterm)
    text_length = len(text)

    if IS_COUNTER:
        COUNTER += 1

    left = 0
    right = text_length - 1

    while right - left > 1:
        if IS_COUNTER:
            COUNTER += 1

        middle = (left + right) // 2

        # search left right border
        if compare_serachterm_anyterm(searchterm, text[suffix_array[middle]:], term_length):
            right = middle

        else:
            left = middle

    # right is the last value that gets updated
    result = right

    return result
```

```
# counts the words with an lcp and the first appearance in a suffix array
def count_word_appearance(suffix_array, text, searchterm):
    global COUNTER

    position_suffix_array = simple_binary_search(suffix_array, text, searchterm)
    lcp_array = create_lcp_array(text, suffix_array)
    counter = 0

    for n in lcp_array[position_suffix_array:]:
        if IS_COUNTER:
            COUNTER += 1

        if n < len(searchterm):
            counter += 1
            break

        counter += 1

    return counter
```