Algorithmen und Programmierung 3, WS 2019/2020 — 6. Übungsblatt

Abgabe bis Freitag, 29. November 2019, 12:00 Uhr, in die Fächer der Tutoren

46. Adjazenzlisten, 7 Punkte

    Wie kann man auf einfache Art überprüfen, ob ein ungerichteter Graph, der mit Adjazenzlisten gespeichert ist, ein *einfacher Graph* ist (keine mehrfachen Kanten zwischen den gleichen zwei Knoten enthält)? Versuchen Sie, mit $O(m+n)$ Zeit und mit möglichst wenig zusätzlichem Speicher auszukommen.

    Schreiben Sie Ihren Algorithmus in Pseudocode.

47. Suffixbaum und Verschiebefunktion, 6 Punkte

    (a) (0 Punkte) Bestimmen Sie den Suffixbaum und

    (b) (6 Punkte) die Verschiebefunktion für folgende Zeichenkette:

    ```
                                1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
    aabbbcccabbcccaaabccaaabbbcbbcccaaab
    ```

48. Schnellere Teilwortsuche, 0 Punkte

    Es soll ein Muster der Länge m in einem Text der Länge n gesucht werden. Frau R. S. Tal hat ein neues Teilwortsuchverfahren entwickelt, und sie preist es damit an, dass es nach einer Vorverarbeitung des Musters „im günstigsten Fall“ nur m Zeichen des Textes anschauen muss, um zu entscheiden, dass das Muster im Text vorkommt und nur $\lceil n/m \rceil - 1$ Zeichen, um zu entscheiden, dass das Muster im Text nicht vorkommt.

    Beweisen Sie, dass das unmöglich ist.

49. Teilwortsuche, 7 Punkte

    Wie kann man aus der Verschiebefunktion f des Wortes $z = xy$ ablesen, ob x ein Teilwort von y ist, wenn man $|x| = m$ und $|y| = n$ kennt?

50. (0 Punkte) Für zwei Wörter x und y der Länge n ist x eine *zyklische* Verschiebung von y, wenn man $x = ab$ und $y = ba$ für zwei Wörter a und b schreiben kann. Wie kann man in linearer Zeit feststellen, ob x eine zyklische Verschiebung von y ist? (Man kann diese Frage auf das Teilwortproblem zurückführen.)

51. Verbesserung der Verschiebefunktion, 0 Punkte

    Die verbesserte Verschiebefunktion $\hat{f}$ zum Suchen eines Musters $b_0 b_1 b_2 \dots b_{m-1}$ ist für $j = 1, \dots, m-1$ folgendermaßen definiert:

    $$\hat{f}(j) := \max\big(\{\, k \mid 0 \le k < j-1,\ b_0 \dots b_{k-1} = b_{j-k} \dots b_{j-1} \text{ und } b_k \ne b_j \,\} \cup \{0\}\big)$$

    Für $j = m$ stimmt $\hat{f}(m)$ mit der ursprünglichen Verschiebefunktion $f(m)$ aus der Vorlesung (ohne die Bedingung „$b_k \ne b_j$“) überein.

    (a) Berechnen Sie die verbesserte Verschiebefunktion des Musters aus Aufgabe 47.

    (b) Begründen Sie, warum die Teilwortsuche mit der verbesserten Verschiebefunktion $\hat{f}$ immer noch korrekt ist.

    (c) Zeigen Sie, dass beim Suchen mit der verbesserten Verschiebefunktion auf keinen Fall mehr Vergleiche der Form $a_i = b_j$ durchgeführt werden als mit der ursprünglichen Verschiebefunktion f. Finden Sie ein Beispiel, bei dem echt weniger Vergleiche notwendig sind.

    (d) Auf der rechten Seite kann $\{0\}$ sogar durch $\{-1\}$ ersetzt werden; man muss dann allerdings die Suchprozedur anpassen.

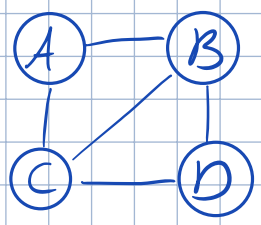
    (e) Schreiben Sie einen Algorithmus zum Berechnen von $\hat{f}$ in linearer Zeit.

46. Adjazenzlisten, 7 Punkte

Wie kann man auf einfache Art überprüfen, ob ein ungerichteter Graph, der mit Adjazenzlisten gespeichert ist, ein *einfacher Graph* ist (keine mehrfachen Kanten zwischen den gleichen zwei Knoten enthält)? Versuchen Sie, mit $O(m+n)$ Zeit und mit möglichst wenig zusätzlichem Speicher auszukommen.

Schreiben Sie Ihren Algorithmus in Pseudocode.

A {B, C}
B {D, C, A}
C {A, B, D}
D {C, B}



	A	B	C	D
A	0	1	1	0
B	1	0	1	1
C	1	1	0	1
D	0	1	1	0

sei n Anzahl an Knoten und m Anzahl an Kanten

sei xs die Liste, welche die Knotenlisten beinhalt.

Bsp. xs = [[B, C], [D, C, A], [A, B, D], [C, B]]

in

Pseudocode (xs):

```
for x in xs:                                   O(n·(m/n + 1)) = O(m+n)
    dicti = {}                                 O(1)
    k = 0                                      O(1)
    for i in x:                                O(m/n)
        dicti[i] = k                           O(1)
        k += 1                                 O(1)

    k = 0                                      O(1)
    for i in x:                                O(m/n)
        if (dicti[i] ≠ k): return True         O(1)
        k += 1                                 O(1)

return False
```

Die for-Schleifen sind voneinander abhängig, daher $O(m+n)$.
Dies Formal zu begründen fällt uns jedoch schwer

47. Suffixbaum und Verschiebefunktion, 6 Punkte

```
                  1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
a a b b b c c c a b b c c c a a a b c c a a a b b b c b b c c c a a a b
0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 2 2 3 0 0 1 2 2 3 4 5 6 0 0 0 0 0 1 2 2 3
```

49. Teilwortsuche, 7 Punkte

Wie kann man aus der Verschiebefunktion f des Wortes $z = xy$ ablesen, ob x ein Teilwort von y ist, wenn man $|x| = m$ und $|y| = n$ kennt?

In der Verschiebefunktion f (Liste) (Tabelle), ab den Index m bis m+n (bis zum Ende), die Werte mit m vergleiche. Falls ein Wert >= m dann ist x ein Teilwort von z

```
Barcodecode (m, n, f):
    for i in range (2m, m+n-1):
        if f[i] >= m : return True
    return False
```