Algorithmen und Datenstrukturen SoSe25

-Assignment 10-

Moritz Ruge
Matrikelnummer: 5600961
Lennard Wittenberg
Matrikelnummer: —

Juli 2025

1 Problem: Dynamisches Programmieren

Sei s eine Zeichenkette der Länge n. Sie vermuten, dass es sich bei s um einen deutschsprachigen Text handelt, bei dem die Leer- und Satzzeichen verloren gegan- gen sind (also zum Beispiel s = "werreitetsospaetdurchnachtundwind"), und Sie möchten den ursprünglichen Text rekonstruieren.

Dazu steht Ihnen ein Wörterbuch zur Verfügung, das in Form einer Funktion

$$dict : String \rightarrow Boolean$$

implementiert ist. dict(w) liefert true für ein gültiges Wort w, und false sonst (z.B. dict("blau") = true und dict("bsau") = false). Verwenden Sie dynamisches Programmieren, um einen schnellen Algorithmus zu entwickeln, der entscheidet, ob sich s als eine Aneinanderreihung von gültigen Wörtern darstellen lässt. Gehen Sie dabei folgendermaßen vor:

1. Definieren Sie geeignete Teilprobleme und geben Sie eine geeignete Rekursion an. Erklären Sie Ihre Rekursion in einem Satz.

2. Geben Sie Pseudocode für Ihren Algorithmus an.

3. Analysieren Sie die Laufzeit und Speicherplatzbedarf Ihres Algorithmus unter der Annahme, dass ein Aufruf von dict konstante Zeit benötigt.

4. Beschreiben Sie in einem Satz, wie man eine gültige Wortfolge finden kann, falls sie existiert.

Teilproblem Definition: Betrachte 'dp[i]' als boolsche Tabelle, wobei 'dp[i] = true', wenn das Präfix der Länge 'i' in gültige Wörter aufgeteilt werden kann.

Rekursion: Ein Präfix bis Index 'i' ist gültig ('dp[i] = true'), wenn es einen Index 'j ¡ i' gibt, so dass: 1. Das Präfix bis 'j' gültig ist ('dp[j] = true'). 2. Die Zeichenkette von 'j' zu 'i' ein gültiges Wort bildet (durch Aufruf der Funktion 'dict').

Pseudocode: "' function wordBreak(s): n = s.length dp[0..n] boolesch, alle auf false initialisieren prev[0..n] integer-Array mit -1, speichert Startposition des letzten Wortes

if dict(s) == true: return [true], [] // Vollständiges Wort ohne Aufteilung

dp[0] = true // leeres Präfix ist gültig for i from 1 to n: dp[i] = false for j from 0 to i-1: if dp[j]: wort = s.substr(j, i-j) if dict(wort) == true: dp[i] = true prev[i] = j // Startposition des aktuellen Wortes speichern break // Ersten gültigen Splitpunkt finden

return dp[n], prev // Wenn true, gibt es einen gültigen Split "'

Laufzeitanalyse: - **Zeitkomplexität:** O(n²), da für jede Position 'i' alle vorherigen Positionen 'j ¡ i' geprüft werden. Jeder Schritt erfordert maximal 'n - j + 1' Zeichenvergleiche (Substrings Erzeugung). - **Raumkomplexität:** O(n) für die Arrays 'dp' und 'prev'.

Wortfolge finden mit 'prev': Falls 'dp[n] == true', folgt eine gültige Aufteilung durch Rücktracking:

```
function getWordList(s, prev):
  i = n
  word_list = []
  while i > 0:
  start_index = prev[i]
  wort_start = s.substr(start_index, i - start_index)
  wordBreakHelper(i)          // Extrahiere Woerter vom Ende zu Anfang durch
    Rueckwaertsgang
```

Erklärung zur Rückschlusskonstruktion: - Das 'prev'-Array speichert die Startposition des vorletzten gültigen Wortes. Indem man bei 'i = n' beginnt und sich schrittweise zu den vorherigen

Splitpunkten bewegt, kann das letzte gültige Wort bestimmt werden (Startindex ‘j‘, Endindex ‘i‘). - Beispiel: - ‘prev[5] = 2‘ → Zeichenkette von Index ‘2‘ bis ‘5‘ ist das letzte gültige Wort. - Setze dann ‘i := j‘ und wiederhole, um die vorherigen Wörter zu finden.

Ergänzung: - Das Problem prüft nur auf Existenz eines gültigen Splits. Wenn ein komplettes Wort ohne Aufteilung möglich ist (‘dict(s) == true‘), wird sofort ‘true‘ zurückgegeben.

2 Problem: Editierabstand

Der Editierabstand zwischen zwei Zeichenketten s und t ist die minimale Anzahl von Editieroperationen, um s nach t zu überführen. Es gibt drei Editieroperationen: (i) Einfügen eines Zeichens; (ii) Löschen eines Zeichens; und (iii) Ersetzen eines Zeichens durch ein anderes. Zum Beispiel beträgt der Editierabstand zwischen "APFEL" und "PFERD" drei: Lösche A, ersetze L durch R, füge D ein.

Beschreiben Sie einen Algorithmus, der den Editierabstand zwischen zwei Zeichenketten s und t in $O(k_l)$ Zeit berechnet, wobei s Länge k und t Länge l hat. Erklären Sie außerdem, wie man eine optimale Folge von Editieroperationen findet.

Hinweis: Benutzen Sie dynamisches Programmieren analog zum LCS-Problem. Betrachten Sie das jeweils letzte Zeichen in s und t und unterscheiden Sie drei Möglichkeiten: (a) überführe s nach t' und füge dann ein Zeichen an; (b) überführe s' nach t und lösche dann ein Zeichen; (c) überführe s' nach t' und ersetze dann ein Zeichen, falls nötig. Hierbei bezeichnen s' und t' jeweils s und t ohne den letzten Buchstaben.

3 Problem: Finden von Senken in Graphen

Betrachtet man die Adjazenzmatrixdarstellung eines Graphen $G = (V, E)$, dann haben viele Algorithmen Laufzeit $|V|2$. Es gibt aber Ausnahmen. Zeigen Sie, dass die Frage, ob ein gerichteter Graph G eine globale Senke — einen Knoten vom Eingrad $|V| - 1$ und Ausgrad 0 − hat, in Zeit $O(|V|)$ beantwortet werden kann, selbst wenn man die Adjazenzmatrixdarstellung von G (die ja selbst schon die Größe $O(|V|2$) hat) verwendet. Beweisen Sie Korrektheit und Laufzeit Ihres Algorithmus.

Hinweis: Sei A die Adjazenzmatrix von G und $u, v \in V, u \neq v$. Was folgt über u und v, wenn $A_{uv} = 1$ ist? Was, wenn $A_{uv} = 0$ ist?