Wortmuster

Aufgabenbeschreibung

Gegeben sind zwei Zeichenketten: `pattern` und `s`. Die Aufgabe besteht darin, zu überprüfen, ob `s` dem `pattern` folgt.

Hier bedeutet "folgen", dass es eine Bijektion (1-zu-1-Zuordnung) zwischen jedem Buchstaben in `pattern` und jedem nicht-leeren Wort in `s` gibt.

```
/**
 * @param {string} pattern
 * @param {string} s
 * @return {boolean}
 */
const wordPattern = (pattern, s) => {
    let patterns = patterns.split("");
    let words = s.split(" ");

    // Hier kommt deine Funktions-Logik hin

    return true;
};
```

Beispiele

```
Eingabe: muster = "abba", s = "hund katze katze hund"
Ausgabe: true
Erklärung: "a" wird "hund" zugeordnet, "b" wird "katze" zugeordnet.
```

```
Eingabe: muster = "abba", s = "hund katze maus hund"
Ausgabe: false
Erklärung: "a" und "b" können nicht beide "hund" zugeordnet werden.
```

```
Eingabe: muster = "aaaa", s = "hund hund hund hund"
Ausgabe: true
Erklärung: "a" wird "hund" zugeordnet.
```

```
Eingabe: muster = "abba", s = "hund hund hund hund"
Ausgabe: false
Erklärung: Es gibt keine 1-zu-1-Zuordnung zwischen "a", "b" und "hund".
```

```
Eingabe: muster = "abc", s = "hund katze maus"
Ausgabe: true
Erklärung: "a" wird "hund", "b" wird "katze" und "c" wird "maus"
zugeordnet.
```

```
Eingabe: muster = "aabb", s = "elefant elefant giraffe giraffe"
Ausgabe: true
Erklärung: "a" wird "elefant" und "b" wird "giraffe" zugeordnet.
```

```
Eingabe: muster = "abcba", s = "rot grün blau grün rot"
Ausgabe: true
Erklärung: "a" wird "rot", "b" wird "grün" und "c" wird "blau" zugeordnet.
```

```
Eingabe: muster = "abaa", s = "katze hund katze hund"
Ausgabe: false
Erklärung: "a" kann nicht sowohl "katze" als auch "katze hund" zugeordnet
werden.
```

```
Eingabe: muster = "abcd", s = "apfel birne apfel birne"
Ausgabe: false
Erklärung: Es gibt nicht genug eindeutige Wörter in s für jedes Zeichen in
muster.
```

```
Eingabe: muster = "aabbcc", s = "eins eins zwei zwei drei drei"
Ausgabe: true
Erklärung: "a" wird "eins", "b" wird "zwei" und "c" wird "drei"
zugeordnet.
```

Lösungsvorschlag

- Die Zeichenkette in einzelne Wörter aufgeteilen.
- prüfen, ob die Länge des Musters mit der Anzahl der Wörter übereinstimmt.
- Zwei Maps verwenden, eine für Muster-zu-Wort-Zuordnung und eine für Wort-zu-Muster-Zuordnung.
- Das Muster und die Wörter parallel durchlaufen.
- Für jedes Muster-Wort-Paar prüfen, ob es bereits eine Zuordnung gibt.
    - Falls ja, muss diese konsistent sein.

    - Falls nein, wird eine neue Zuordnung erstellt, sofern das Wort noch nicht einem anderen Muster zugeordnet ist.
- Die Funktion gibt true zurück, wenn das Muster vollständig und konsistent auf die Wörter abgebildet werden kann, andernfalls wird false zurückgegeben.

Das musst du lernen

Für diese Aufgaben werden Maps verwendet, schau dir mal die Erklärung dazu an.

Eine Map in JavaScript ist eine Sammlung von Schlüssel-Wert-Paaren, bei der sowohl die Schlüssel als auch die Werte beliebige Datentypen sein können. Maps bieten mehrere Vorteile gegenüber herkömmlichen Objekten, insbesondere in Bezug auf die Flexibilität der Schlüssel und die Einfügereihenfolge der Elemente.

Eigenschaften und Methoden von Maps

- **Erstellung**: Eine Map wird mit dem Konstruktor `new Map()` erstellt. Du kannst optional ein Array von Arrays übergeben, um die Map zu initialisieren.
- **Einfügen**: Mit der Methode `set(key, value)` wird ein neues Schlüssel-Wert-Paar hinzugefügt oder ein vorhandener Schlüssel aktualisiert.
- **Abrufen**: Mit `get(key)` wird der Wert für einen bestimmten Schlüssel abgerufen.
- **Prüfen**: Mit `has(key)` wird überprüft, ob ein Schlüssel in der Map vorhanden ist.
- **Löschen**: Mit `delete(key)` wird ein Schlüssel-Wert-Paar entfernt.
- **Größe**: Die Anzahl der Elemente in der Map kann mit der Eigenschaft `size` abgefragt werden.
- **Iterieren**: Mit Methoden wie `keys()`, `values()`, `entries()` und `forEach()` kannst DU über die Map iterieren.

Beispiel

```
const map = new Map();

// Einfügen von Schlüssel-Wert-Paaren
map.set("schlüssel1", "wert1");
map.set(2, "wert2");
map.set(true, "wert3");

// Abrufen von Werten
console.log(map.get("schlüssel1")); // 'wert1'
console.log(map.get(2)); // 'wert2'

// Prüfen, ob ein Schlüssel vorhanden ist
console.log(map.has(true)); // true

// Löschen eines Schlüssel-Wert-Paares
map.delete(2);
console.log(map.has(2)); // false

// Anzahl der Elemente in der Map
console.log(map.size); // 2

// Iterieren über die Map
```

```
map.forEach((wert, schlüssel) => {
    console.log(`${schlüssel}: ${wert}`);
});
```

Maps sind besonders nützlich, wenn Du eine Sammlung von Schlüssel-Wert-Paaren benötigst, bei der die Schlüssel nicht auf Strings beschränkt sind oder wenn Du die Einfügereihenfolge beibehalten möchtest.

Vorgehen

- Projekt auf GitHub anlegen
- dieses Dokument im Projekt mit aufnehmen
- Lösung als Pseudo-Code erstellen
- Pseudo-Code in JavaScript umsetzen
- alle 10 Testfälle aus dem Dokument durchführen und Ergebnisse vergleichen
- Zeitaufwand 90 Minuten

Hinweis

- Versuche die Lösung selbst zu entwicklen
- Verwende kein fertige Lösung, die KI generiert wurde