Rabattierte Preisberechnung

In einem Geschäft wird ein spezielles Rabattsystem eingeführt. Deine Aufgabe ist es, ein Programm zu schreiben, das die endgültigen prices nach Anwendung des Rabatts berechnet.

Aufgabenstellung

Gegeben ist ein Array `prices`, das die ursprünglichen prices von n Produkten enthält. Das Rabattsystem funktioniert wie folgt:

- Für jedes Produkt i wird geprüft, ob es ein Produkt j gibt, für das gilt: j > i und prices[j] <= prices[i].
- Wenn ein solches Produkt j existiert, erhält das Produkt i einen Rabatt in Höhe von prices[j].
- Falls mehrere Produkte für einen Rabatt in Frage kommen, wird das Produkt mit dem kleinsten Index j gewählt.
- Wenn kein Rabatt möglich ist, bleibt der ursprüngliche Preis bestehen.

Dein Programm soll ein Array `finalPrices` zurückgeben, wobei `finalPrices[i]` den Preis des i-ten Produkts nach Anwendung des Rabatts darstellt.

```
/**
 * @param {number[]} prices
 * @return {number[]}
 */
var finalPrices = function(prices) {
      // Deine Funktions-Logik
};
```

Beispiele

1. Eingabe: prices = [8, 4, 6, 2, 3] Ausgabe: [4, 2, 4, 2, 3] Erklärung:
    - Produkt 0: Rabatt von 4, Endpreis = 8 - 4 = 4
    - Produkt 1: Rabatt von 2, Endpreis = 4 - 2 = 2
    - Produkt 2: Rabatt von 2, Endpreis = 6 - 2 = 4
    - Produkt 3: Kein Rabatt, Endpreis = 2
    - Produkt 4: Kein Rabatt, Endpreis = 3
2. Eingabe: prices = [10, 1, 1, 6] Ausgabe: [9, 0, 1, 6]
3. Eingabe: prices = [1, 2, 3, 4, 5] Ausgabe: [1, 2, 3, 4, 5]
4. Eingabe: prices = [5, 4, 3, 2, 1] Ausgabe: [1, 1, 1, 1, 1]
5. Eingabe: prices = [3, 3, 3, 3, 3] Ausgabe: [0, 0, 0, 3, 3]

Einschränkungen

- 1 <= prices.length <= 500

- 1 <= prices[i] <= 1000

Aufgabe

Entwickle eine Funktion, die das Array `prices` als Eingabe erhält und das Array `finalPrices` zurückgibt.

Hier ist eine Schritt-für-Schritt-Anleitung zur Lösung der Aufgabe:

1. Vorbereitung:
    - Erstelle eine Kopie des Eingabe-Arrays `prices` und nenne sie `finalPrices`.
    - Initialisiere einen leeren Stack zur Speicherung von Indizes.
2. Durchlaufe das Array:
    - Gehe von links nach rechts durch das Array `prices`.
3. Für jeden Preis:
    - Solange der Stack nicht leer ist und der Preis am oberen Index des Stacks größer oder gleich dem aktuellen Preis ist:
        - a. Entferne den obersten Index vom Stack.
        - b. Berechne den rabattierten Preis für diesen Index und speichere ihn in `finalPrices`.
    - Füge den aktuellen Index zum Stack hinzu.
4. Ergebnis:
    - Nach der Verarbeitung aller Elemente enthält `finalPrices` die rabattierten prices.
    - Gib das `finalPrices`-Array zurück.

Erklärung:

- Der Stack speichert Indizes von Produkten, für die noch kein Rabatt gefunden wurde.
- Wenn ein günstigeres oder gleichpreisiges Produkt gefunden wird, können Rabatte für alle teureren Produkte im Stack berechnet werden.
- Die Verwendung eines Stacks gewährleistet, dass immer der nächstgelegene niedrigere oder gleiche Preis für den Rabatt verwendet wird.

Zeitkomplexität: O(n), da jedes Element höchstens einmal auf den Stack gelegt und einmal vom Stack entfernt wird. Raumkomplexität: O(n) für den Stack im ungünstigsten Fall.

Diese Lösung verwendet einen monoton steigenden Stack, um effizient die nächsten kleineren oder gleichen Elemente zu finden. Der Algorithmus durchläuft das Array einmal und verarbeitet jedes Element in konstanter amortisierter Zeit.

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