Aufgabe 2: Schwierigkeiten

Team-ID: 00433

Team-Name: Was?

Bearbeiter dieser Aufgabe:
Mathieu de Borman

18. Oktober 2024

Inhaltsverzeichnis

1 Interpretation der Aufgabenstellung

Aufgabe ist es, das Programm auch für Konflikte auszulegen. Deswegen wird bei der Ausgabe ein besonderer Syntax verwendet, um diese darzustellen (Siehe 3.3 Ausgeben).

2 Lösungsidee

Es wird sich gemerkt, welcher Buchstabe wie häufig größer ist als der andere, um anschließend anhand dessen zu ermitteln, in welcher Reihenfolge die neue Klausur ausgegeben werden sollte

2.1 Laufzeitanalyse

Die Laufzeitkomplexität hängt von zwei Parametern ab, nämlich der Anzahl der Klausuren n und der Länge der längsten Klausur l. Das Programm hat eine Laufzeitkomplexität von $O(n \cdot l^3)$ bei einer Speicherkomplexität von $O(n \cdot l^2)$.

3 Umsetzung

Die Lösungsidee ist in JavaScript implementiert.

3.1 Einlesen

Die Eingabe wird über ein Eingabefenster (prompt) ausgelesen. Anschließend werden alle Zahlen rausgefiltert. Jede Klausur wird in eine Hashmap eingepflegt. In der Hashmap wird gezählt, wie häufig ein Buchstabe größer ist als der andere.

3.2 Sortieren

Hierfür wird eine rekursive Funktion (Funktion sort()) aufgerufen, die immer nach dem Buchstaben sucht, der am wenigsten kleinere Buchstaben im Vergleich zu sich selbst hat. Danach wird dieser gespeichert und mit den restlichen Buchstaben wird die Funktion wieder aufgerufen. Wenn der kleinste Buchstabe gefunden wurde, dieser aber größer als andere ist, dann werden diese als Konflikte in der Hashmap gespeichert.

3.3 Ausgeben

Die sortierte Klausur wird formatiert und in die Konsole ausgegeben. Ein = bedeutet, dass der Buchstabe links davon mit dem rechts davon getauscht werden kann, ohne weitere Konflikte auszulösen. Bei einer solchen Bewegung fallen aber alle anliegenden = weg. Wenn eine Klammer mit Buchstaben vor oder nach einem Buchstaben steht, dann bedeutet das, dass die Buchstaben in der Klammer im direkten Vergleich zu dem Buchstaben kleiner bzw. größer sein müssen. Jedoch werden sie nicht so angeordnet, da das nicht weniger Konflikte auslösen würde. Bsp.: schwierigkeiten6.txt

4 Beispiele

Wir rufen nun das JavaScript-Programm mit den verschiedenen BWINF-Eingabedateien auf. Diese Dateien sind in demselben Ordner wie die Programmdatei. Das Programm wird mit Hilfe des Browsers ausgeführt. In dem Dialogfeld kann man dann die Tests eingeben. Die Ausgabe kann man in der Konsole auslesen. schwierigkeiten6.txt ist ein eigenes Beispiel um den Syntax der Konflikte zu veranschaulichen. Das Programm terminiert für alle getesteten Eingaben in weniger als 5ms auf einem gewöhnlichen PC.

4.1 Lösungen

4.1.1 schwierigkeiten0.txt

```
B = E < D < F < C
```

4.1.2 schwierigkeiten1.txt

```
A < C = G < F < D
```

4.1.3 schwierigkeiten2.txt

```
D = E < A < B = F < G
```

4.1.4 schwierigkeiten3.txt

```
B = C < D < A < E < F < G = I < L < H < J < K = M = N
```

4.1.5 schwierigkeiten4.txt

```
B < I = F < W = N
```

4.1.6 schwierigkeiten5.txt

```
H < C = R = S < E < N < M = O < J < L < F < V < D < G = Z < Q < K = (X)B < W < (X)I < Y = (X)U <
(T)P < X(B,I,U) < A = T(P)
```

4.1.7 schwierigkeiten6.txt

```
(C)A < B < C(A)
```

5 Quellcode

```
//Lese Eingabe aus
let input = (prompt("Schwierigkeiten") || "0").split("")
let compare = {}
let test = []
//letter speichert ob die das letzte Zeichen ein Buchstabe war
//splitter speichert ob in dieser Zeile schon ein kleiner als Zeichen war
let letter, splitter = false
//Filtere alle Zahlen raus
for (let i = 0; i < input.length; i++) {
    if ((input[i] || 0) != 0 && (Number(input[i]) || 0) == 0) {
        //Wenn ein Buchstabe ins System eingefügt wird
        if (input[i] != "<") {
            //Und eine Neue Zeile Anfängt
            if (letter && splitter) {
                //Speichere die verhältnisse der Zeile in der Hashmap
                for (let j = 0; j < test.length - 1; j++) {
                    for (let l = j + 1; l < test.length; l++) {
                        compare[test[j] + test[l]] = compare[test[j] + test[l]] + 1 || 1
                    }
                }
                splitter = false
                test = []
            }
            test.push(input[i])
            letter = true
        } else {
            splitter = true
            letter = false
        }
    }
}
//Funktion um die Buchstaben des letzten Tests nach schwirigkeit zu sortieren
function sort(list) {
    //Wenn die liste nicht mehr sortiert werden kann, dann gebe sie zurück
    if (list.length <= 1) {
        return list[0] || " "
    }
    let smalInd = 0
    let smalNum = Number.POSITIVE_INFINITY
    let smalRef = []
    //Vergleiche jeden Buchstaben (i) mit jeden Buchsteben (j)
    for (let i = 0; i < list.length; i++) {
        let bigger = 0
        let ref = []
        for (let j = 0; j < list.length; j++) {
            //Wenn i>j ist dann merke dir das
            if ((compare[list[i] + list[j]] || 0) < (compare[list[j] + list[i]] || 0)) {
                bigger++
                ref.push(list[j])
            }
        }
        //Merke dir den Buchstaben, bei dem es die wenigsten konflikte gibt,
```

```
        //wenn es das kleinste ist
        if (bigger < smalNum) {
            smalNum = bigger
            smalInd = i
            smalRef = ref
        }
    }
    //Wenn es konflikte gibt dann speichere das in der Hashmap
    if (smalNum > 0) {
        for (let i = 0; i < smalRef.length; i++) {
            compare[list[smalInd] + " "] = compare[list[smalInd] + " "] || ""
            compare[smalRef[i] + "<"] = compare[smalRef[i] + "<"] || ""
            compare[list[smalInd] + " "] += smalRef[i] + ","
            compare[smalRef[i] + "<"] += list[smalInd] + ","
        }
    }
    //rufe diese Funktion für die restliche liste auf
    return [list.splice(smalInd, 1), ...sort(list)]
}
let sorted = sort(test)
let out = ""
//Funktion um alle Konflikte eines Buchstaben aufzulisten
function add(ref) {
    if (compare[ref] !== undefined) {
        out += `(${compare[ref].slice(0, -1)})`
    }
}
//Gehe durch jeden Buchsteben und notiere die Konflikte für jeden Buchstaben
for (let i = 0; i < sorted.length - 1; i++) {
    add(sorted[i] + " ")
    out += sorted[i]
    add(sorted[i] + "<")
    //Wenn der Folgende Buchstabe mit dem eigenen getauscht werden könnte,
    //dann trenne sie mit einem '='. Andernfalls mit einem '<'
    if (compare[sorted[i] + sorted[i + 1]] == compare[sorted[i + 1] + sorted[i]]) {
        out += " = "
    } else {
        out += " < "
    }
}
//füge die konfliktstellen des letzten Buchstabens hinzu
add(sorted[sorted.length - 1] + " ")
out += sorted[sorted.length - 1]
add(sorted[sorted.length - 1] + "<")
//Gebe das ergebnis in die console aus
console.log(out)
```