Junioraufgabe 1: Wundertüte

Team-ID: 00064

Team-Name: SpaceX

Bearbeiter dieser Aufgabe:
Fabian Lehmann

20. November 2023

Inhaltsverzeichnis

1 Lösungsidee

Das Ziel meines Programms ist es, die Differenz zwischen der Anzahl der einzelnen Spiele in den jeweiligen Tüten sowie die Differenz zwischen der Gesamtanzahl aller Spiele in den Tüten so gering wie möglich zu halten. Um die Differenz zwischen der Anzahl der einzelnen Spiele in den jeweiligen Tüten möglichst gering zu halten, werden zwei Variablen berechnet:

1. Anteil des jeweiligen Spiels pro Tüte: $\lfloor \frac{\text{Anzahl des jeweiligen Spiels}}{\text{Anzahl der Tüten}} \rfloor$
2. Anzahl des Rests des jeweiligen Spiels pro Tüte: Anzahl des jeweiligen Spiels mod Anzahl der Tüten

Für den ersten Teil, den Anteil des jeweiligen Spiels pro Tüte, addiert man diesen zu jeder Tüte. Dadurch hat jede Tüte bis zu diesem Punkt eine gleiche Anzahl des entsprechenden Spiels.
Für den zweiten Teil, beginnend von der Tüte mit den wenigsten Spielen, addiert man eins dazu, solange, bis man genau so oft eins zu den Tüten hinzugefügt hat, wie die Anzahl des Rests bzw. des Modul ist. Da wir den Rest auf die Tüten aufgeteilt haben ,indem wir immer zu den Tüten eins hinzugefügt haben, ist die Differenz zwischen der Anzahl des entsprechenden Spiel in allen Tüten maximal 1. Außerdem haben wir den Rest auf die Tüten mit der wenigsten Anzahl an Spielen aufgeteilt, dadurch ist die Differenz zwischen der Anzahl aller Spiele in den jeweiligen Tüte maximal 1.

2 Umsetzung

2.1 Implementierung

Um die Anzahl der Spiele verteilt auf die jeweiligen Tüten zu speichern, verwende ich eine zweidimensionalen Array.

```
int bags[numOfBags][numOfGames]

//numOfBags ist in diesem Fall die Anzahl der Tueten
//numOfGames ist die Anzahl der verschiedenen Spiele
```

Zuerst speichere ich die Anzahl der Spiele in einer Queue, da mir die LIFO Property zu Gunsten kommt, weil ich mit dem Spiel anfangen will, was als erstes in die Queue hinzugefügt wurde. Der erste Schritt ist zunächst auszurechnen, wie viele Spiele von der jeweiligen Spielsorte in alle Tüten müssen. Das machen wir mit einer ganzzahligen Division. Wenn das über 0 ist, iteriert man durch jede Tüte durch und addiert die Anzahl des Spiels dazu.

```
int count = games.front() / numbOfBags;
int remainderCount= games.front() % numbOfBags;
games.pop();

if(count){
    for(int i = 0;i<numOfBags;i++){
        bags[i][currentGame] += count;
    }
}
```

Jetzt müssen wir nur noch den Rest auf die passenden Tüten aufteilen. Dies machen wir, indem wir durch eine While-Schleife iterieren solange bis der Rest leer ist. Eine besondere Sache hierbei ist , man merkt sich die Position, wo man zuletzt den Rest hinzugefügt hat und weiß somit, dass diese Position plus 1 gleich die Position der Tüte mit den wenigsten Spielen ist.

```
while(remainderCount){
    currentBag = currentBag % numOfBags;
    bags[currentBag][currentGame] +=1;
    remainderCount--;
    currentBag++;
}
```

Für eine bessere Ausgabe zähle ich noch die gleichen Tüten und gebe die Anzahl und die passende Tüten aus. Dabei benutze ich eine Unordered Map, die die Anzahl der Vorkommen jeder Kombination von Spielen in den Taschen zählt. Hier wird als Key die Taschenkombination benutzt und als Value die Anzahl des Vorkommens.

```
std::unordered_map<std::string, int> arrayCount;

for(int i = 0; i < numOfBags;i++){
    std::string key;
    for(int j = 0;j<numOfGames;j++){
        key += std::to_string(bags[i][j]);
        if(j < numOfGames-1){
            key+= "␣,"; //Dadurch wird die Ausgabe lesbarer
        }
    }
    arrayCount[key]++;
}
```

2.2 Laufzeitanalyse

Zunächst analysieren wir die Laufzeit für einen Durchlauf für eine bestimmte Spielsorte unter der Annahme des Worst-Case-Szenarios. Im schlimmsten Fall wäre die Anzahl des entsprechenden Spiels (n + n-1), wobei n die Anzahl der Tüten ist. Man muss einmal durch alle Tüten iterieren um den count-Wert zu addieren. Das wäre eine Laufzeit von $O(n)$. Da der Rest maximal die Anzahl der Tüten minus 1 sein kann, iterieren wir ,um den Rest zu addieren, mit einer Laufzeit von $O(n-1)$ durch. Dies lässt sich zu $O(n+n-1)$ zusammenfassen und, da n der dominante Term ist, vereinfacht es sich zu $O(n)$. Das müssen wir für jede Spielsorte durchführen. Da kommt man auf eine Gesamtlaufzeit von $O(n*m)$, wobei n die Anzahl der Tüten und m die Anzahl der Spielsorten sind.

3 Beispiele

WICHTIG: Man muss die Beispiel-Dateien im gleichen Ordner haben und Windows benutzen.

3.1 Beispiel 0

```
1x : [1 ,1 ,1]
1x : [1 ,2 ,0]
1x : [2 ,1 ,1]
```

3.2 Beispiel 1

```
6x : [3 ,1 ,2]
```

3.3 Beispiel 2

```
5x : [1 ,1 ,0 ,1]
3x : [1 ,1 ,1 ,0]
1x : [2 ,1 ,0 ,0]
```

3.4 Beispiel 3

```
1x : [0 ,1 ,0 ,0 ,1]
2x : [0 ,1 ,0 ,1 ,0]
6x : [0 ,1 ,1 ,0 ,0]
2x : [1 ,1 ,0 ,0 ,0]
```

3.5 Beispiel 4

```
1x : [1 ,6 ,3 ,5 ,31 ,5]
1x : [1 ,7 ,2 ,5 ,31 ,5]
10x : [1 ,7 ,2 ,6 ,30 ,5]
1x : [2 ,6 ,2 ,6 ,30 ,6]
1x : [1 ,7 ,2 ,6 ,30 ,6]
2x : [2 ,6 ,3 ,5 ,30 ,6]
1x : [2 ,6 ,3 ,5 ,31 ,5]
```

3.6 Beispiel 5

```
6x : [0 ,0 ,2 ,1 ,2 ,0 ,2 ,0 ,1 ,1 ,1 ,1 ,0 ,1 ,2 ,2 ,0 ,1 ,1 ,2 ,0 ,2 ,1]
2x : [0 ,1 ,1 ,1 ,2 ,1 ,1 ,1 ,0 ,1 ,1 ,1 ,0 ,1 ,2 ,2 ,0 ,1 ,1 ,2 ,0 ,2 ,1]
1x : [0 ,1 ,1 ,2 ,1 ,1 ,1 ,1 ,0 ,1 ,1 ,1 ,0 ,1 ,2 ,2 ,0 ,1 ,1 ,2 ,0 ,2 ,1]
6x : [0 ,1 ,1 ,2 ,1 ,1 ,1 ,1 ,0 ,1 ,1 ,1 ,0 ,1 ,2 ,2 ,1 ,1 ,1 ,1 ,0 ,2 ,1]
1x : [0 ,1 ,1 ,2 ,1 ,1 ,1 ,1 ,1 ,0 ,1 ,1 ,0 ,1 ,2 ,2 ,1 ,1 ,1 ,1 ,1 ,1 ,1]
1x : [1 ,0 ,2 ,1 ,2 ,0 ,2 ,0 ,1 ,1 ,1 ,1 ,0 ,1 ,2 ,2 ,0 ,1 ,1 ,2 ,0 ,2 ,1]
5x : [0 ,1 ,1 ,2 ,1 ,1 ,1 ,1 ,0 ,1 ,1 ,1 ,0 ,1 ,2 ,2 ,0 ,2 ,0 ,2 ,0 ,2 ,1]
4x : [0 ,1 ,1 ,2 ,1 ,1 ,1 ,1 ,0 ,1 ,1 ,1 ,0 ,1 ,2 ,2 ,0 ,2 ,1 ,1 ,0 ,2 ,1]
5x : [1 ,0 ,2 ,1 ,2 ,0 ,2 ,0 ,1 ,1 ,1 ,1 ,0 ,1 ,2 ,1 ,1 ,1 ,1 ,2 ,0 ,2 ,1]
1x : [0 ,1 ,2 ,1 ,1 ,1 ,1 ,1 ,1 ,0 ,2 ,0 ,1 ,0 ,2 ,2 ,1 ,1 ,1 ,1 ,1 ,1 ,1]
3x : [1 ,0 ,2 ,1 ,1 ,1 ,2 ,0 ,1 ,0 ,2 ,1 ,0 ,1 ,1 ,2 ,1 ,1 ,1 ,2 ,0 ,1 ,2]
2x : [0 ,1 ,1 ,1 ,2 ,0 ,2 ,0 ,1 ,1 ,1 ,1 ,0 ,1 ,2 ,2 ,0 ,1 ,1 ,2 ,0 ,2 ,1]
9x : [1 ,0 ,2 ,1 ,1 ,1 ,1 ,1 ,1 ,0 ,2 ,1 ,0 ,1 ,1 ,2 ,1 ,1 ,1 ,2 ,0 ,1 ,2]
9x : [0 ,1 ,1 ,2 ,1 ,1 ,1 ,1 ,1 ,0 ,1 ,1 ,1 ,0 ,1 ,2 ,2 ,1 ,1 ,1 ,1 ,1 ,1]
3x : [1 ,0 ,2 ,1 ,1 ,1 ,1 ,1 ,1 ,0 ,2 ,1 ,0 ,0 ,2 ,2 ,1 ,1 ,1 ,2 ,0 ,1 ,1]
1x : [1 ,0 ,2 ,1 ,1 ,1 ,1 ,1 ,1 ,0 ,2 ,1 ,0 ,0 ,2 ,2 ,1 ,1 ,1 ,2 ,0 ,1 ,2]
8x : [1 ,0 ,2 ,1 ,1 ,1 ,1 ,1 ,1 ,0 ,2 ,0 ,1 ,0 ,2 ,2 ,1 ,1 ,1 ,1 ,1 ,1 ,1]
4x : [1 ,0 ,2 ,1 ,1 ,1 ,2 ,0 ,1 ,1 ,1 ,1 ,0 ,1 ,2 ,1 ,1 ,1 ,1 ,2 ,0 ,2 ,1]
9x : [0 ,1 ,2 ,1 ,1 ,1 ,1 ,1 ,1 ,0 ,1 ,1 ,1 ,0 ,2 ,2 ,1 ,1 ,1 ,1 ,1 ,1 ,1]
5x : [1 ,0 ,2 ,1 ,1 ,1 ,1 ,1 ,1 ,0 ,2 ,0 ,1 ,0 ,2 ,2 ,1 ,1 ,1 ,2 ,0 ,1 ,1]
11x : [1 ,0 ,2 ,1 ,1 ,1 ,2 ,0 ,1 ,1 ,1 ,1 ,0 ,1 ,2 ,1 ,1 ,1 ,1 ,2 ,0 ,1 ,2]
1x : [0 ,1 ,1 ,2 ,1 ,1 ,1 ,1 ,1 ,0 ,1 ,1 ,1 ,0 ,2 ,2 ,1 ,1 ,1 ,1 ,1 ,1 ,1]
```

4 Quellcode

4.1 Verteilung

```
int currentBag = 0;
int currentGame = 0;

while(!games.empty()){
    int count = games.front() / numOfBags;
    int remainderCount = games.front() % numOfBags;
    games.pop();

    if(count){
        for(int i = 0;i<numOfBags;i++){
            bags[i][currentGame] += count;
        }
    }

    while(remainderCount){
        currentBag = currentBag % numOfBags;
        bags[currentBag][currentGame] +=1;
        remainderCount--;
        currentBag++;
    }
    currentGame++;
}
```

4.2 Zählen und Ausgeben

```
//COUNT DER GLEICHEN TASCHEN BZW ARRAYS
std::unordered_map<std::string, int> arrayCount;

for(int i = 0; i < numOfBags;i++){
    std::string key;
    for(int j = 0;j<numOfGames;j++){
        key += std::to_string(bags[i][j]);
        if(j < numOfGames-1){
            key+= "␣,";
        }
    }
    arrayCount[key]++;
}

//PRINT ANZAHL DER GLEICHEN TASCHEN + DIE TASCHEN SELBER
for(const auto&pair: arrayCount){
    std::string key_bags = pair.first;
    int count = pair.second;
    std::cout << count << "x␣:␣[" << key_bags << "]\n";
}
```