Junioraufgabe 2: Container

Team-ID: 00772

Team: Aino Spring

Bearbeiter/-innen dieser Aufgabe:
Aino Spring

20. November 2022

Inhaltsverzeichnis

Lösungsidee

Jeder Container bekommt einen Wert. Dieser Wert ist von Anfang an 1. Es wird für jeden Term der Wert des schwereren Containers laut des Terms auf den Wert des anderen Containers plus 1 gesetzt, wenn er nicht schon größer ist. Dies wird so lange wiederholt, bis sich kein Wert mehr ändert. Der Container mit dem größten Wert ist am schwersten. Wenn mehrere Container diesen Wert besitzen, kann der schwerste nicht bestimmt werden.

Umsetzung

Erkennung des schwersten Containers

Die erste Funktion „Sort(terms []Term)“ sortiert die Container. Der erste und einzige Parameter ist ein „Term“-array (in diesem Fall ein slice). Ein „Term“ ist ein struct, welches zwei Elemente enthält (a und b). Sie sind die Namen von zwei Containern. Der Term {“a“, “b“} sagt aus, dass a größer ist als b. Alle in einem der Terme erwähnten Container bekommen einen Wert, welcher standartmäßig 1 ist. Dies wird mit einer Hashmap gemacht. Dann wird über alle Terme iteriert und der Wert des laut dem aktuellen Term größeren Container wird auf den Wert des anderen Containers plus 1 gesetzt, wenn er nicht schon größer ist. Dies wird so lange getan, bis sich kein Wert mehr verändert. Diese Werte werden zurückgegeben.

Die Funktion „Greatest(sortedMap map[string]int)“ findet den größten Container der im ersten Parameter angegebenen Hashmap. Dazu wird der größte Wert der Hashmap herausgefunden und danach wird überprüft, ob dieser Wert mehrmals auftritt. Fall ja, wird “-1“ als string zurückgegeben. Falls nein, das größte Element der Hashmap.

Wortimportierung

Die Funktion „ImportFiles()“ importiert die Liste aus Termen aus den in der config-Datei angegebenen Dateien. Die Dateien sind in der config-Datei als relative Pfade angegeben, jedoch können es auch absolute Pfade sein. Diese Dateien werden eingelesen und jede Zeile des Inhalts ist ein Term. Die Termlisten werden als 2-dimensionales array aus Termen (in diesem Fall wieder ein 2-dimensionaler slice aus Termen, zu welchen im obigen Punkt genauer eingegangen wird) zurückgegeben.

Alles zusammenfügen

In der main-Funktion wird alles zusammengefügt. Die Termlisten werden mithilfe der im obigen Punkt genannten Funktion „ImportFiles“ importiert und für jede Termliste wird der schwerste Container mit den Funktionen „Sort“ und „Greatest“ berechnet und jener wird mit dem Index der Termliste über die Standartausgabe (stdout) ausgegeben.

Beispiele

container0.txt:

Schwerster Container konnte nicht ermittelt werden.

container1.txt:

"4"

container2.txt:

Schwerster Container konnte nicht ermittelt werden.

container3.txt:

"5"

container4.txt:

"5"

Quellcode

Erkennung des schwersten Containers

Das Term struct:

```
type Term struct {
   a string
   b string
}
```

Die im 1. Punkt erwähnte Funktion:

```
func Sort(terms []Term) map[string]int {
   var valueMap = make(map[string]int)
   var oldMap = CopyMap(valueMap)
   for _, term := range terms {
      if valueMap[term.a] == 0 {
         valueMap[term.a] = 1
      }
      if valueMap[term.b] == 0 {
         valueMap[term.b] = 1
      }
      if valueMap[term.a] <= valueMap[term.b] {
         valueMap[term.a] = valueMap[term.b] + 1
      }
      if CompareMap(valueMap, oldMap) {
         break
      }
   }
   return valueMap
}
```

Die im 2. Punkt erwähnte Funktion:

```
func Greatest(sortedMap map[string]int) string {
   var greatest string
   for k, v := range sortedMap {
      if greatest == "" {
         greatest = k
         continue
      }
      if sortedMap[greatest] < v {
         greatest = k
      }
   }
   for k, v := range sortedMap {
      if v == sortedMap[greatest] && k != greatest {
         return "-1"
      }
   }
```

```
    return greatest
}
```

Wortimportierung

Der Pfad zur config-Datei:

```
const (
    CONFIG = "config"
)
```

Die im 3. Punkt erwähnte Funktion:

```
func ImportFiles() [][]Term {
    var termLists = make([][]Term, 0)
    data, err := os.ReadFile(CONFIG)
    if err != nil {
        log.Panic(err)
    }
    for _, line := range strings.Split(string(data), "\n") {
        line = strings.TrimSuffix(line, "\r")
        termListData, termListErr := os.ReadFile(line)
        if termListErr != nil {
            log.Panicf("%#v: %v", line, termListErr)
        }
        var termList = make([]Term, 0)
        for _, line := range strings.Split(strings.Replace(string(termList-
Data), "\r", "", -1), "\n") {
            if line == "" {
                continue
            }
            termList = append(termList, Term{strings.Split(line, " ")[0],
strings.Split(line, " ")[1]})
        }
        termLists = append(termLists, termList)
    }
    return termLists
}
```

Alles zusammenfügen

Die im letzten Punkt erwähnte Funktion:

```
func main() {
    var termLists = ImportFiles()
    for idx, termList := range termLists {
        var greatest = Greatest(Sort(termList))
        fmt.Printf("Greatest container from termlist %v: \n", idx)
        if greatest == "-1" {
```

```
            fmt.Printf("Greatest container could not be determined.")
        } else {
            fmt.Printf("%#v", greatest)
        }
        fmt.Print("\n")
    }
}
```