Aufgabe 4: Fahrradwerkstatt

Team-ID: 00534

Team: Monads

Bearbeiter/-innen dieser Aufgabe:
Leopold Jofer, Benjamin Hantschel, Robert Zeitter

20. November 2022

Inhaltsverzeichnis

Lösungsidee

Sowohl für die durchschnittliche als auch die maximale Wartezeit sind die Arbeitszeiten von Relevanz. Marc arbeitet pro Tag von 9 – 17 Uhr, d.h. acht Stunden reine Arbeitszeit und 16 weitere Stunde, in der die Werkstatt geschlossen ist.
Um nun die Wartezeit eines einzelnen Auftrages zu ermitteln sind mehrere Schritte von Nöten: Zunächst muss die jeweilige Bearbeitungsdauer ausgelesen werden. Außerdem muss stets die restliche Arbeitszeit des Tages berechnet werden. Dies dient dazu, um zu überprüfen, ob der Auftrag innerhalb des Tages erledigt werden kann. Übersteigt nun ein Auftrag diese Zeit müssen zu der eigentlichen Bearbeitungszeit weitere 16 Stunden hinzugefügt werden. Hierbei darf der Eingangszeitpunkt nicht vernachlässigt werden. Diesen subtrahiert man nun von der berechneten Wartezeit.
Diese anfallende Dauer eines einzelnen Auftrages muss nun zu allen Bearbeitungszeiten vorheriger Aufträge addiert werden. Dies wird solange wiederholt, bis der gewünschte Auftrag erfüllt ist.
Die durchschnittliche Wartezeit ergibt sich, indem man alle Zeiten der Warteschlange addiert. Anschließend wird diese Summe durch die Anzahl der Aufträge geteilt.
Die maximale Wartezeit entspricht der Wartezeit des letzten Elementes.

Methode 1: Nach Reihenfolge der Warteschlange

Hierbei werden die zunächst die älteren Aufträge abgearbeitet. Dadurch, dass älteren Aufträge weiter oben stehen, müssen bereits erledigte Aufträge lediglich als abgeschlossen markiert werden. Die Wartezeit ergibt sich wie gewohnt und es muss nichts weiter beachtet werden.

Methode 2: Nach reiner Arbeitszeit

Bei dieser Umsetzung muss zunächst die Bearbeitungsdauer überprüft werden. Umso kleiner diese ist, desto früher kann ein Auftrag bearbeitet werden.
Durch diese Lösung kann an einem Tag das Maximum an Aufträgen abgeschlossen werden, wodurch besonders kürzere Aufträge profitieren.
Dadurch, dass dieser Ansatz jedoch den Eingangszeitpunkt ignoriert, ist es möglich, dass Kunden mit einem großen Auftrag eine deutlich längere Wartezeit haben.

Methode 3: Kombination Bearbeitungsdauer und Eingangszeitpunkt

Bei diesem Ansatz wird zwischen der Bearbeitungsdauer und dem Eingangszeitpunkt gewechselt. Ein großer Vorteil hierbei ist, dass Kunden mit lange Bearbeitungsdauer nicht ewig auf die Fertigstellung warten müssen. Ebenfalls werden diese bedient, welche nur kurze Aufträge aufgeben haben.
Erwartung: Die maximale Wartezeit sollte kleiner ausfallen, als in Methode 1 und Methode 2, wohingegen die durchschnittliche Wartezeit sich wahrscheinlich nur geringfügig (in Vergleich zu Methode 2) ändern wird.

Umsetzung

Zunächst muss ein Struct erstellt werden, welches den Anfangszeitpunkt, die Bearbeitungsdauer und ein Boolean (ob der Auftrag bereits fertiggestellt ist) als Attribute besitzt. Dieses wird im späteren Verlauf für die Rechnungen herangezogen.

Methode 1:

Wir benötigen eine Liste, welche jegliche Wartezeiten speichert. Außerdem wird ein Verzeichnis aller Aufträge und eine Variable, welche immer die vorherige Aufgabe speichert, benötigt.
Sobald diese Variablen angelegt wurden beginnt eine Schleife. Diese geht durch alle Aufträge hindurch. Innerhalb einer Wiederholung wird mit Hilfe des vorherigen Auftrages nun die Wartezeit berechnet. Anschließend wird dieser Wert zu der Liste aller Wartezeiten hinzugefügt. Als letzter Schritt der Schleife wird nun die vorherige Aufgabe gleich dem momentanen Auftrag gesetzt.
Sobald die Schleife fertig ist, werden innerhalb der Liste alle Elemente summiert. Diese Summe entspricht der maximalen Wartezeit.
Die durchschnittliche Wartezeit lässt sich ermitteln, indem man die maximale Wartezeit durch die Anzahl der Elemente innerhalb der Liste teilt.

Methode 2:

Wie in Methode 1 wird wieder eine Liste für alle Wartezeiten erstellt. Dieses mal wird jedoch noch eine interne Zeit hinzugefügt, welche den Startwert 0 hat und mit einer globalen Uhrzeit verglichen werden kann.
Nun startet man wieder eine Schleife. Diese soll solange laufen, bis alle Aufträge als abgeschlossen markiert wurden (eine entsprechende Liste wird als Parameter übergeben).
Als ersten Schritt wird eine neue Variabel, welche sowohl den Auftrag selber, als auch den Index innerhalb der Liste speichert. Hierfür wird zuerst durch alle Aufträge durchiteriert. Anschließend

kann für jeden Auftrag der Index gespeichert werden. Ebenfalls müssen die Aufträge gefiltert werden, ob es den jeweiligen Auftrag bereits „gibt“ (Eingabezeitpunk muss kleiner gleich der internen Zeit sein) und ob dieser noch fertiggestellt werden muss. Sind diese Bedingungen erfüllt wird der Auftrag ausgewählt, welche die geringste Bearbeitungsdauer besitzt.
Findet das Programm bei der vorangegangenen Analyse noch keinen Auftrag, welcher die genannten Vorrausetzungen erfüllt, wird die interne Zeit um eins erhöht.
Falls jedoch ein Auftrag gefunden wurde werden der Index und der Auftrag zur weiteren Bearbeitung wieder geteilt. Um den Auftrag bedenkenlos bearbeiten zu können wird von diesem eine Kopie erstellt. Außerdem wird der Auftrag, innerhalb der Liste aller Aufträge, als bearbeitet markiert.
Nun wird die interne Zeit um die Bearbeitungsdauer des Auftrages erhöht. Um die Wartezeit zu ermitteln, wird der Eingangszeitpunkt von der internen Zeit (Fertigstellungszeitpunkt) subtrahiert und dieser Wert der List aller Wartezeiten hinzugefügt.
Die durchschnittliche und maximale Wartezeit kann nun wie in Methode 1 berechnet werden.

Methode 3:

Je nach Abarbeitungsmethode funktioniert die Implementation unterschiedlich. Für den ersten Mechanismus ist keine simulierte Zeit notwendig, denn es reicht, wenn sich die Simulation von Aufgabe zu Aufgabe “hangelt”, also nur die Zeitdifferenz zwischen der Beendigung der letzten Aufgabe und der Beendigung der aktuellen berechnet.

Beispiele

Datei/Eingabe	Durchschnittliche Wartezeit	Maximale Wartezeit
fahrradwerkstatt0.txt	Methode 1: 3382 Methode 2: 2049 Methode 3: 2072	Methode 1: 27130 Methode 2: 8501 Methode 3: 8501
fahrradwerkstatt1.txt	Methode 1: 553 Methode 2: 426 Methode 3: 448	Methode 1: 11340 Methode 2: 2928 Methode 3: 2745
fahrradwerkstatt2.txt	Methode 1: 2205 Methode 2: 1434 Methode 3: 1571	Methode 1: 18706 Methode 2: 10296 Methode 3: 9769
fahrradwerkstatt3.txt	Methode 1: 1792 Methode 2: 1076 Methode 3: 1088	Methode 1: 15732 Methode 2: 5832 Methode 3: 5581
fahrradwerkstatt4.txt	Methode 1: 4293 Methode 2: 2897 Methode 3: 3105	Methode 1: 34063 Methode 2: 18886 Methode 3: 11804

Methode 3 bearbeitet die Aufträge durchschnittlich beinahe so schnell, wie Methode 2 (geringfügige Abweichungen). Besonders auffällig ist, dass die maximale Wartezeit eine rapide Verbesserung zu verzeichnen hat. Dadurch wird deutlich, dass Methode 3 die bereits oben genannten Erwartungen erfüllt und (was die maximale Wartezeit betrifft) sogar übersteigt.

Quellcode

```
#[derive(Clone)]

struct Task {
	start: u32,
	duration: u32,
	completed: bool,
}
fn better(mut tasks: Vec<Task>) {
	let mut time = 0;
	let mut wait_times: Vec<u32> = Vec::new();
	loop {
		let count = tasks.iter().filter(|task| !task.completed).count();
		if count == 0 {
			break;
		}
		let task = tasks
			.iter()
			.enumerate()
			.filter(|(_, task)| task.start < time && !task.completed)
			.min_by_key(|(_, task)| task.duration);
		match task {
			Some((index, task)) => {
				let task = task.clone();
				tasks[index].completed = true;
				time += task.duration;
				wait_times.push(time.checked_sub(task.start).unwrap_or(0));
			}
			None => {
				time += 1;
			}
		}
	}
}
```

```
fn simple(tasks: Vec<Task>) {
	let mut wait_times: Vec<u32> = Vec::new();
	let mut tasks_iter = tasks.iter();
	let mut prev_task = tasks_iter.next().unwrap();
	for task in tasks_iter {
		let wait_time = task
			.start
			.checked_sub(prev_task.duration + prev_task.start)
			.unwrap_or(0);
		wait_times.push(wait_time);
		prev_task = task;
	}
}

fn mixed(mut tasks: Vec<Task>) {
	let mut wait_times: Vec<u32> = Vec::new();

	let mut time = 0u32;

	let mut take_first = false;

	loop {
		let count = tasks.iter().filter(|task| !task.completed).count();
		if count == 0 {
			break;
		}

		take_first = !take_first;

		let mut available_tasks = tasks
			.iter()
			.enumerate()
			.filter(|(_, task)| task.start < time && !task.completed);

		let task;

		if take_first {
			task = available_tasks.next();
		} else {
			task = available_tasks.min_by_key(|(_, task)| task.duration);
		}
```

```
                match task {
                        Some((index, task)) => {
                                let task = task.clone();
                                tasks[index].completed = true;
                                time += task.duration;
                                wait_times.push(time.checked_sub(task.start).unwrap_or(0));
                        }
                        None => {
                                time += 1;
                        }
                }
        }
}
```