Bundeswettbewerb Informatik
Runde 1

4 Fahrradwekstatt

Die Lösung dieses Problems beruht auf folgendem Prinzip.: Die Daten der Aufgabenstellung werden in ein `pandas.DataFrame` mit den Spalten `"recieved"`, `"duration"`, `"begin"`, `"completed"` und `"delay"`, welche den Zeitpunkt des Eintreffens eines Auftrages, dessen Länge, Beginn und Fertigstellung dessen Abarbeitung und die Wartezeit speichern.
Der folgende Algorithmus greift hintereinander jeden Auftrag des `DataFrames` auf, arbeitet ihn ab und verschiebt Folgeaufträge gegebenenfalls nach hinten. Dieser Ablauf bildet das Grundgerüst für die später Folgenden Erweiterungen.:

Nach dem nebenstehenden Schema wird über jedes Element, also jeden Auftrag iteriert. Eine späte Fertigstellung des ersten Auftrags bewirkt einen späten Beginn des zweiten Auftrags. Dies setzt sich bis zum Ende der Liste fort.

Nach dem Iterieren ist das Dataframe gefüllt mit den Daten zum Erhalt und Abschluss von Aufträgen. Hieraus berechnet sich dann in der Spalte `'delay` die Verzögerung für jeden Auftrag.

$$Verzoegerung = Abschluss - Erhalt$$

Das genutzte `pandas.DataFrame` stellt nun Mehtoden zur statistischen Auswertung bereit, die es ermöglichen Kennwerte der Größe Verzögerung berechnen zu lassen. So lässt sich die durchschnittliche Verzögerung über `df.mean()` anzeigen.

Dies ist an diesem Punkt allerdings noch nicht sinnvoll, da der Algorithmus noch nicht ganz im Sinne der Aufgabenstellung arbeitet. Es fehlt eine Implementierung der Arbeitszeiten.

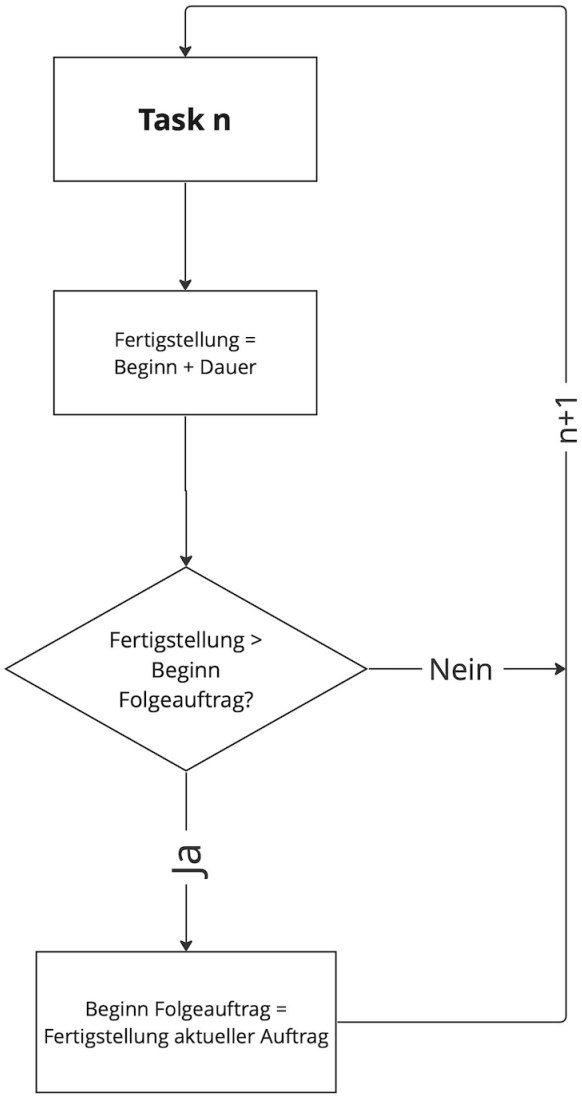

Abbildung 1: Grundablauf Algorithmus

4.1 Erweiterungen

4.1.1 Arbeitszeiten

Die Implementierung der genannten Arbeitszeiten findet an zwei Stellen im Algorithmus statt. Zum einen muss geprüft werden, ob ein aktueller Auftrag zum geplanten Arbeitsbeginn innerhalb der Öffnungszeiten liegt. Ist dies nicht der Fall, muss der Beginn des Auftrags auf den Beginn der Arbeitszeit des nächsten Tages verschoben werden. Dies leistet die Methode `get_valid_begin(t)`, die zu Beginn jeder Iteration aufgerufen wird.

```
1 def get_valid_begin(t):
2     task_time = t%MINUTES_PER_DAY
3     if(task_time in OPENING_HOURS):return t
4     else:
5         if(task_time < OPENING_TIME):return t + (OPENING_TIME - task_time)
6         elif(task_time >= CLOSING_TIME):return get_opening_time_of_next_day(t)
```

Die zweite Stelle, an der Arbeitszeiten implementiert werden müssen, ist die Auftragsdauer. So ist es möglich, dass ein besonders langer Auftrag in den definierten Feierabend hineinragt. Ist dies der Fall, so wird dieser Auftrag zu Ende der Arbeitszeit unterbrochen und am nächsten Morgen wieder aufgenommen. Diese Funktion macht die Berechnung des Zeitpunktes der Fertigstellung eines Auftrags wesentlich komplexer, weshalb auch diese Berechnung in eine Methode ausgelagert wird:

```
1 def get_completion_time(begin, time_remaining):
2     projected_completion = begin + time_remaining
3     if(not projected_completion > get_closing_time_of_day(begin)):
4         return projected_completion
5     else:
6         time_remaining = time_remaining-get_time_til_closing(begin)
7         begin = get_opening_time_of_next_day(begin)
8         return get_completion_time(begin, time_remaining)
```

Diese Methode operiert, indem sie zunächst prüft, ob das vorhergesagte Ende durch die Addition von Arbeitsbeginn und -Dauer noch in die Arbeitszeit des aktuellen Tages fällt. Ist dies der Fall, stimmt die einfache Rechnung und ihr Ergebnis wird zurückgegeben. Andernfalls wird von der noch ausstehenden Arbeitszeit die restliche Arbeitszeit des Tages abgezogen und die Mehtode erneut mit verkürzter Dauer für den nächsten Tag aufgerufen. Diese Rekkursion läuft bis die verbleibende Arbeitszeit in den aktuellen Tag passt.

4.1.2 Priorisierung kurzer Aufträge

Die Veränderung des Verfahrens vom First in-First-out Prinzip hin zur Priorisierung von kürzeren Aufträgen findet vor dem Überschreiben des Beginns des nächsten Auftrags statt. Hierzu muss vorher bestimmt werden, welches der nächste Auftrag ist, für den Fall, dass während der Bearbeitung

des aktuellen Auftrags mehrere neue hinzugekommen sind.
Diese Aufträge lassen sich durch ein wiederholen der Überprüfung aus dem Grundablauf bestimmen.

```
counter = 1
while(i+counter < len(df) and (df['completed'][i] > df['recieved'][i+counter])):
    df['begin'][i+counter] = df['completed'][i]
    counter+=1

df[i+1: i + counter] = df[i+1: i + counter].sort_values(by='duration')
```

Dies tut die `while` Schleife, die zunächst den Folgeauftrag betrachtet. Danach den Übernächstn, Über-Übernächstn und so weiter, bis ein Auftrag erreicht wird, der erst nach der Fertigstellung des aktuellen Auftrags eingeht. Dann wird der Teil des `DataFrame`, der so bestimmmt wurde nach der Auftragslänge sortiert.
Nun kann mit dem Folgeelement, wie bereits beschrieben weiterverfahren werden, wobei sicher ist, dass dieses der kürzeste verfügbare Auftrag ist.

4.2 Ergebnisse

Mit dem oben erläuterten Programm ergeben sich folgende Werte

Eingabe	Durchschnitt 1	Min 1	Max 1	Durchschnitt 2	Min 2	Max 2
fahrradwerkstatt0.txt	32753	1368	68771	16981	29	188734
fahrradwerkstatt1.txt	63535	13	128321	11884	13	433563
fahrradwerkstatt2.txt	51194	14	110973	14813	14	327087
fahrradwerkstatt3.txt	30028	15	60821	17242	15	382016
fahrradwerkstatt4.txt	74427	202	167059	42200	202	363155

Es fällt auf, dass die durchschnittliche Wartezeit pro Auftrag für das zweite Auswahlverfahren der Aufträge deutlich reduziert ist. Allerdings fällt auch auf, dass die maximale Wartezeit pro Auftrag wesentlich höher ist, was eine Unzufriedenheit der Kunden bewirken könnte. Die höhere maximale Wartezeit wird dadurch verursacht, dass lange Aufträge durch immer wieder priorisierte, kurze Aufträge immer weiter aufgeschoben werden.

4.3 Modellkritik und optimierte Verfahren

Die Kenngröße der durchschnittlichen Wartezeit pro Auftrag ist als Maß für die Kundenzufriedenheit ungeeignet. Dies liegt daran, dass ein Kunde, der einen 300 minütigen Auftrag erteilt erwartet 300 Minuten mindestens warten zu müssen. Ein Künde, der hingegen einen 5-Minuten Auftrag hat und 300 Minuten warten muss wird allerdings Unzufrieden sein.
Die Zufriedenheit der Kunden Z kann also nicht proportional zu den absoluten Wartezeiten W_a sein. Eine geeignetere Metrik wäre eine relative Wartezeit W_r gemessen auch an der Arbeitszeit D eines Auftrages. Es gilt

$$\frac{1}{Z} \approx W_r = \frac{W_a}{D} \tag{1}$$

Berechnet man diese Metrik für die beiden Verfahren ergeben sich folgende Werte für die relative Wartezeit W_r

Eingabe	W_r1	Min 1	Max 1	W_r2	Min 2	Max 2
fahrradwerkstatt0.txt	150	2.36	3902	30.8	1.812500	1329
fahrradwerkstatt1.txt	694	1	9335	18	1	261
fahrradwerkstatt2.txt	1713	1	8521	169	1	1024
fahrradwerkstatt3.txt	429	1	3508	37	1	279
fahrradwerkstatt4.txt	614	1	8931	47	1	363

Obwohl auch diese Metrik die These der Kundenzufriedenheit stützt, wiegt eine negative Stimme oft mehr als mehrere positive. Somit muss das Auswahlverfahren der Aufträge dahingehend angepasst werden, dass auch bei einer niedrigen durchschnittlichen (relativen) Wartezeit Wartezeit, Maximalwerte keine übermäßige Unzufriedenheit bewirken können. Dies wird durch das Anpassen der Sortierkriteriums erreicht. Es soll nicht mehr nach Auftragsdauer, sondern nach relativer Wartezeit sortiert werden. Hierzu wird die Spalte `priority` dem Dataframe hinzugefügt. Die Priorität P eines Auftrags wird berechnet, wenn der Algorithmus entscheiden muss, welchen Auftrag er als nächstes auswählt. Sie berechnet sich, wie folgt

$$P = \frac{W_a}{D} \tag{2}$$

Mit dieser Methode ergeben sich folgende Werte, visualisiert in einem Histogramm (arithmetisches Mittel als gestrichelte Linie)

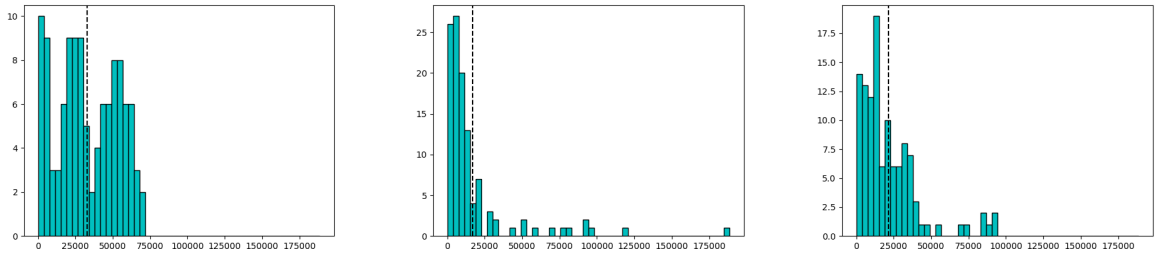

Abbildung 2: Histogramm Delay von "fahrradwerkstatt0.txt"mit Verfahren 1, 2 und 3

Die Auswertungen der Verfahren in einem Histogramm validiert die getätigte Aussage, dass Verfahren 2 im Vergleich zu Verfahren 1 viele große Werte hat. Der neue Algorithmus 3 verhindert dies. Hierbei weist er eine durchschnittliche Verzögerung von $21518min$ auf. Algorithmus 3 verbindet somit die niedrige maximale Wartezeit von Algorithmus 1 mit der niederigen durchschnittlichen Wartezeit von Algorithmus 2.