Aufgabe 4: Fahrradwerkstatt

Team-ID: 00339

Team: SilverBean

Bearbeiter/-innen dieser Aufgabe:
Richard Tschirpke

19. November 2022

Inhaltsverzeichnis

Lösungsidee

Berechnung der Wartezeiten

Um die durchschnittliche und maximale Wartezeit eines Verfahrens zu berechnen, müssen die Wartezeiten der einzelnen Aufträge ermittelt werden, wobei die Verfahren sich nur in der Reihenfolge der Aufträge unterscheiden. Die Wartezeit eines einzelnen Auftrages ist die Differenz zwischen der Zeit der Fertigstellung F und der gegebenen Eingangszeit E. F ergibt sich aus der Summe der Startzeit S eines Auftrages und der Gesamtarbeitsdauer. Die Startzeit des eines Auftrages ist gleich der Zeit der Fertigstellung des vorherigen Auftrages, wenn dieser zur Zeit der Fertigstellung vorliegt. Sollte dies nicht der Fall sein, ist S gleich der Eingangszeit des nächsten Auftrages, der gestellt wurde. Sollte der nächste Auftrag außerhalb der Auftragszeit abgegeben werden (z.B. über eine Website), muss beachtet werden, dass die Startzeit immer innerhalb der Arbeitszeit liegen muss.

Die Gesamtarbeitsdauer ist gleich der Bearbeitungsdauer B und der Zeit, die zwischen einzelnen Arbeitstagen vergeht. Wenn kein Arbeitstag überschritten wird, also B kleiner ist als der Restarbeitstag R, dann gilt für den ersten Fall B < R:

$$F = S + B$$

Ist B > R, dann wird mindestens einmal ein Arbeitstag überschritten und zusätzlich so häufig, wie B-R ganzzahlig geteilt durch 480, also die Länge eines Arbeitstages ist. Das stammt daher, dass für alle 480min in B die Dauer eines Arbeitstages gefüllt wird, aber der erste Arbeitstag nur noch R min hat. Zwischen den Arbeitstagen liegen 960min. Somit ergibt sich für den 2. Fall B > R:

$$F = S + B + \left(1 + \left\lfloor \frac{B-R}{480} \right\rfloor\right) * 960$$

Zur Darstellung der ganzzahligen Division wird hier die Abrundungsfunktion verwendet, da B-R positiv ist und somit das Ergebnis der Abrundungsfunktion immer dem der ganzzahligen Division entspricht.

Die Verfahren

Die Verfahren unterscheiden sich in der Reihenfolge, in der Aufträge abgearbeitet werden. Somit sind die durchschnittliche und maximale Wartezeit der Verfahren unterschiedlich.

Im ersten Verfahren hier genannt first-come-first-served (FCFS) werden die Aufträge aufsteigend nach dem Eingangszeitpunkt abgearbeitet. Dadurch ist die maximale Wartezeit zwar niedrig, da kein Verfahren vor ein anderes gezogen bevorzugt wird, aber die durchschnittliche Wartezeit ist sehr hoch.

Im zweiten Verfahren, hier genannt shortest-job-next, kurz SJN, wird immer der vorliegende Auftrag bearbeitet, der die geringste Bearbeitungsdauer aufweist. Das SJN-Verfahren weist im Durchschnitt eine deutlich geringere durchschnittliche Wartezeit auf als das FCFS-Verfahren. Es kann jedoch dazu führen, dass Aufträge mit einer hohen Bearbeitungsdauer verhungern, wenn immer genug kurze Aufträge hinzukommen, sodass ein langer Auftrag nie oder erst sehr spät bearbeitet wird. Das führt zu höheren Wartezeiten für lange Aufträgen und somit zu einer hohen maximalen Wartezeit. Deshalb werden Kunden mit langen Aufträgen vermutlich nicht zufrieden sein.

Ein Verfahren, welches bevorzugt kurze Aufträge abarbeitet, aber Aufträge kaum verhungern lässt, ist highest-response-ratio-next, kurz HRRN. Dadurch steigen die Wartezeiten für kürzere Aufträge im Vergleich zu SJN, aber die Wartezeiten für lange Aufträge ist deutlich niedriger. In diesem Verfahren wird immer der Auftrag bearbeitet, der die höchste response ratio hat, welche sich aus der Bearbeitungsdauer und der bisherigen Wartezeit zusammensetzt. Die response ratio ist:

$$response\ ratio = \frac{Bearbeitungszeit + bisherige\ Wartezeit}{Bearbeitungszeit} = 1 + \frac{bisherige\ Wartezeit}{Bearbeitungszeit}$$

Die response ratio eines Auftrages steigt an, je länger er warten muss. Somit ist das Problem, dass ein Auftrag verhungert, nur dadurch gegeben ist, dass immer kürzere Aufträge kommen könnten und somit immer höhere response ratios vorhanden sind, als die eines langen Auftrages. Dies ist in einer Fahrradwerkstatt jedoch nicht gegeben.

Um die Wartezeiten der Verfahren besser miteinander zu vergleichen, lässt sich noch der Median verwenden, da dieser Auskunft über die Verteilung der Wartezeiten gibt. Hat Verfahren im Median eine Wartezeit von X min, dann hatte eine Hälfte der Aufträge eine kürzere oder gleiche Wartezeit, während die andere Hälfte länger oder gleich lange warten musste.

Umsetzung

Die Lösungsidee wurde in Python implementiert. Es wird davon ausgegangen, dass die Aufträge in der Reihenfolge eingetragen wurden, in der sie aufgegeben wurden, da dies in der Realität so geschähe und die Beispiele dies berücksichtigen.

Die Eingangszeitpunkte und Bearbeitungsdauern werden in separaten Arrays gespeichert. Da bearbeitete Aufträge aus den Arrays entfernt werden, sind die Arrays für jedes Verfahren zu kopieren. Dazu wird das Bibliotheksmodul copy importiert, mit welchem deepcopies eines Arrays erschaffen

werden können. Jedes Verfahren ist eine eigene Funktion, welche nacheinander die zu bearbeitenden Aufträge ermittelt und für diese simulation aufruft, um die Wartezeiten zu ermitteln.

Die Funktion simulation wird für jeden Auftrag aufgerufen. Sie berechnet die Zeit nach Fertigstellung eines Auftrages und daraus die Wartezeit des fertiggestellten Auftrages. Als Parameter werden die Arrays der Eingangszeitpunkte, Bearbeitungsdauer und Wartezeiten, der Index des zu bearbeitenden Auftrages und die letzte Zeit der Fertigstellung, welche der Startzeit des nächsten Auftrages entspricht, übergeben.

Beispiele

Das Python-Programm wird mit den verschiedenen BWINF-Eingabedateien und einem eigenen Beispiel aufgerufen. Das eigene Beispiel deckt die Sonderfälle ab, dass Aufträge zwischen den Arbeitszeiten aufgegeben werden und Aufträge genau zum Ende eines Arbeitstages fertiggestellt werden. Die Dateien liegen im gleichen Ordner wie die Programmdatei.

beispiel0.txt:
first_come_first_serve
Median 30021 Durchschnitt 32753.5 Maximum 68771
shortest_job_next
Median 8238 Durchschnitt 16981.280701754386 Maximum 188734
highest_response_ratio_next
Median 14751 Durchschnitt 21518.535087719298 Maximum 93882

beispiel1.txt:
first_come_first_serve
Median 58969 Durchschnitt 63535.65274151436 Maximum 128321
shortest_job_next
Median 1154 Durchschnitt 11883.921671018277 Maximum 433563
highest_response_ratio_next
Median 10124 Durchschnitt 21370.281984334204 Maximum 248126

beispiel2.txt:
first_come_first_serve
Median 53216 Durchschnitt 51194.48924731183 Maximum 110973
shortest_job_next
Median 5946 Durchschnitt 14813.52688172043 Maximum 327087
highest_response_ratio_next
Median 9879 Durchschnitt 19082.870967741936 Maximum 135612

beispiel3.txt:
first_come_first_serve
Median 31874 Durchschnitt 30028.927272727273 Maximum 60821
shortest_job_next
Median 4406 Durchschnitt 17242.831818181818 Maximum 382016
highest_response_ratio_next
Median 8833 Durchschnitt 19880.422727272726 Maximum 66263

beispiel4.txt:

first_come_first_serve
Median 66611 Durchschnitt 74427.52222222222 Maximum 167059
shortest_job_next
Median 10315 Durchschnitt 42200.9 Maximum 363155
highest_response_ratio_next
Median 34585 Durchschnitt 49841.12222222222 Maximum 193086

Quellcode

```
import copy

def main():
    # Einlese der Eingabedatei
    with open("beispiel0.txt", "r") as f:
        file = f.read()

    # Auftragszeit und Eingangszeit jeweils in Liste
    start = [int(s) for s in file.split()[::2]]
    work = [int(s) for s in file.split()[1::2]]

    # Anzahl der Auftraege, und die Zeit
    iterationen = len(start)
    Zeit = start[0]

    # fuer den Fall, dass Auftrag ausserhalb Arbeitszeit gestellt wurde
    if Zeit%1440 < 540:
        Zeit += 540 - Zeit%1440
    elif Zeit%1440 > 1020:
        Zeit += 1980 - Zeit%1440

    first_come_first_serve(start, work, iterationen, Zeit)
    shortest_job_next(start, work, iterationen, Zeit)
    highest_response_ratio_next(start, work, iterationen, Zeit)


def simulation(i, time_of_order, order_duration, wait, time):

    i_duration = order_duration[i]
    work_day_left = 1020 - time%1440
    if i_duration > work_day_left:
        time += i_duration + (1 + ((i_duration - work_day_left)//480)) * 960
    else:
        time += i_duration

    wait.append(time - time_of_order[i])
    time_of_order.pop(i)
    order_duration.pop(i)
```

```
    # Startzeit des naechsten Auftrages
    if len(order_duration) > 0 :
        time = max(time, time_of_order[0])
        if time%1440 < 540:
            time += 540 - time%1440
        elif time%1440 > 1020:
            time += 1980 - time%1440

    return time


def wait_results(wait, name):

    max_wait = max(wait)
    averagewait = sum(wait)/len(wait)
    # implementierung des medians benoetigt eine sortierte Liste
    for i in range(1, len(wait)):
        e = wait[i]
        j = i - 1
        while j >= 0 and e < wait[j]:
            wait[j + 1] = wait[j]
            j -= 1
        wait[j + 1] = e

    if len(wait)%2 == 0:
        median_wait = wait[len(wait)//2]
    else:
        median_wait = wait[len(wait)//2 + 1]

    # with open("wartezeiten.txt", "a") as f:
    #     f.write(f"{name}\nMedian{median_wait:7}\t\tDurchschnitt{average-
wait:20}\tMaximum{max_wait:20}\n\n")
    print(f"{name}\nMedian{median_wait:7}\t\tDurchschnitt{averagewait:20}\tMaxi-
mum{max_wait:7}\n")
    return median_wait, averagewait, max_wait


def first_come_first_serve(start, work, iterations, Zeit):
    wait = []
    time_of_order = copy.deepcopy(start)
    order_duration = copy.deepcopy(work)

    for n in range(iterations):
        Zeit = simulation(0, time_of_order, order_duration, wait, Zeit)

    return wait_results(wait, first_come_first_serve.__name__)
```

```
def shortest_job_next(start, work, iterations, Zeit):
    wait = []
    time_of_order = copy.deepcopy(start)
    order_duration = copy.deepcopy(work)
    for j in range(iterations):
        x = 0
        for i in range(len(time_of_order)):
            if time_of_order[i] > Zeit:
                x = i
                break
        else:
            x = len(time_of_order)

        n = 0
        for i in range(x):
            if order_duration[i] < order_duration[n]:
                n = i
        Zeit = simulation(n, time_of_order, order_duration, wait, Zeit)

    return wait_results(wait, shortest_job_next.__name__)

def highest_response_ratio_next(start, work, iterations, Zeit):
    wait = []
    time_of_order = copy.deepcopy(start)
    order_duration = copy.deepcopy(work)
    for j in range(iterations):
        x = 0
        for i in range(len(time_of_order)):
            if time_of_order[i] > Zeit:
                x = i
                break
        else:
            x = len(time_of_order)

        n = 0
        for i in range(x):
            if 1 + (Zeit - time_of_order[i]) / order_duration[i] > 1 + (Zeit -
time_of_order[n]) / order_duration[n]:
                n = i
        Zeit = simulation(n, time_of_order, order_duration, wait, Zeit)

    return wait_results(wait, highest_response_ratio_next.__name__)

if __name__ == "__main__":
    main()
```