Fahrradwerkstatt

❔ A4 👥 00122 👴 Leonhard Masche 🗓️ 9.09.2022

Lösungsidee

Die Idee zur Lösung ist, den Zeitpunkt der Aufgabe des Auftrags und dessen Dauer in einer Liste zu speichern. Nun werden die Aufträge mit den gegebenen Methoden (nach Zeitpunkt oder Dauer) bearbeitet und die Ergebnisse ausgegeben.

Umsetzung

Das Programm ist in Python umgesetzt und mit einer Umgebung ab der Version `3.6` ausführbar. Das Programm befindet sich in der Datei `program.py` in diesem Ordner.

Zuerst werden die Aufträge aus dem Beispiel in eine Liste geladen. Die Aufträge werden dabei als Tuple gespeichert.

Nun werden die Aufträge mit den gegebenen Methoden bearbeitet. Hier gibt zusätzlich zu den vorgeschlagenen Methoden `by_submit` (Die Aufträge werden in der Reihenfolge ihrer Aufgabe bearbeitet) und `by_duration` (Unter den verfügbaren Aufträgen wird immer der Kürzeste ausgewählt) auch noch `by_duration_resumable`, die im Folgenden beschrieben wird. Die Ergebnisse werden in einem DataFrame aus der Bibliothek `pandas` gespeichert, um eine formatierte Ausgabe zu erleichtern.

Zum Schluss werden die Ergebnisse formatiert ausgegeben.

Verbesserungen

Wie die Simulation zeigt, wird durch die Auswahl des Auftrags mit der kürzesten Dauer die Durchschnittszeit der Bearbeitung der Aufträge deutlich reduziert. Dadurch verschlechtert sich aber gleichzeitig die maximale Wartezeit, da längere Aufträge immer weiter 'nach hinten' in der Warteschlange geschoben werden. Kunden mit längeren Aufträgen werden daher unzufriedener sein, da ihre Aufträge zugunsten von kürzeren Aufträgen immer wieder "unfairerweise" weiter nach Hinten in der Warteschlange verschoben werden.

Achtet man jedoch nur auf die durchschnittliche Bearbeitungszeit, lässt sich eine noch bessere Bearbeitungsmethode für Warteschlangen des Typs `G/G/1` finden:

`by_duration_resumable`

Es wird angenommen, dass jeder Auftrag niedergelegt, ein kürzerer Auftrag aufgenommen werden, und nach Fertigstellung der vorherige Auftrag wieder aufgenommen werden kann. So kann immer der kürzeste Auftrag bearbeitet werden, und nicht nur wenn ein neuer Auftrag ausgewählt wird. Dies geht mit einer deutlichen Verbesserung der durchschnittlichen Wartezeit einher.

Dies ist die optimale Art eine `G/G/1` Warteschlange zu bearbeiten, da zu jedem Zeitpunkt der kürzeste Auftrag bearbeitet wird und die Zeit bis zur Fertigstellung eines Auftrags immer minimal ist. Diese Verbesserung ist auch deutlich in der Simulation zu erkennen.

Beispiele

Hier wird das Programm auf die fünf Beispiele von der Website angewendet:

`fahrradwerkstatt0.txt`

```
Aufträge simuliert in 0.005ms:

╒═══════════════════════════════╤═══════════╤═══════════════╤═══════════════════════╕
│                               │ by_submit │ by_duration   │ by_duration_resumable │
╞═══════════════════════════════╪═══════════╪═══════════════╪═══════════════════════╡
│ Maximale Wartezeit (min)      │  8500.000 │ 8500.000      │ 11528.000             │
├───────────────────────────────┼───────────┼───────────────┼───────────────────────┤
│ Durchschnittliche Wartezeit (min) │  2098.868 │ 2048.456      │ 1893.860              │
╘═══════════════════════════════╧═══════════╧═══════════════╧═══════════════════════╛
```

`fahrradwerkstatt1.txt`

```
Aufträge simuliert in 0.016ms:

╒═══════════════════════════════╤═══════════╤═══════════════╤═══════════════════════╕
│                               │ by_submit │ by_duration   │ by_duration_resumable │
╞═══════════════════════════════╪═══════════╪═══════════════╪═══════════════════════╡
│ Maximale Wartezeit (min)      │  2638.000 │ 2927.000      │ 4367.000              │
├───────────────────────────────┼───────────┼───────────────┼───────────────────────┤
│ Durchschnittliche Wartezeit (min) │   484.820 │ 425.800       │ 356.349               │
╘═══════════════════════════════╧═══════════╧═══════════════╧═══════════════════════╛
```

`fahrradwerkstatt2.txt`

```
Aufträge simuliert in 0.015ms:

╒═══════════════════════════════╤═══════════╤═══════════════╤═══════════════════════╕
│                               │ by_submit │ by_duration   │ by_duration_resumable │
╞═══════════════════════════════╪═══════════╪═══════════════╪═══════════════════════╡
│ Maximale Wartezeit (min)      │  9755.000 │ 10295.000     │ 10816.000             │
├───────────────────────────────┼───────────┼───────────────┼───────────────────────┤
│ Durchschnittliche Wartezeit (min) │  1660.387 │ 1433.258      │ 1129.478              │
```

`fahrradwerkstatt3.txt`

```
Aufträge simuliert in 0.016ms:

╒═══════════════════════════════╤═══════════╤═══════════╤═══════════════════╕
│                               │ by_submit │ by_duration │ by_duration_resumable │
╞═══════════════════════════════╪═══════════╪═══════════╪═══════════════════╡
│ Maximale Wartezeit (min)      │  5319.000 │  5831.000 │          5831.000 │
├───────────────────────────────┼───────────┼───────────┼───────────────────┤
│ Durchschnittliche Wartezeit (min) │  1099.882 │  1075.273 │          1004.564 │
╘═══════════════════════════════╧═══════════╧═══════════╧═══════════════════╛
```

`fahrradwerkstatt4.txt`

```
Aufträge simuliert in 0.004ms:

╒═══════════════════════════════╤═══════════╤═══════════╤═══════════════════╕
│                               │ by_submit │ by_duration │ by_duration_resumable │
╞═══════════════════════════════╪═══════════╪═══════════╪═══════════════════╡
│ Maximale Wartezeit (min)      │ 11803.000 │ 18885.000 │         18885.000 │
├───────────────────────────────┼───────────┼───────────┼───────────────────┤
│ Durchschnittliche Wartezeit (min) │  3284.300 │  2896.111 │          2785.222 │
╘═══════════════════════════════╧═══════════╧═══════════╧═══════════════════╛
```

Quellcode

`program.py`

```python
from queue import PriorityQueue
from time import time
from os import path
from typing import List, Tuple
from pandas import DataFrame


def r_path(path_: str) -> str:
    return path.join(
        path.dirname(path.abspath(__file__)),
        path_
    )
```

```python
def by_submit(queue: List[Tuple[int, int]]) -> Tuple[int, int]:
    queue.sort(key=lambda x: x[0])
    done = 0

    max_wait = 0

    avg_wait = 0
    n_avg_wait = 0

    for submit, duration in queue:
        wait = max(0, done - submit) + duration
        done = max(done, submit)+duration

        max_wait = max(max_wait, wait)
        avg_wait = (n_avg_wait * avg_wait + wait) / (n_avg_wait + 1)
        n_avg_wait += 1

    return max_wait, avg_wait


def by_duration(queue: List[Tuple[int, int]]) -> Tuple[int, int]:
    p = PriorityQueue()
    done = 0

    max_wait = 0

    avg_wait = 0
    n_avg_wait = 0

    for i, (submit, duration) in enumerate(queue):
        done = max(done, submit)
        p.put((duration, submit))
        while not p.empty() and (i + 1 == len(queue) or done < queue[i + 1][0]):
            duration, submit = p.get(block=False, timeout=0)
            wait = max(0, done - submit) + duration
            done = done+duration

            max_wait = max(max_wait, wait)
            avg_wait = (n_avg_wait * avg_wait + wait) / (n_avg_wait + 1)
            n_avg_wait += 1

    return max_wait, avg_wait


def by_duration_resumable(queue: List[Tuple[int, int]]) -> Tuple[int, int]:
    p = PriorityQueue()

    max_wait = 0

    avg_wait = 0
    n_avg_wait = 0

    done = 0
```

```
    for i, (submit, duration) in enumerate(queue):
        done = max(done, submit)
        p.put((duration, submit))
        next_submit = queue[i + 1][0] if i + 1 < len(queue) else 9999999999999999
        while not p.empty() and done < next_submit:
            duration, submit = p.get(block=False, timeout=0)
            if duration > next_submit - done:
                p.put((duration - (next_submit - done), submit))
                done = next_submit
                break
            done = done+duration
            wait = done - submit

            max_wait = max(max_wait, wait)
            avg_wait = (n_avg_wait * avg_wait + wait) / (n_avg_wait + 1)
            n_avg_wait += 1

    return max_wait, avg_wait


# hauptprogramm
def main():
    print()
    bsp_nr = int(input('Bitte die Nummer des Beispiels eingeben [0; 4]: '))

    print()

    start_time = time()  # variable für die zeitmessung

    # laden des graphen mit umgedrehten pfeilen um memoization zu erleichtern
    queue: List[Tuple[int, int]] = []
    with open(r_path(f'beispieldaten/fahrradwerkstatt{bsp_nr}.txt'), 'r') as f:
        for a, b in map(lambda x: tuple(map(int, x.split())),
                        filter(lambda x: x != '', f.read().split('\n'))):
            queue.append((a, b))

    queue.sort()

    processors = [by_submit, by_duration, by_duration_resumable]

    data = {processor.__name__: processor(queue) for processor in processors}

    df = DataFrame(data, index=['Maximale Wartezeit (min)',
                   'Durchschnittliche Wartezeit (min)'])

    print(f'Aufträge simuliert in {format(time()-start_time, ".3f")}ms:')
    print()
    print(df.to_markdown(tablefmt='fancy_grid', floatfmt='.3f'))


# programmschleife
print('Fahrradwerkstatt')
print('(Drücke ^C um das Programm zu beenden)')

while True:
    try:
```

```
        main()
    except Exception as e:
        print(e)
    except KeyboardInterrupt:
        print('\n')
        exit()
```