41. Bundeswettbewerb Informatik

Dokumentation zur Aufgabe 4: „Fahrradwerkstatt“

Teamname: SRZinfo4/1
Team-Id: 00495
Bearbeiter: Karl Jahn

Dresden, den 21. November 2022

Schülerrechenzentrum der TU-Dresden

Inhaltsverzeichnis

1 Lösungsidee

Das „Simulieren“ ist relativ trivial. Marc beginnt seinen Arbeitstag um neun Uhr. Es ist notwendig zwischen den Aufgaben zu wissen, wie spät es ist - der Beginn der Simulation ist sozusagen auch „ dazwischen“, also ist es zum Beginn um neun. Aufgaben/Aufträge kann er natürlich auch eher bekommen. In der ersten Simulation bearbeitet er die Aufträge in der Reihenfolge, in der sie ihn erreichen. Hierbei wird für jede Aufgabe die Wartezeit folgendermaßen berechnet:

$$\Delta Wartezeit = |\, t_{Auftragseingang} - t_{jetzt} \,|$$

$$t_{jetzt} = \Delta Geschlossen * \frac{((t_{Start} - t_{Öffnung}) \bmod t_{Öffnung}) + Arbeitszeit}{t_{Öffnung}} + t_{Start}$$

$$t_{jetzt} = 16\,\mathrm{h} * \frac{((t_{Start} - 9\,\mathrm{h}) \bmod 9\,\mathrm{h}) + Arbeitszeit}{9\,\mathrm{h}} + t_{Start}$$

Hierbei ist die Wartezeit die Differenz $\Delta Wartezeit$ des Zeitpunktes des Auftragseingang und der Zeit des Auftragsabschluss (t_{jetzt}).

Auch beim zweiten Verfahren von Marc werden nicht alle Kunden, welche in seine Werkstatt gehen zufrieden sein. Zum einen könnte Marc ja an einen größeren Auftrag sitzen und jemand braucht nur eine Lampe gewechselt, zum anderen kann es auch gut möglich sein, dass Marc sich „verkleckert“ und nur kleine Aufträge hintereinander erfüllt, die großen aber immer weiter in die Zukunft verschoben - und so vielleicht nie fertig werden. Zudem könnten „Überstunden“ dabei helfen um die vierzehn Stunden eher fertig zu werden, was den Kunden erfreut und auch ihn selber Zeit sparen lässt, da er sich nicht wieder „ einarbeiten“ muss.

2 Umsetzung

Ich habe meine Idee in C++ (MSVC++ 20) umgesetzt. Es nutzt die Standardbibliotheken `iomanip` (zur Manipulation von Ausgabestreams), `iostream` (zur Verwendung des Consolenausgabestreams), `fstream` (zur Nutzung von Filestreams), `string` (zur Nutzung von `std::string` und string-Funktionen), `vector` (Dynamische „C++-Arrays“) und `tuple` (zur Verwendung von Tupeln). Die Executable wird über die Konsole mit `"[Name der *.exe] [Pfad/Name der Datei]"` ausgeführt. Hierbei wird die Executable ausgeführt und alle drei Simulationen mit den Daten der angegebenen Datei ausgeführt. Getestet wurde das Programm unter Windows 10, dessen Ergebnisse sind in „3 Beispiele“ zu sehen.

Die Funktion `simulation1` simuliert Marcs ersten Ansatz, `simulation2` seine Optimierung und `simulation3` meine Anpassungen. Alle sind vollständig in „4 Quellcode“ abgebildet.

3 Beispiele

fahrradwerkstatt0.txt

```
PS D:\Projekte\Programmierung\BWINF 2022\Aufgabe4\x64\Debug> .\Aufgabe4.exe D:\BWINF\a40.txt
Input file: "D:\BWINF\a40.txt"
Total tasks: 115

Start first simulation (sorted by incoming Tasks) . . . done!
Maximum waiting time: 8852.0 min, average waiting time: 2202.6 min

Start second simulation (sorted by task length) . . . done!
Maximum waiting time: 8100.0 min, average waiting time: 1823.9 min

Start third simulation (own ideas) . . . done!
Maximum waiting time: 8100.0 min, average waiting time: 1822.6 min
```

fahrradwerkstatt1.txt

```
PS D:\Projekte\Programmierung\BWINF 2022\Aufgabe4\x64\Debug> .\Aufgabe4.exe D:\BWINF\a41.txt
Input file: "D:\BWINF\a41.txt"
Total tasks: 767

Start first simulation (sorted by incoming Tasks) . . . done!
Maximum waiting time: 2718.0 min, average waiting time: 509.2 min

Start second simulation (sorted by task length) . . . done!
Maximum waiting time: 2039.0 min, average waiting time: 256.9 min

Start third simulation (own ideas) . . . done!
Maximum waiting time: 2023.0 min, average waiting time: 254.8 min
```

fahrradwerkstatt2.txt

```
PS D:\Projekte\Programmierung\BWINF 2022\Aufgabe4\x64\Debug> .\Aufgabe4.exe D:\BWINF\a42.txt
Input file: "D:\BWINF\a42.txt"
Total tasks: 187

Start first simulation (sorted by incoming Tasks) . . . done!
Maximum waiting time: 10107.0 min, average waiting time: 1745.3 min

Start second simulation (sorted by task length) . . . done!
Maximum waiting time: 7066.0 min, average waiting time: 1155.7 min

Start third simulation (own ideas) . . . done!
Maximum waiting time: 7066.0 min, average waiting time: 1154.4 min
```

fahrradwerkstatt3.txt

```
PS D:\Projekte\Programmierung\BWINF 2022\Aufgabe4\x64\Debug> .\Aufgabe4.exe D:\BWINF\a43.txt
Input file: "D:\BWINF\a43.txt"
Total tasks: 221

Start first simulation (sorted by incoming Tasks) . . . done!
Maximum waiting time: 5479.0 min, average waiting time: 1151.4 min

Start second simulation (sorted by task length) . . . done!
Maximum waiting time: 3789.0 min, average waiting time: 984.3 min

Start third simulation (own ideas) . . . done!
Maximum waiting time: 3789.0 min, average waiting time: 982.3 min
```

fahrradwerkstatt4.txt

```
PS D:\Projekte\Programmierung\BWINF 2022\Aufgabe4\x64\Debug> .\Aufgabe4.exe D:\BWINF\a44.txt
Input file: "D:\BWINF\a44.txt"
Total tasks: 91

Start first simulation (sorted by incoming Tasks) . . . done!
Maximum waiting time: 12395.0 min, average waiting time: 3461.6 min

Start second simulation (sorted by task length) . . . done!
Maximum waiting time: 6090.0 min, average waiting time: 2181.7 min

Start third simulation (own ideas) . . . done!
Maximum waiting time: 6074.0 min, average waiting time: 2180.7 min
```

4 Quellcode

Codebeispiel (1) : Funktion zur Berechnung des „Hintereinanderabarbeitens“

```
#define HOUR_TO_MIN(hour)(60 * (hour))

// Die erste Simulation
auto simulation0(std::vector<uint32_t> incoming_time, std::vector<uint32_t> work_time) {
    std::vector<uint32_t> waiting_time(incoming_time.size());   /* Hier werden die Wartezeiten der Kunden
     fuer die einzelnen Auftraege gespeichert */

    uint32_t current_time = HOUR_TO_MIN(9);
    for (size_t t = 0; t < incoming_time.size(); ++t) {
        if (current_time < incoming_time[t]) current_time = incoming_time[t];
        current_time += work_time[t] +
            (((current_time - HOUR_TO_MIN(9)) % HOUR_TO_MIN(9) + work_time[t]) / HOUR_TO_MIN(9)) * 16;
        waiting_time[t] = current_time - incoming_time[t];
    }

    return waiting_time;
}
```

Codebeispiel (2) : Die Verbesserung von Marc

```
// zweite Simulation (Marcs Verbesserung)
auto simulation1(std::vector<uint32_t> incoming_time, std::vector<uint32_t> work_time) {
    std::vector<uint32_t> waiting_time(incoming_time.size(), 0);

    uint32_t current_time = HOUR_TO_MIN(9);
    for (size_t t = 0; t < incoming_time.size(); ++t) {

        /* Unterschied zur ersten Funktion: schnellstes (verfuegbares) wird
        genommen, das fuert zu dieser "Vorsortierung" */
        for (size_t t_2 = t; t_2 < incoming_time.size() and incoming_time[t_2] < current_time; ++t_2) {
            if (not waiting_time[t_2] and work_time[t_2] < work_time[t]) t = t_2;
        }

        while (waiting_time[t]) t++;    // Falls alles Auftraege erfuellt sind => "gehe in die Zukunft"

        if (current_time < incoming_time[t]) current_time = incoming_time[t];
        current_time += work_time[t] +
        (((current_time - HOUR_TO_MIN(9)) % HOUR_TO_MIN(9) + work_time[t]) / HOUR_TO_MIN(9)) * 16;
        waiting_time[t] = current_time - incoming_time[t];
    }

    return waiting_time;
}
```

Codebeispiel (3) : Meine eigene Modifikation

```
auto simulation2(std::vector<uint32_t> incoming_time, std::vector<uint32_t> work_time) {
    std::vector<uint32_t> waiting_time(incoming_time.size(), 0);

    uint32_t current_time = HOUR_TO_MIN(9); size_t _30min = 0;
    for (size_t t = 0; t < incoming_time.size(); ++t) {

        if (work_time[t] < 30) _30min++;

        for (size_t t_2 = t; t_2 < incoming_time.size() and incoming_time[t_2] < current_time; ++t_2) {
            if (_30min < 4) {
                if (not waiting_time[t_2] and work_time[t_2] < work_time[t]) t = t_2;
            }
            else {
                if (not waiting_time[t_2] and work_time[t_2] > work_time[t]) t = t_2;
                _30min = 0;
            }
        }

        while (waiting_time[t]) t++;

        if (current_time < incoming_time[t]) current_time = incoming_time[t];
        auto
            wrk_time_more_houres = (((current_time - HOUR_TO_MIN(9)) % HOUR_TO_MIN(9) + work_time[t] - 60)
                / HOUR_TO_MIN(9)) * 16,
            wrk_time = work_time[t] + (((current_time - HOUR_TO_MIN(9)) % HOUR_TO_MIN(9) + work_time[t])
                / HOUR_TO_MIN(9)) * 16;
        if (wrk_time_more_houres < wrk_time) current_time += work_time[t] + wrk_time_more_houres;
        else current_time += wrk_time;

        waiting_time[t] = current_time - incoming_time[t];
    }

    return waiting_time;
}
```