Bouwmarkt BouwNouGouw

Datastructuren Opgave 7, 28 juni 2025[1]

1 Inleiding

Bob, directeur van Bouwmarkt *BouwNouGouw*, onderzoekt hoeveel kassa's zijn nieuwe filiaal moet hebben. Daarom wil hij een simulatie maken van het proces in de winkel. Bob weet dat elke klant de kortste rij kiest; als er meerdere rijen het kortst zijn, daarvan de meest linkse. Verder heeft Bob gemerkt dat elke cassière op zijn of haar eigen snelheid werkt en daarom heeft Bob gemeten hoelang elke cassière over één artikel doet.

2 Invoer

De invoer begint met een regel met drie waarden. De eerste waarde is k, het aantal kassa's. De tweede waarde is n, het aantal klanten. Beide zijn maximaal 2147483647. De derde waarde is de uitvoermodus, `F` of `D`. Hierna volgen k regels met elk één positief getal, namelijk de tijd per item voor elke kassa. Hierop volgen n regels met twee positieve getallen, voor elke klant een regel met de tijd waarop de klant gaat afrekenen, een spatie en het aantal artikelen.

3 Simulatie

Voor deze opdracht moet je het proces in de bouwmarkt simuleren. In elke tijdstap kan het volgende op deze volgorde gebeuren:

- Een klant komt bij de kassa's. Hij sluit aan in de kortste rij, dwz. die waarin het laagste aantal klanten wacht (zonder rekening te houden met de snelheid van kassa's of het aantal producten dat andere klanten kopen). Als er meerdere kortste rijen zijn, kiest hij daarvan de kassa met het laagste nummer. Elke tijdstap komt maximaal één klant. Als de rij leeg is begint de kassa direct.
- Een kassa is klaar. Als deze rij niet leeg is, begint de cassière direct met de volgende klant. Als er meerdere kassa's tegelijk klaar zijn, zijn de kassa's op oplopende volgorde van kassanummer klaar. Kassa's met een lager nummer beginnen ook weer voordat kassa's met hogere nummers klaar zijn.

Als een cassière op tijdstip b begint voor een klant met a artikelen, is hij klaar op tijdstip $b + a * s$, met s de snelheid van de kassa.

4 Uitvoer

Er zijn twee uitvoermodi. In modus `F` (Full) moet de hele simulatie worden weggeschreven naar de uitvoer. Bij de volgende gebeurtenissen wordt het volgende geprint:

- Als een klant in een rij gaat staan, wordt `T: enqueue K` geschreven.

[1] Versie: 18 februari 2025.

- Als een kassa begint, wordt `T: start K`.
- Als een kassa klaar is, wordt `T: finish K` geschreven.

T is hierbij de tijdstap en K is de index van de kassa. Als een klant aansluit bij een lege kassa, begint die kassa dezelfde tijd nog. Je moet dan wel eerst printen dat de klant in de rij komt te staan en daarna dat die kassa begint.

Modus `D` (Duration) schrijft alleen wanneer het klaar is. In beide uitvoermodi (`F` en `D`) staat op de laatste regel het tijdstip van sluiten van de bouwmarkt (één tijdstap nadat de laatste klant klaar is), en het nummer van de kassa die als laatste klaar was in het format `T: close K`. Als er meerdere kassa's als laatste klaar zijn, moet je het hoogste nummer printen.

5 Voorbeeld

Voorbeeld `Bklein`: Twee kassa's en vier klanten.

```
2 4 F
3
2
10 1
2 2
1 5
3 2
```

```
1: enqueue 0
1: start 0
2: enqueue 1
2: start 1
3: enqueue 0
6: finish 1
10: enqueue 1
10: start 1
12: finish 1
16: finish 0
16: start 0
22: finish 0
23: close 0
```

Op tijdstip 1 komt de eerste klant (die als derde in de invoer staat) binnen, sluit aan bij kassa 0 en deze kassa begint meteen. Die kassa is op tijdstip 16 klaar.

Op tijdstip 2 komt de tweede klant bij kassa 1. Deze kassa is 4 tijdstappen later klaar.

Op tijdstip 3 komt de derde klant binnen. Beide rijen zijn dan even lang. Daarom wacht de klant bij kassa 0. Op tijdstap 16 is hij aan de beurt en op tijdstap 22 is hij klaar.

Op tijdstap 10 komt de laatste klant binnen. Hij wordt direct geholpen bij kassa 1 en is op tijdstap 12 klaar.

De bouwmarkt kan sluiten op tijdstap 23, en kassa 0 was de kassa die het laatst klaar was.

6 Algoritmische Aanwijzingen

Het doel van deze opdracht is het uitprogrammeren van de PriorityQueue met behulp van een heap; daarom is het gebruik van de heap in deze opdracht verplicht. Alle events en queue-operaties moeten lopen in $\mathcal{O}(\log(n+k))$. Daarvoor kan je op twee plekken een Min Heap of Priority Queue gebruiken. Je kan hiermee bijhouden welke rij het kortst is (kassa-queue) en wat het volgende event is (event-queue).

Je kunt events gebruiken voor de binnenkomst van een klant en voor het moment waarop een kassa klaar is. Deze kan je in dezelfde Priority Queue zetten. Denk er om dat je events

op hetzelfde tijdstip en van een verschillend type goed moet ordenen. De klanten staan *niet* gesorteerd op tijd van binnenkomst. Lees daarom meteen in het begin alle klanten in, en maak van elke aankomst een event dat je in de eventqueue zet. Begin dan pas met de simulatie.

Om je code net te houden, is het handig om de Priority Queue maar één keer te implementeren. Daarvoor moet je werken met generics, `<T>` in C#. Je kunt zo een klasse maken die met elk type `T` werkt. Voor een Priority Queue wil je wel eisen dat je twee elementen kan vergelijken. Die beperking op het type kun je op de volgende manier schrijven:

```
class MinHeap<T> where T : IComparable<T>
{
    ..
}
```

Je kunt nu in klasse `MinHeap` twee objecten `a` en `b` vergelijken met `a.CompareTo(b) <= 0`, maar die vergelijking moet je wel implementeren in klasse `T`.

Omdat je in een MinHeap niet kunt zoeken op key, zul je ook nodig hebben om in een object op te slaan, wat zijn positie in de MinHeap is. Hiervoor kun je nog een beperking toevoegen op de types waarmee de heap werkt:

```
class MinHeap<T> where T : IComparable<T>, IHeapItem
{
    ..
}

interface IHeapItem
{
    void SetHeapIndex(int index);
}
```

Nu vereis je dat `T` een methode `SetHeapIndex` heeft. Je kan deze methode telkens aanroepen als je het element op een andere plek in de Min Heap zet.