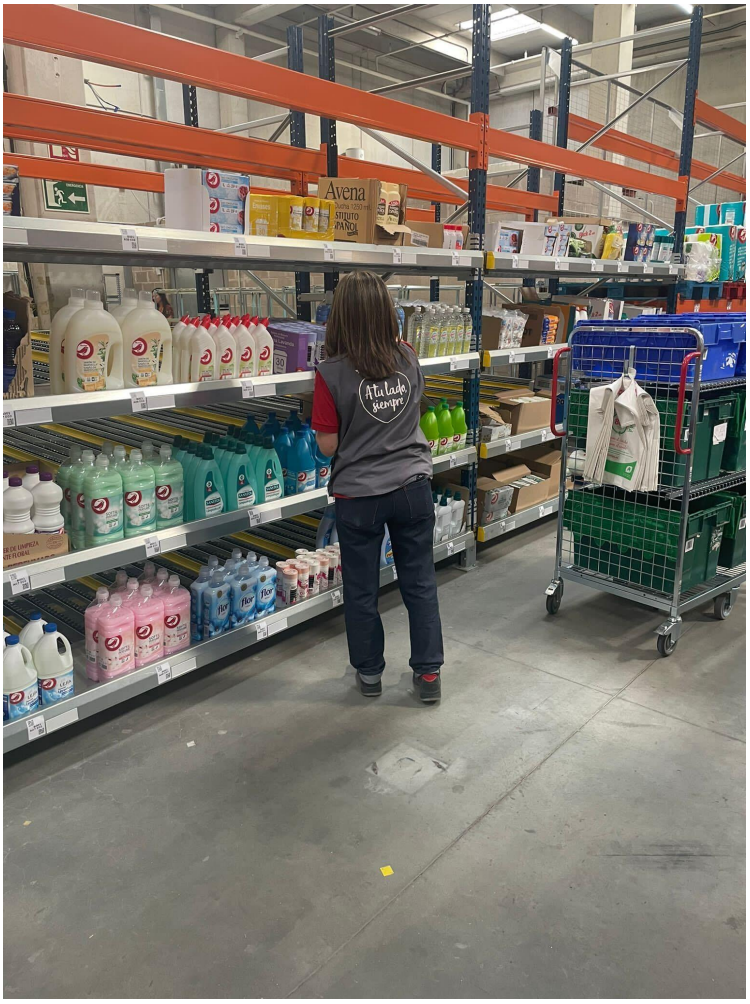
Wstęp



Cześć!

Już tradycyjnie, jak co roku w okolicach marca, mamy do rozwiązania poważny problem. Co za zbieg okoliczności… Spółka, która dostarcza rozwiązanie do obsługi ISFa (opis niżej), od przyszłego miesiąca zmienia cennik. Sam system działa bardzo dobrze, ale problemem jest to, że od 1 kwietnia zaczniemy do naszego biznesu dopłacać (i uwierz nam, nie jest to żart). Żeby tego uniknąć, musisz pomóc nam stworzyć rozwiązanie, które zastąpi istniejący system. Całą specyfikację znajdziesz poniżej. Żeby zdążyć przed zmianą cennika, musimy wdrożyć aplikację w ciągu kilku dni. W mailu znajdziesz dokładny termin, do którego należy odesłać rozwiązanie. Pomożesz nam?

In Store Fulfillment

In Store Fulfillment (ISF) to jeden ze sposobów realizacji zamówień w OSP (Ocado Smart Platform). Pracownicy chodzą po sklepie i zbierają przedmioty, które klienci zamówili online. Następnie skompletowane zamówienia umieszczane są w zamykanych szafkach, z których klienci mogą je samodzielnie odebrać. ISF'em może być zwykły sklep, otwarty także dla innych klientów, lub tak zwany "dark store", który jest dedykowany do realizacji zamówień online.

W każdy dzień roboczy o północy wysyłamy do ISFa listę zamówień do przygotowania na cały dzień. Później, w godzinach pracy sklepu, pracownicy szukają przedmiotów, które kupili klienci, oraz ich zamienników i kompletują w ten sposób zamówienia. Następnie wkładają je do szafek, a klienci mogą je odbierać 24/7.

Zadania

Zadanie 1

Twoi celem jest stworzenie aplikacji, która przygotuje harmonogram kompletowania zamówień dla pracowników ISFa. Dążymy do tego, by skompletować ich jak najwięcej. Każdy “picker” powinien mieć przypisaną listę zamówień do realizacji w dniu roboczym. Zamówienia, których nie uda się dokończyć o czasie, powinny zostać pominięte w wyniku. W tej części zadania Twój algorytm nie powinien polegać na wartości zamówienia.

Zadanie 2 (opcjonalne)

W drugiej iteracji chcielibyśmy zmaksymalizować wartość zrealizowanych zamówień, poprzez uwzględnienie parametru `orderValue` w algorytmie. Zadanie 2 ma na celu modyfikację rozwiązania z pierwszej części tak, aby lepiej spełniało nasze potrzeby biznesowe.

Dane wejściowe

Do uruchomienia aplikacji wymagane będą dwa pliki. Oba w formacie JSON, tak jak w przykładach. Uruchamiając aplikację musimy mieć możliwość określenia bezwzględnych ścieżek do obu z tych plików jako argumenty w wierszu poleceń. Pierwszy to konfiguracja sklepu (`store.json`):

`/home/tester/files/store.json`

```
{
 "pickers": [
   "picker-1",
   "picker-2"
 ],
 "pickingStartTime": "06:00",
 "pickingEndTime": "08:30"
}
```

Opis atrybutów:

- `pickers (List<String>)` - Identyfikatory pracowników, którzy kompletują zamówienia. Będzie ich nie więcej niż 50;
- `pickingStartTime (LocalTime)` - czas rozpoczęcia kompletowania zamówień w sklepie;
- `pickingEndTime (LocalTime)` - czas zakończenia kompletowania zamówień w sklepie

Drugi to lista zamówień na dany dzień (`orders.json`):

/home/tester/files/orders.json

```
[
 {
   "orderId": "order-1",
   "orderValue": "52.40",
   "pickingTime": "PT12M",
   "completeBy": "15:30"
 },
 {
   "orderId": "order-2",
   "orderValue": "82.40",
   "pickingTime": "PT17M",
   "completeBy": "14:00"
 },
 ...
]
```

Opis atrybutów zamówienia:

- `orderId (String)` - identyfikator zamówienia;
- `orderValue (BigDecimal)` - wartość zamówienia;
- `pickingTime (Duration)` - czas potrzebny na skompletowanie zamówienia;
- `completeBy (LocalTime)` - najpóźniejsza godzina, o której zamówienie musi być skompletowane i gotowe do odbioru.

Uwaga: Zamówień nie będzie nigdy więcej niż 5000; wszystkie atrybuty związane z czasem są zgodne z ISO 8601, dokładne co do minuty.

Uruchomienie aplikacji i spodziewany wynik

Aplikację będziemy uruchamiać używając Javy 17:

```
java -jar /home/…/app.jar /home/…/store.json /home/…/orders.json
```

Wynik działania powinien zostać wyświetlony na standardowym wyjściu.

Format wyniku:

```
picker-id order-id picking-start-time
picker-id order-id picking-start-time
...
```

`picking-start-time` to godzina, kiedy pracownik ma zacząć kompletować dane zamówienie według wyliczonego harmonogramu. Kolejność zadań w wyniku nie ma znaczenia.

Uwaga: na wynik działania programu będziemy czekać maksymalnie 20 sekund.

Samodzielna, wstępna weryfikacja rozwiązania

Aby zweryfikować, czy dostarczone rozwiązanie uruchomi się poprawnie w docelowym środowisku, przygotowaliśmy testowe pliki z konfiguracją ISFa i listą zamówień. Możesz ich użyć do przetestowania swojej aplikacji, czy wszystko działa jak należy oraz zgodności wyniku ze zdefiniowanym formatem. Załączone dane testowe są bardzo proste i pozwolą na zweryfikowanie algorytmu w ograniczonym zakresie. Pamiętaj że system docelowo będzie działał w większej skali – więcej pracowników i zamówień.

Wszystkie pliki znajdziesz w folderze `self-test-data/`. Pogrupowaliśmy je w foldery, a każdy z nich jest opisany poniżej. Każdy folder zawiera pliki `orders.json` i `store.json` oraz plik `output.txt` z oczekiwanym wynikiem. Pamiętaj, że kolejność przypisania pickerów i wypisania wykonywanych przez nich zadań nie wpływają na rozwiązanie.

Ważne: Jeśli Twój algorytm nie daje w pełni zgodnych wyników, to nie musi oznaczać, że jest nieprawidłowy. Zdecydowanie możesz w takim przypadku wysłać nam swoje rozwiązanie.

Podstawowe przypadki testowe

- `complete-by:` sprawdzenie, czy zamówienia są kompletowane przed wymaganym czasem.
- `isf-end-time:` sprawdzenie, czy zamówienia są kompletowane przed wymaganym czasem.
- `optimize-order-count:` sprawdzenie, czy algorytm wybierze zrealizowanie dwóch zamówień zamiast jednego.
- `logic-bomb:` maksymalny rozmiar problemu, kóry Twój algorytm powinien obsłużyć.

Rozszerzone przypadki dla zadania pierwszego

- `any-order-length-is-ok:` wizualizacja sytuacji, w której picker może zrobić jedno z dwóch zadań i nie ma znaczenia, które wybierze.
- `advanced-optimize-order-count:` rozszerzenie `optimize-order-count`.
- `advanced-allocation:` test, dla którego najprostesze algorytmy nie znajdują optymalnego rozwiązania, ale da się je otrzymać w krótkim czasie.

Rozszerzone przypadki dla zadania drugiego

- `optimize-order-value:` test weryfikujący, czy algorytm wybierze zamówienie o wyższej wartości.
- `advanced-optimize-order-value:` rozszerzenie poprzedniego z wykorzystaniem dwóch pickerów.

Nasze oczekiwania

- W odpowiedzi na maila z zadaniem spodziewamy się archiwum .zip lub .tar.gz z:
    - kodem źródłowym aplikacji
    - możliwym do uruchomienia plikiem .jar.
- Aplikacja powinna być napisana w Javie 17
- Można korzystać z dowolnych bibliotek, należy jednak wiedzieć, co robią.
    - Uwaga! Prosimy aby jar zawierał wszystkie wymagane zależności aplikacji.
- To, na co zwracamy uwagę: poprawność wyników, czytelność kodu, jakość projektu (chcielibyśmy aby kod, a przynajmniej jego kluczowe fragmenty, były pokryte testami).
- Możesz uzyskać dodatkowe punkty za użycie narzędzia do budowania projektu oraz stworzenie pliku README wraz z kluczowymi informacjami.