TrainFixer: Validér og reparér toge med linkede lister

Det her er den første afleveringsopgaver i Datastrukturer og algoritmer efterår 2025.

Formalia

Deadline er **torsdag den 18. september kl. 20:00**.

Du kan løse opgaven i grupper på op til 3, eller alene.

Du må selv vælge programmeringssprog, fx Java eller JavaScript.

Men sørg for at vælge noget, hvor opgavens krav er realistiske.

På Itslearning under **Ressourcer > Afleveringer > Aflevering 1: Trainfixer** indsender I et link til et GitHub-repository.

I GitHub-repository'et pusher I en løsning til nedenstående opgave, samt en README.md, hvor der står:

- Gruppemedlemmer (enten navne eller EK-id'er)
- En kort instruktion i hvordan man kører jeres tests
- Tekstafsnit til delopgave 4 om køretidskompleksitet

I kan gøre repository'et privat og tilføje min konto som observatør: **sshine**

Oversigt

Opgaven kan inddeles i følgende trin:

1. En generisk linket liste-datastruktur
2. Vogne og toge som linkede lister af vogne
3. En funktion der kan validere om en vogn er korrekt
4. En funktion der kan fikse et tog så det bliver korrekt

1. Definér en linket liste-datastruktur

Man skal kunne tilføje vilkårlige slags ting til listen.

Den må ikke være hardcoded til kun at virke med togvogne.

I Java svarer det til at ens datastruktur tager en type-parameter, fx `T`'et i `MyLinkedlist<T>`.

I JavaScript betyder det bare, man kan gemme en hvilken som helst data/payload i den enkelte node.

2. Vogne og toge som linkede lister af vogne

Lav en datastruktur som repræsenterer et tog.

Et tog er en linket liste af vogne.

Der findes følgende vogntyper:

- Lokomotiver
- Passagervogne:
  - Siddevogne
  - Sengevogne
  - Spisevogne
- Godsvogne

3. En funktion der kan validere om en vogn er korrekt

Lav en funktion som validerer om et tog er gyldigt ud fra følgende regler:

- Lokomotiver:
  - For tog med 10 eller færre vogne er den eneste gyldige placering som forreste vogn.
  - For tog med flere end 10 vogne SKAL der være et lokomotiv både forrest og bagest.
- Passagervogne: Skal være foran alle godsvogne. Der findes tre slags passagervogne:
  - Siddevogne: Ingen særlige regler.
  - Sengevogne: Hvis der er mere end én sengevogn på toget, skal de ligge i forlængelse af hinanden.
  - Spisevogne: Hvis der er en spisevogn, skal det være muligt at gå til den fra alle siddevogne uden at krydse en sengevogn.
- Godsvogne: Skal være bag alle passagervogne.

Det er tilstrækkeligt hvis funktionen returnerer sand/falsk.

Til hver vogntype skal du lave afdækkende positive og negative tests. For eksempel, for at lave positive tests af reglen med sengevogne, er det nødvendigt at have mindst tre tests: En hvor toget har nul sengevogne, en hvor toget har én sengevogn, og en hvor toget har to sengevogne. Og for at have negative tests a reglen med sengevogne, er det nødvendigt at have mindst to tests: En hvor der er to vogne som ikke er forbundet, og en hvor der er tre vogne, hvor to er forbundet og en ikke er.

- Du må gerne bruge AI til at generere tests.
- Sørg for at læse dine tests igennem.
- Sørg for at der ikke er tests som gør afprøver det samme.

4. En funktion der kan fikse et tog så det bliver korrekt

Lav en funktion som fikser togvogne med de ovenstående regler. Desuden, så sørg for at:

- Hvis der er for mange lokomotiver, så fjern dem. Hvis der er nok, så sørg for de står rigtigt.
- Hvis der ikke er nok spisevogne, så tilføj dem. Hvis der er nok, så sørg for de står rigtigt.

Argumentér for hvilken Big-O (asymptotisk) køretid funktionen har.

Tæl desuden hvor mange gange funktionen løber hele toget igennem.

Overvej hvilke fordele og ulemper der er ved at den har ét gennemløb vs. flere gennemløb.