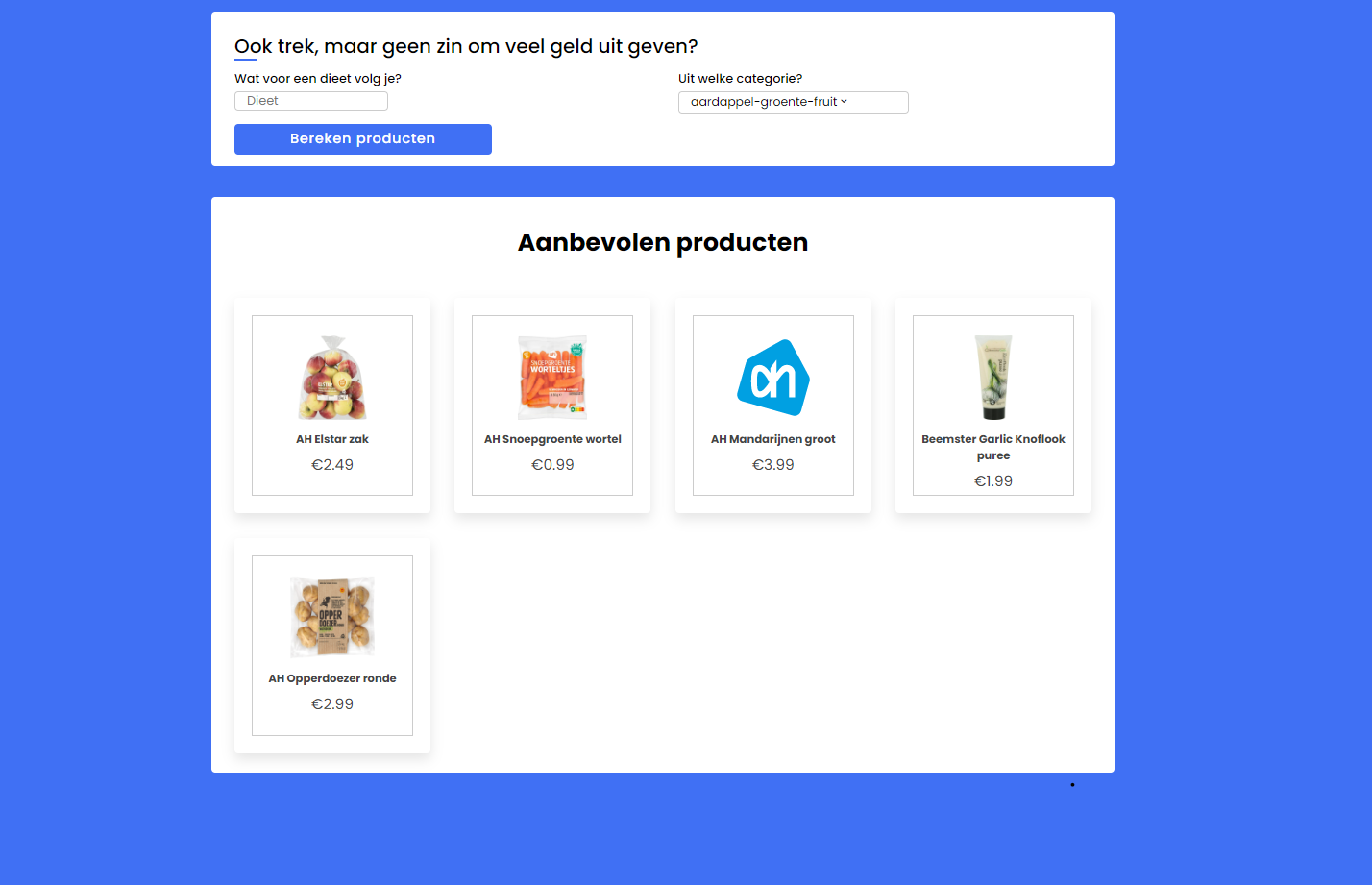
Project samenvatting: De betere boodschap



Daan van Dijk

V1B

IPASS

# Probleembeschrijving

Aangezien de primaire levensbehoeften steeds duurder worden lijkt het mij een goed idee om hier een algoritme op te verzinnen. In het specifiek gaat het over boodschappen van een warenhuis. Echter zijn er al genoeg bronnen waar je je goedkoopste boodschappen kan uitkiezen en is er ook geen algoritme voor nodig, aangezien je alleen de prijs vergelijkt. Ik als sportliefhebber merk dat voeding steeds belangrijker wordt door de mens. Veel mensen hebben een speciaal voedingsschema om bijvoorbeeld af te vallen, spieren te kweken of gewoon aan te komen. Deze voedingspatronen worden natuurlijk beïnvloed door de voedingsstoffen die in het voedsel zit. Het ene dieet bevat bijvoorbeeld veel meer eiwitten of koolhydraten dan een ander dieet. Het lijkt mij een goed idee om hier een oplossing op te bedenken in combinatie met de prijs doormiddel van een algoritme. Aangezien het over het algemeen bekend is dat studenten nou eenmaal weinig geld hebben is dit algoritme bedoeld als hulpmiddel, zo kunnen ze straks voor het minste geld het meest praktische voeding kopen wat bij hen past.

# Algoritme

Door van dit algoritme een knapsack-probleem te maken is het noodzakelijk om met meerdere factoren rekening te houden. In dit geval met voedingsstoffen, gewicht en prijs. Er zijn meerdere benaderingen om een knapsack probleem op te lossen, doordat het algoritme geïmplementeerd wordt in python blijven deze benaderingen over: greedy method, dynamic programming en brute force. De benadering dynamisch programmeren is de beste methode om het knapsack probleem op te lossen. Deze methode verdeelt het probleem steeds in sub problemen totdat je sub problemen hebt die je makkelijk kunt oplossen.

Producten van een supermarkt keten worden in een database gezet. De database wordt verder gesorteerd op categorieën en bijvoorbeeld producten die het meeste kans hebben om aanbevolen te worden. Doordat een gebruiker zijn voorkeuren doorgeeft kan met het knapsack probleem hierboven uitgelegd de meest ideale producten worden aangeraden.

Met het ‘normale’ knapsack probleem bereken je de maximale hoeveelheid prijs en het maximale hoeveelheid gewicht. Echter moeten we bij dit probleem gebruik maken van een minimale prijs en een maximale hoeveelheid. Er zijn helaas geen eerdere algoritmes van dit probleem geïmplementeerd in python, maar er is wel een paper die er veel informatie over geeft. Deze paper heb ik gebruikt om mijn algoritme uit te werken in python. Het algoritme staat op de volgende pagina

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Het knapsack probleem werkt als volgt: Er wordt een 2 dimensionale matrix gemaakt van de lengte van de aantal values + 1 en de capaciteit + 1. De rijen staan voor de values en de kolommen staan voor de gewichten. Integendeel tot het normale probleem wordt elk element in de eerste rij gelijk gesteld aan INF. Dit staat voor oneindig. De eerste kolom is zoals bij de normale knapsack gevuld met nullen. Dan worden alle rijen en alle kolommen afgegaan. Als de value niet in het gewicht van de kolom past dan wordt de waarde van de cel erboven gekopieerd. Dan wordt er gekeken of de huidige value plus de waarde in de cel in de rij erboven maar dan w(gewicht van het huidige item) plekken naar links kleiner is dan de value van de cel erboven. Als daar sprake van is dan is dat de huidige cel waarde en dus tot dusver het beste voor de capaciteit. Er wordt zo door gegaan totdat alle cellen gevuld zijn. Als alles af is staat in de rechter onder hoek de laagst mogelijke prijs voor het vullen van de tas.

# Testen

Er is eigenlijk maar een mogelijkheid om te testen of het algoritme naar behoren werkt. Dat is het zelfstandig uitschrijven op papier. Op de andere pagina staan afbeeldingen van matrixen waar ik het algoritme op heb toegepast. De uiteindelijke uitkomst heb ik vergeleken met mijn algoritme in python doormiddel van unittest. Alle tests zijn geslaagd, dus het algoritme werkt.

## Test 1

Afbeelding met tafel

Automatisch gegenereerde beschrijving

## Afbeelding met tafel Automatisch gegenereerde beschrijvingTest 2 en 3