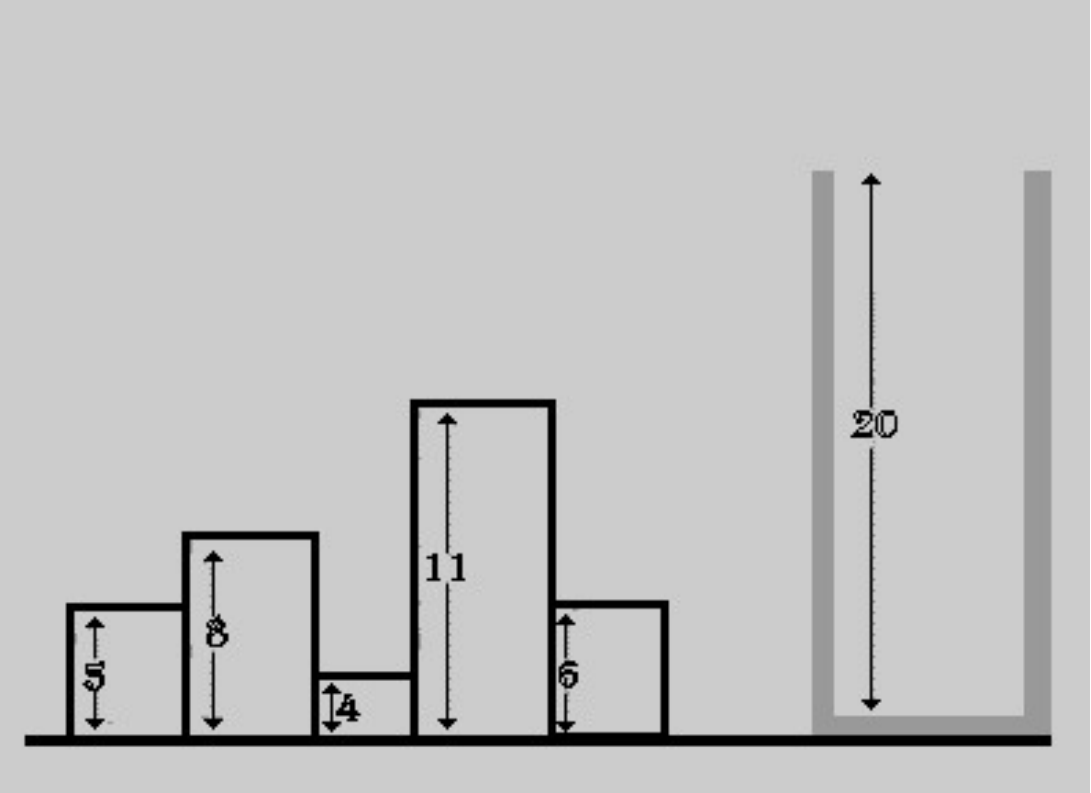
Universiteit van Amsterdam

Programmeertalen

Python

Bas Terwijn

Knapsack



Deadline: Donderdag 23 februari 2023, 23:59

1 Introductie

Bij een knapsack-probleem, zoals weergegeven in figuur 1, krijgen we punten voor elk item dat we inpakken in de knapsack. Elk item mag maar één keer worden ingepakt. De knapsack heeft beperkte resources, bestaande uit 'weight' en 'volume', waardoor niet alle items kunnen worden ingepakt. Het totaal aan resources van de ingepakte items mag de resources van de knapsack niet overschrijden. Welke items in figuur 1 zou jij inpakken om een zo hoog mogelijk puntentotaal in de knapsack te krijgen?

Figuur 1: visualisatie van het knapsack_small.csv bestand

In deze opdracht zijn er drie verschillende CSV-bestanden aangeleverd die elk een Knapsack-probleem representeren die jullie gaan oplossen:

- knapsack_small.csv, 4 items
- knapsack_medium.csv, 18 items
- knapsack_large.csv, 1000 items

Deze CSV bestanden zul je tijdens de opdracht inlezen en hebben een duidelijk formaat. Kijk als voorbeeld naar het small.csv bestand die ook in Figuur 1 is geïllustreerd.

2 Het opstellen van het probleem: Object-Oriented Design

Maak voordat je gaat programmeren eerst een ontwerp wat een natuurlijke opdeling van het probleem zou moeten geven. Maak een UML class diagram met minimaal 3 verschillende classes. Gebruik hierbij dunder methods om het gebruik van deze operatoren/functies mogelijk te maken:

- $+=$
- $-=$
- $<$
- len()

Zorg dat je ontwerp een duidelijk idee geeft van de structuur van je code die je gaat schrijven. Vergelijk je ontwerp met anderen.

2.1 UML class diagram

Houd het simpel, we gebruiken hier UML slechts op een informele manier. Een formele UML specificatie met veel details helpt niet voor ons doel hier.

We willen graag dat jullie zelf vooraf over de structuur van je code nadenken en dat je deze structuur daarna makkelijker met elkaar kan bespreken. Het diagram is een hulpmiddel, dus als je later besluit om een andere structuur te gebruiken is dat prima, pas dan je diagram aan. Gebruik pen en papier of bijvoorbeeld één van deze tools:

- Umletino
  - instructie: https://youtu.be/3UHZedDtr28
- Dia Diagram Editor (see package manager)
  - instructie: https://youtu.be/f-IeQbc2o5k

3 Het probleem oplossen: Solvers

We maken verschillende solvers om de knapsack-problemen op te lossen. Een solver probeert verschillende manieren van inpakken en onthoudt daarvan de beste. Elk type solver moet aan onderstaande solve() functie mee kunnen worden gegeven, een voorbeeld van polymorfisme. Deze solve() functie is gedefinieerd in het bestand “knapsack.py”.

```
def solve(solver, knapsack_file, solution_file):
    """ Uses 'solver' to solve the knapsack problem in
    file 'knapsack_file' and writes the best solution to
    file 'solution_file'.
    """
    knapsack, items = load_knapsack(knapsack_file)
    solver.solve(knapsack, items)
    knapsack = solver.get_best_knapsack()
    knapsack.save(solution_file)

solve(example_solver,
      "knapsack_small.csv",
      "knapsack_small_solution.csv") # for example
```

De oplossing voor het bestand 'knapsack_small_solution.csv' zou er dan zo uit moeten zien:

```
points:60
item1
item3
item4
```

Houd je aan deze uitvoer! Print dus eerst het puntentotaal op dezelfde manier uit, en print daarna per regel de items uit.

3.1 De eerste stap: Random Solver

We gaan nu alle Solvers in subsecties kort toelichten, beginnend met deze eenvoudige solver. Schrijf hiervoor een Random_Solver die random items in de Knapsack inpakt totdat er geen item meer bij kan. Deze Solver hoeft nog niet een optimale inpakking te vinden. De Solver krijgt een argument mee (e.g. 1000), wat aangeeft hoeveel keer deze solver moet proberen om een lege Knapsack in te pakken. Misschien vindt deze solver met genoeg pogingen wel de beste oplossing.

```
solver_random = Solver_Random(1000)
```

3.2 Een stap beter: Optimal Solver

Om een optimale oplossing gemakkelijker te vinden kunnen we depth-first alle mogelijke inpakkingen van items af gaan.

3.2.1 Recursieve solver

Schrijf een Solver_Optimal_Recursive solver die dit recursief doet.

```
solver_optimal_recursive = Solver_Optimal_Recursive()
```

3.2.2 Iteratieve solver met deepcopy

Schrijf een Solver_Optimal_Iterative_Deepcopy solver die dit niet recursief maar iteratief doet. Hierbij mag nog deepcopy[1] gebruikt worden, ondanks dat dit meer geheugen kost en langzamer is.

```
solver_optimal_iterative_deepcopy = Solver_Optimal_Iterative_Deepcopy()
```

3.2.3 Iteratieve solver zonder deepcopy

Schrijf een Solver_Optimal_Iterative() die dit iteratief doet maar nu zonder gebruik van deepcopy.

```
solver_optimal_iterative = Solver_Optimal_Iterative()
```

[1] Klik op deze tekst voor de documentatie

3.3 Mijn programma is te traag: Random Improved

Het knapsack_large.csv probleem is veel te groot om met een optimale solver die alle mogelijkheden afgaat op te lossen. De Random_Solver kan wel snel redelijke oplossingen vinden, maar deze hebben meestal geen hoog puntentotaal. Schrijf een eigen Solver_Random_Improved solver die tot betere oplossingen komt dan de Random_Solver.

```
solver_random_improved = Solver_Random_Improved(5000)
```

Denk daarbij bijvoorbeeld aan deze aanpak:

```
Pak de knapsack random in
Herhaal:
   Maak een random aanpassing: 1 item eruit, andere items erin
   Maak de aanpassing ongedaan als het puntentotaal hierdoor verslechtert
```

maar een andere aanpak is ook toegestaan zolang dit een significant beter resultaat geeft dan de Solver_Random.

4 Tips & Tricks

- Zorg voor een duidelijk UML class diagram wat een goed overzicht van de structuur van je code geeft.
- Zorg voor goed leesbare code met makkelijk te begrijpen namen. Object-Orientatie zou hierbij moeten helpen.
- Maak gebruik van encapsulation (geen directe toegang tot interne class attributes van buiten een class).
- Vaak moet het puntentotaal van een knapsack worden berekend. Het is niet efficiënt om elke keer opnieuw de som van punten van alle items te berekenen. Beter is het om de knapsack zelf een eigen “points” attribute te geven. Dit zal leiden tot een invariantie.
- Zorg dat alle invarianties goed worden afgeschermd door gebruik van encapsulation en een “_” prefix bij namen van attributen. Kijk hierbij ook goed terug naar de slides van het hoorcollege!
- Voorkom code duplicatie bijvoorbeeld door het gebruik van inheritance.
- Schrijf documentatie voor classes en functies in Docstring formaat.
- Volg de Style Guide zoals gedefinieerd door Flake8. We zullen hier automatisch op testen!
- Check de rubric op CodeGrade!

5 Inleveren

Lever de volgende bestanden in:

knapsack.pdf	UML class diagram voor het Knapsack probleem
knapsack.py	Gegeven “knapsack.py” bestand aangevuld met classes voor de representatie van het Knapsack probleem en de solvers