Rucksack-Problem Solver

Das folgende Programm löst das Rucksackproblem unter Verwendung unterschiedlicher Algorithmen. Es werden vier verschiedene Algorithmen angeboten:

- ***First Fit:*** Dabei werden die Gegenstände der Reihe nach versucht in den Rucksack zu packen, jeder Gegenstand der hineinpasst, wird aufgenommen. Dabei wird die Wertigkeit der verschiedenen Gegenstände nicht berücksichtigt und das Ergebnis ist meist nicht sehr gut.
- ***Greedy:*** Bei diesem Algorithmus werden die Gegenstände zunächst der Wertigkeit nach sortiert (hohe Wertigkeit zuerst) und anschließend versucht jeweils der mit der größtmöglichen Wertigkeit in den Rucksack zu packen. Passt ein Gegenstand nicht, wird als nächstes der Gegenstand mit der nächstkleineren Wertigkeit ausprobiert. Hierbei wird wenig Wert auf das Gewicht der unterschiedlichen Gegenstände gelegt.
- ***Weight:*** Zunächst werden alle Gegenstände dem Gewicht nach sortiert (kleinere zuerst) und anschließend probiert, zuerst die kleinen Gegenstände in den Rucksack zu packen. Dadurch wird versucht so viel Gegenstände wie möglich in den Rucksack zu packen und über die Anzahl eine möglichst hohe Wertigkeit zu erzielen.
- ***Backtracking:*** Beim Backtracking werden alle unterschiedlichen Packmöglichkeiten für den Rucksack angeschaut und die Bestmögliche ausgegeben.

Die Anwendung wurde in **Java** implementiert und benötigt daher eine **Java Runtime**. Das Benutzerinterface wurde mittels Konsolenausgabe (Apache Commons CLI) implementiert. Um die Anwendung zu starten muss die **jar** Datei in die Runtime Umgebung geladen werden, hierzu muss das folgende Kommando in der Konsole (gilt für Windows 10) ausgeführt werden:

```
java -jar RucksackProblem.jar
```

Als erstes wird dann eine Hilfe mit den unterschiedlichen Funktionen der Anwendung dargestellt.

```
Rucksack problem solver started...
usage: Rucksack problem solver:
 -a,--add            Add a new item to the possible items.
 -c,--clear          Removes all defined items.
 -g,--get-weight     Get the rucksack weight.
 -h,--help           Shows the help page.
 -l,--list           List all items.
 -q,--quit           Quit the program.
 -r,--remove         Remove a defined item.
 -s,--solver <arg>   Start the selected solver [f (first fit)| g (greedy)
                     | b (backtracking) | w (weight)].
 -w,--set-weight     Set the rucksack weight.
```

Um eine möglichst einfache Bedienung von Anfang an zu gewährleisten wurde die Rucksackkapazität bereits auf **15** gestellt und **10** unterschiedliche Gegenstände eingefügt. Die beiden Eingaben können mit *-g* (get-weight) und *-l* (list items) abgefragt werden.

```
-g
Rucksack weight:15
-l
All used items:
1. Item{name='Item 1', weight=5, value=12}
2. Item{name='Item 2', weight=15, value=6}
3. Item{name='Item 3', weight=3, value=5}
4. Item{name='Item 4', weight=7, value=8}
5. Item{name='Item 5', weight=4, value=1}
6. Item{name='Item 6', weight=8, value=6}
7. Item{name='Item 7', weight=2, value=3}
8. Item{name='Item 8', weight=4, value=7}
9. Item{name='Item 9', weight=3, value=5}
10. Item{name='Item 10', weight=8, value=20}
Sum weight:59 Sum value:73
```

Aufgrund des Gesamtgewichts (59) aller Gegenstände ist ersichtlich, dass nicht alle Gegenstände in den Rucksack gelegt werden können, da dieser eine geringere Kapazität aufweist (15).

Sollten die bestehenden Gegenstände nicht passen, kann mittels *-r* (remove) Befehl ein einzelner Gegenstand aus der Liste entfernt werden.

```
-r
Enter index of item to remove:2
Item was removed. Item{name='Item 3', weight=3, value=5}
-l
All used items:
1. Item{name='Item 1', weight=5, value=12}
2. Item{name='Item 2', weight=15, value=6}
3. Item{name='Item 4', weight=7, value=8}
4. Item{name='Item 5', weight=4, value=1}
5. Item{name='Item 6', weight=8, value=6}
6. Item{name='Item 7', weight=2, value=3}
7. Item{name='Item 8', weight=4, value=7}
8. Item{name='Item 9', weight=3, value=5}
9. Item{name='Item 10', weight=8, value=20}
Sum weight:56 Sum value:68
```

Eine vollständige Löschung der vorhandenen Gegenstände kann mit *-c* (clear) durchgeführt werden.

```
-c
All items were removed.
-l
All used items:
Items are empty.
```

Ein neuer Gegenstand kann dann mittels *-a* (add) Befehl hinzugefügt werden.

```
-a
Enter new item name:Item 1
Enter new item weight:10
Enter new item value:35
Item was added. Item{name='Item 1', weight=10, value=35}
-l
All used items:
1. Item{name='Item 1', weight=10, value=35}
Sum weight:10 Sum value:35
```

Nachdem nun alle Gegenstände sowie die Rucksackkapazität den Bedürfnissen entsprechend angepasst wurden. Kann mittels *-s* (solver) Kommando und zugehörigem Solvertyp (f, g, w, b) das Rucksackbepacken gestartet werden.

```
-s f
First fit solver selected.
Solver started...
Solution of the solver:
1. Item{name='Item 1', weight=5, value=12}
2. Item{name='Item 3', weight=3, value=5}
3. Item{name='Item 4', weight=7, value=8}
Sum weight:15 Sum value:25
-s b
Backtracking solver selected.
Solver started...
Solution of the solver:
1. Item{name='Item 1', weight=5, value=12}
2. Item{name='Item 7', weight=2, value=3}
3. Item{name='Item 10', weight=8, value=20}
Sum weight:15 Sum value:35
```

In dem obigen Beispiel hat der Backtracking Algorithmus ein besseres Ergebnis als der First Fit Algorithmus geliefert.

Zudem ist es ganz einfach, die bestehende Rucksackkapazität zu verändern und die Solver erneut zu starten.

```
-w
Enter new rucksack weight:30
Rucksack weight was set to:30
-s f
First fit solver selected.
Solver started...
Solution of the solver:
1. Item{name='Item 1', weight=5, value=12}
2. Item{name='Item 2', weight=15, value=6}
3. Item{name='Item 3', weight=3, value=5}
4. Item{name='Item 4', weight=7, value=8}
Sum weight:30 Sum value:31
-s b
Backtracking solver selected.
Solver started...
Solution of the solver:
1. Item{name='Item 1', weight=5, value=12}
2. Item{name='Item 3', weight=3, value=5}
3. Item{name='Item 4', weight=7, value=8}
4. Item{name='Item 8', weight=4, value=7}
5. Item{name='Item 9', weight=3, value=5}
6. Item{name='Item 10', weight=8, value=20}
Sum weight:30 Sum value:57
```