7.1 Rucksackproblem (15 Punkte)

Gegeben Sei ein Rucksack, der ein Maximalgewicht von $w_{\max}$ aushält. Weiterhin gibt es Gegenstände $i = 1, \ldots, n$ mit jeweils einem Gewicht w_i und Wert v_i.

In dieser Aufgabe sollen Sie mittels dynamischer Programmierung das Problem lösen, wie man den Rucksack belädt, so dass der zusammengezählte Wert aller enthaltenen Gegenstände maximal ist. Jeder Gegenstand darf nur einmal verwendet werden.

a) (2P) Beschreiben Sie das Vorgehen, wie man das Problem auf das Lösen von Teilproblemen reduzieren kann.
b) (4P) Beschreiben Sie die Datenstruktur, in der Sie die Werte eintragen, so dass Sie Memoization nutzen können. Erklären Sie dabei, was genau Sie in einem Datenfeld speichern, wie und in welcher Reihenfolge Sie die Werte berechnen.
c) (4P) Laden Sie sich die Datei `Knapsack.java` herunter. Das Grundgerüst und die Ausgabe sind darin bereits implementiert. Ersetzen Sie den Inhalt der Methode `solveProblem` durch eine Implementierung des Problems und setzen Sie die Variable `maxValue` auf den berechneten Wert.
   Achten Sie bei der Implementierung darauf, dass Sie sich an Ihre Beschreibungen der vorhergehenden Teilaufgaben halten. Eventuelle Abweichungen sind zu kommentieren.
d) (5P) Erweitern Sie das Programm so, dass Sie nicht nur den Maximalwert abspeichern, sondern auch welche Gegenstände dazu benötigt werden. Fügen Sie diese der Variable `usedItems` hinzu.

7.2 Floyd-Warshall (4 Punkte)

a) (2P) In der Vorlesung vom 2012-12-03 wurde erklärt, dass man die kürzesten Wege durch Iteration über

   ```
   d[k+1][i][j] = min(d[k][i][j], d[k][i][k+1] + d[k][k+1][j])
   ```

   berechnen kann. Wieso kann man den Index `[k]` entfallen lassen, d.h. immer `d[i][j]` schreiben?
b) (3P) Ferner wurde erklärt, wie man idealerweise zusätzlich zu den Kosten den Weg speichert. Schreiben Sie Pseudocode, der alle Knoten auf dem kürzesten Weg zwischen i nach j in korrekter Reihenfolge ausgibt.