Solution 1

```java
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

import java.util.*;

class Program {
  // O(nc) time | O(nc) space
  public static List<List<Integer>> knapsackProblem(int[][] items, int capacity) {
    int[][] knapsackValues = new int[items.length + 1][capacity + 1];
    for (int i = 1; i < items.length + 1; i++) {
      int currentWeight = items[i - 1][1];
      int currentValue = items[i - 1][0];
      for (int c = 0; c < capacity + 1; c++) {
        if (currentWeight > c) {
          knapsackValues[i][c] = knapsackValues[i - 1][c];
        } else {
          knapsackValues[i][c] =
              Math.max(
                  knapsackValues[i - 1][c],
                  knapsackValues[i - 1][c - currentWeight] + currentValue);
        }
      }
    }
    return getKnapsackItems(knapsackValues, items, knapsackValues[items.length][capacity]);
  }

  public static List<List<Integer>> getKnapsackItems(
      int[][] knapsackValues, int[][] items, int weight) {
    List<List<Integer>> sequence = new ArrayList<List<Integer>>();
    List<Integer> totalWeight = new ArrayList<Integer>();
    totalWeight.add(weight);
    sequence.add(totalWeight);
    sequence.add(new ArrayList<Integer>());
    int i = knapsackValues.length - 1;
    int c = knapsackValues[0].length - 1;
    while (i > 0) {
      if (knapsackValues[i][c] == knapsackValues[i - 1][c]) {
        i--;
      } else {
        sequence.get(1).add(0, i - 1);
        c -= items[i - 1][1];
        i--;
      }
      if (c == 0) {
        break;
      }
    }
    return sequence;
  }
}

```