CS2302 - Data Structures

Spring 2020
Lab 7

Due Monday, May 11, 2020

The knapsack problem is formulated as follows: a thief enters a house and finds items $i_1, ..., i_n$ with values $v_1, ..., v_n$ and weights $w_1, ..., w_n$. The thief has a bag (knapsack) that can carry at most W weight. He also has to pay a debt D with the items he steals. Is it possible for the thief to steal items with value D or more and combined weight W or less?

The knapsack problem can be solved with a backtracking algorithm that is very similar to subset sum. The first few lines of a function to implement it could be as follows:

```
def knapsack(W,D,v,w):
    # W is the remaining knapsack capacity, D is the remaining debt, v and w are lists of the
    # same length where item 0 has value v[0] and weight w[0], item 1 has value v[1] and
    # weight w[1], and so on.
    if W<0: # knapsack capacity exceeded
       return False
    if D<0: # debt paid
       return True
    ...
```

For this lab, your task is to implement three algorithms to solve knapsack.

- A backtracking algorithm based on subsetsum and the code presented above.
- A greedy algorithm that works as follows: sort the items by decreasing value to weight ratio ($vwr = v/w$), then go through the list of items sorted by vwr, adding the item to the knapsack if there is still room for it. If at the end the debt has been paid return True, otherwise return False.
- A randomized algorithm that works as follows: sort the items randomly, then go through the list of items sorted that way, adding the item to the knapsack if there is still room for it. If at the end the debt has been paid return True, otherwise try another random order. Keep trying for a fixed number of times, if none of the orderings results in a solution, return False.

The backtracking algorithm will always find a solution if one exists, but since its running time is $O(2^n)$, it will only work for small instances of the problem. The other two algorithms won't always find a solution if it exists, but can be used for larger problems.

Note: using numpy functions for array manipulation and random number generation will make this assignment a lot easier. The functions np.array, np.random.permutation, and np.argsort may be very useful.

As usual, write a report describing your work. Given the little time available, a demo will not be required, thus it is very important that your report accurately reflects your work. Use the test cases in file *knapsack_data.txt* to test your implementations. We recommend you convert the lists to numpy arrays.