# CS325 Winter 2013: HW 4

Daniel Reichert
Trevor Bramwell

February 22, 2013

## 1

**Problem:** Knapsack without repetitions. Consider the following knapsack problem: The total weight limit $W = 10$ and

| Item | Weight | Value |
|---|---|---|
| 1 | 6 | \$30 |
| 2 | 3 | \$14 |
| 3 | 4 | \$16 |
| 4 | 2 | \$9 |

Solve this problem using the dynamic programming algorithm presented in class. Please show the two dimensional table $L(w, j)$ for $w = 0, 1, \ldots, W$ and $j = 1, 2, 3, 4$.

**Solution:**

| $j \backslash w$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 30 | 30 | 30 | 30 |
| 2 | 0 | 0 | 0 | 0 | 14 | 14 | 14 | 30 | 30 | 30 | 44 |
| 3 | 0 | 0 | 0 | 0 | 14 | 16 | 16 | 30 | 30 | 30 | 46 |
| 4 | 0 | 0 | 0 | 9 | 14 | 16 | 23 | 30 | 30 | 39 | 46 |

## 2

**Problem:** Give a dynamic programming algorithm for solving the following problem.

**Input:** A list of n positive integers $a_1, a_2, \ldots, a_n$ and a number $t$.

**Goal:** Decide if some subset of the $a_i$'s add up to $t$. (You can use each $a_i$ at most once.)

The running time should be $O(nt)$.

Consider the subproblem $L(i, s)$, which returns the answer of "Does a subset of $a_1, \ldots, a_i$ sum up to $s$?". Below are some sub steps that will help you develop your algorithm.

1. What are the two options we have regarding item $i$ toward answering the subproblem $L(i, s)$?
   Either the integer is added to the subset, or it is not added to the subset.

2. For each of the option, how would it change the subproblem? More specifically, what happens to the target sum and what happens to the set of available integers?
   If the integer is added to the subset, then it is removed from the list of positive integers being considered, and the value of the integer is subtracted from $t$. If the integer is not added to the subset, then it is removed from the list of positive integers being considered.

3. Based on the answers to the previous two questions, write a recursive formula that expresses $L(i, s)$ using the solutions to smaller subproblems.
   $L(i, s) = max(L(i-1, s-a_i), L(i-1, s))$

4. Provide pseudo code for the dynamic programming algorithm that builds the solution table $L(i, s)$ and returns the correct answer to the final problem.

```
$L(0, s) = 0 \forall S$ from $0 - t$
$L(i, 0) = 0 \forall i$ from $0 - n$
$for(i) 1 ... n :$
    $for(s) 1 ... t :$
        $L(i, s) = max[L(i-1, s-a_i), L(i-1, s)]$
return $L(i, S)$
```

5. Modify the pseudo code such that it will not only return the correct "yes", "no" answer, but also return the exact subset if the answer is "yes".

```
$L(0, s) = 0 \forall S$ from $0 - t$
$L(i, 0) = 0 \forall i$ from $0 - n$
$for(i) 1...n :$
    $for(s) 1...t :$
        $L(i, s) = max[L(i-1, s-a_i), L(i-1, s)]$
return $L(i, S)$
```

## 3

**6.2 from book:** You are going on a long trip. You start on the road at mile post 0. Along the way there are $n$ hotels, at mile posts $a_1 < a_2 < \cdots < a_n$ , where each $a_i$ is measured from the starting point. The only places you are allowed to stop are at these hotels, but you can choose which of the hotels you stop at. You must stop at the final hotel (at distance $a_n$ ), which is your destination.

You'd ideally like to travel 200 miles a day, but this may not be possible (depending on the spacing of the hotels). If you travel $x$ miles during a day, the penalty for that day is $(200 - x)^2$. You want to plan your trip so as to minimize the total penalty—that is, the sum, over all travel days, of the daily penalties. Give an efficient algorithm that determines the optimal sequence of hotels at which to stop.

**Solution:** $P(i) = min((200 - x_j)^2 + P(j)) \forall 0 \leq j < i$

## 4

**6.8 from book:** Given two strings $x = x_1x_2 \cdots x_n$ and $y = y_1y_2 \cdots y_m$, we wish to find the length of their longest common substring, that is, the largest k for which there are indicies $i$ and $j$ with $x_ix_{i+1} \cdots x_{i+k-1} = y_jy_{j+1} \cdots y_{j+k-1}$. Show how to do this in time $O(mn)$.

```
# The length of S is n and T is m.
LCS(S, T):
    K(i,0) = 0 for 0 to n
    K(0,j) = 0 for 0 to m
    # m is the maximum length substring
    m = 0
    m_i = 0
    for i = 1 to n:
        for j = i to m:
            if i > j:
                return 0
            if $S_i$ = $T_i$:
                # compute sequence
                K(i,j) = K(i-1,j-1)+1
                # update max
                if K(i,j) > m:
                    m = K(i,j)
                    m_i = i
            else
                K(i,j) = 0
    return S[m_i, m_i+m-1]
```