Lindsey Bieda and Joe Frambach
Dynamic Programming Problems
10.03.2011

13. Our goal is now to consider the Knapsack problem, and develop a method for computing the actual items to be taken in $O(L)$ space and $O(nL)$ time.

(a) Consider the following problem. The input is the same as for the knapsack problem, a collection of n items $I_1, \ldots, I_n$ with weights $w_1, \ldots, w_n$, and values $v_1, \ldots, v_n$, and a weight limit L. The output is in two parts. First you want to compute the maximum value of a subset S of the n items that has weight at most L, as well as the weight of this subset. Let us call this value and weight v_a and w_a. Secondly for this subset S you want to compute the weight and value of the items in $\{I_1, \ldots, I_{n/2}\}$ that are in S. Let use call this value and weight v_b and w_b. So your output will be two weights and two values. Give an algorithm for this problem that uses space $O(L)$ and time $O(nL)$.

12. Consider the code for the Knapsack program given in the class notes.

(a) Explain how one can actually find the highest valued subset of objects, subject to the weight constraint, from the Value table computed by this code.

From the table you can find the highest valued subset of objects by examining the table starting at $Value[k, S]$, where k = number of objects considered and S = the maximum weight. If $Value[k-1, S] = Value[k, S]$, then do not add object k to the solution set and let $k = k - 1$. Otherwise, add object k to the solution set and let $S = S - w_k$, where w_k = the weight of object k, then let $k = k - 1$. Repeat until $k = 0$.

(b) Explain how to solve the Knapsack problem using only $O(L)$ memory/space and $O(nL)$ time. You need only find the value and weight of the optimal solution, not the actual collection of objects.

In the iterative solution when building the table, the writes for column k depend only on the values stored in column $k-1$. Therefore, our array only needs to have two columns. At each iteration (when k increases), the columns alternate between being the read column and the write column. This eliminates copy time, although this isn't necessary. The alternating of read-column and write-column is a method taken from the idea of double-buffering.

From 12b, we take the resulting array and then iterate as follows to determine the set of items:
$\ell = 0$, $v_a = 0$, $w_a = 0$, $v_b = 0$, $w_b = 0$

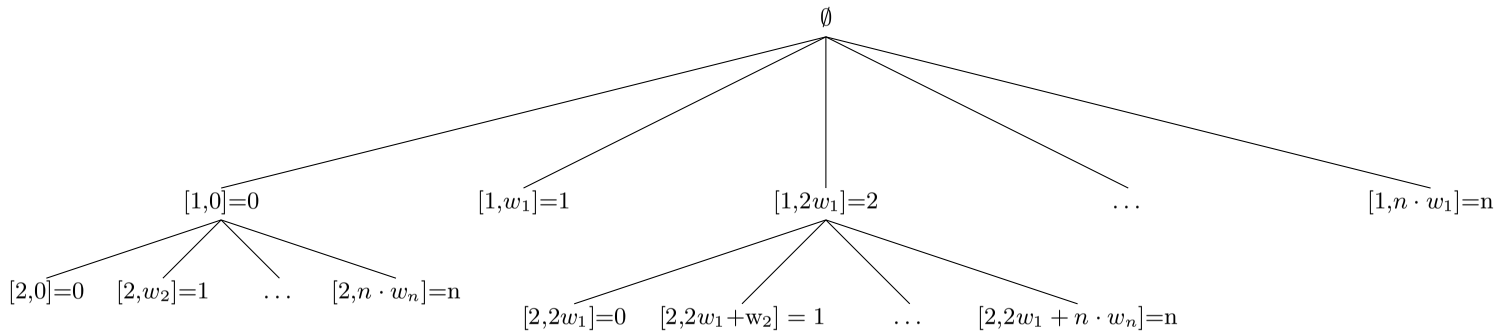
```
For i = 1 to L:
    if ∃ k = Array[i] - ℓ:
        add item k to the solution set S, and ℓ = Array[i]
        v_a = v_a + v_k
        w_a = w_a + w_k
        if k ≤ n/2:
            v_b = v_b + v_k
            w_b = w_b + w_k
```

(b) Explain how to use the algorithm from the previous subproblem to get a divide and conquer algorithm for finding the items in the Knapsack problem a and uses space $O(L)$ and time $O(nL)$.

16. Give an algorithm for the following problem whose running time is polynomial in $n + W$:
Input: positive integers $w_1, \ldots, w_n$, $v_1, \ldots, v_n$ and W.
Output: The maximum possible value of $\sum_{i=1}^{n} x_i v_i$ subject to $\sum_{i=1}^{n} x_i w_i \leq W$ and each x_i is a nonnegative integer.

Pruning Rules:
1) At each level, there may be no more than W configurations. Ignore nodes where the cumulative sum is greater than W.
2) At each level, multiple children are created. Only create W/w_k children, since more would have a weight greater than W.
3) sgs

Begin with root node $A[k = 0, S = 0]$, where k is the *level* of the tree, for 0 to n, and S is the cumulative weight, for 0 to W.
For $k = 0$ *to* $n - 1$:
    For $S = 0$ *to* W:
        For $x = 0$ *to* W/w_k:
            $A[k + 1,\ A[k, S] + x \cdot w_k] = x$.


∅
[1,0]=0
[1,w_1]=1
[1,2w_1]=2
...
[1,$n \cdot w_1$]=n
[2,0]=0
[2,w_2]=1
...
[2,$n \cdot w_n$]=n
[2,2w_1]=0
[2,2w_1+w$_2$] = 1
...
[2,2$w_1 + n \cdot w_n$]=n

17. Give an algorithm for the following problem whose running time is polynomial in $n + L$, where $L = \max\left(\sum_{i=1}^{n} v_i^3, \prod_{i=1}^{n} v_i\right)$.
Input: positive integers $v_1, \ldots, v_n$
Output: A subset S of the integers such that $\sum_{v_i \in S} v_i^3 = \prod_{v_i \in S} v_i$.