Lindsey Bieda and Joe Frambach
Dynamic Programming Problems
9.28.2011

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.

14. Give an algorithm for the following problem whose running time is polynomial in $n + L$.
Input: positive integers $v_1, \ldots, v_n$, with $L = \sum_{i=1}^{n} v_i$.
Output: A solution (if one exists) to $\sum_{i=1}^{n} (-1)^{x_i} v_i = 0$ where each x_i is either 0 or 1.

Pruning rules:
1) $-v_{h+1} - \ldots - v_n < current\ sum < v_{h+1} + \ldots + v_n$, that is if the current sum is too high or too low to be negated by all remaining values, prune that sum.
2) If there are duplicate values at the same level of the tree prune all but one.

Begin with v_1 at the root of tree, because of the symmetric nature.
A[1,v_1] = 0

```
For h = 2 to n:
    For $\ell$ = -L to L:
        A[h,$\ell-$v$_h$] = 1
        A[h,$\ell+$v$_h$] = 0
        L = L - $v_h$
```

If there is a solution A[n,0] will be defined.
Traceback: Begin at $A[\ell = n, s = 0]$.

```
Repeat:
    If $A[\ell, s] = 1$, then add $x_\ell = 1$ to the solution set and go to $A[\ell - 1, s + v_\ell]$
    Else, add $x_\ell = 0$ to the solution set and go to $A[\ell - 1, s - v_\ell]$
```

At the end of the last iteration, the current cell will be $A[1, v_1]$.

15. Give an algorithm for the following problem whose running time is polynomial in $n + logL$:
Input: positive integers $v_1, \dots, v_n$ and L.
Output: A solution (if one exists) to $(\sum_{i=1}^{n} x_i v_i) \bmod n = L \bmod n$ where each x_i is either 0 or 1.
Here x mod y means the remainder when x is divided by y.
Pruning rule(s):
1) If there are duplicate values at the same level of the tree prune all but one.

Begin with v_1 at the root of tree, because of the symmetric nature.
A[1,v_1] = 1

```
For h = 2 to n:
    For $\ell$ = 0 to n-1:
        A[h,$\ell$] = 0
        $A[h, (\ell + \mathrm{v}_h) \bmod n] = 1$
```

If there is a solution $A[n, L \bmod n]$ will be defined.
Traceback: Begin at $A[\ell = n, s = L \bmod n]$.

```
Repeat:
    If $A[\ell, s] = 0$, then add $x_\ell = 0$ to the solution set and go to $A[\ell - 1, s]$
    Else, add $x_\ell = 1$ to the solution set and go to $A[\ell - 1, (s - v_\ell) \bmod n]$
```

At the end of the last iteration, the current cell will be $A[1, v_1]$.