To receive credit for a problem, you must both have sufficient narrative description of your algorithm and a sentence stating what the intended meaning of each array entry is (e.g. $A(i, j)$ is 1 if there is a subset of the first i items with aggregate sum j).

17. (2 points for the first 3 subproblems plus 2 points if you do the last subproblem) We consider several variations on the standard Subset Sum and Knapsack problems. For each problem describe a tree with all of the feasible solutions at the leaves, give a pruning rule for this tree, and finally give the iterative array-based code that naturally results from this pruning rule.

    (a) The problem is defined as follows:
    Input: positive integers $v_1, \ldots, v_n$, with $L = \sum_{i=1}^{n} v_i$.
    Output: Determine where a solution to the equation $\sum_{i=1}^{n} (-1)^{x_i} v_i = 0$ exists, where each variable x_i can be either 0 or 1.
    The running time should be polynomial in $n + L$ where $L = \sum_{i=1}^{n} v_i$.

    (b) The problem is defined as follows:
    Input: positive integers $v_1, \ldots, v_n$ and L.
    Output: Determine where a solution to the equation $(\sum_{i=1}^{n} x_i v_i) \bmod n = L \bmod n$ exists, where each variable x_i can be either 0 or 1. Here $x \bmod y$ means the remainder when x is divided by y.
    The running time should be polynomial in $n + \log L$ (alternatively you can assume L fits in one word of memory, and bound the running time by a polynomial in n).

    (c) The problem is defined as follows:
    Input: positive integers $w_1, \ldots, w_n$, $v_1, \ldots, v_n$ and L.
    Output: The maximum possible value of $\sum_{i=1}^{n} x_i v_i$ subject to $\sum_{i=1}^{n} x_i w_i \leq L$ and each x_i is a nonnegative integer.
    The running time should be polynomial in $n + L$.

    (d) The problem is defined as follows: a
    Input: positive integers $v_1, \ldots, v_n$
    Output: Determine whether subset S of these integers exists such that $\sum_{v_i \in S} v_i^3 = \prod_{v_i \in S} v_i$.
    The running time should be polynomial in $n + L$, where $L = \max(\sum_{i=1}^{n} v_i^3, \prod_{i=1}^{n} v_i)$.

18. (4 points) The input to this problem is two sequences $T = t_1, \ldots, t_n$ and $P = p_1, \ldots, p_k$ such that $k \leq n$, and a positive integer cost c_i associated with each t_i. The problem is to find a subsequence of of T that matches P with maximum aggregate cost. That is, find the sequence $i_1 < \ldots < i_k$ such that for all j, $1 \leq j \leq k$, we have $t_{i_j} = p_j$ and $\sum_{j=1}^{k} c_{i_j}$ is maximized.
    So for example, if $n = 5$, $T = XYXXY$, $k = 2$, $P = XY$, $c_1 = c_2 = 2$, $c_3 = 7$, $c_4 = 1$ and $c_5 = 1$, then the optimal solution is to pick the second X in T and the second Y in T for a cost of $7 + 1 = 8$.

    (a) Give a recursive algorithm to solve this problem. Then explain how to turn this recursive algorithm into a dynamic program.

    (b) Give a dynamic programming algorithm based on enumerating subsequences of T and using the pruning method.

    (c) Give a dynamic programming algorithm based on enumerating subsequences of P and using the pruning method.

19. (6 points) Give a polynomial time dynamic programming algorithm for the following problem. The input consists of a two dimensional array R of non-negative integers, and an integer k. The value $R_{t,p}$ gives the number of of users requesting page p at time t (say from a www server). At each integer time, the server can broadcast an arbitrary collection of pages. If the server broadcasts the j pages $p_1, \ldots, p_j$ at time t, then the requests of all the users who requested pages $p_1, \ldots, p_j$ strictly before time t are satisfied. This counts as j broadcasts. The server can make at most k broadcasts in total

over all times. The goal is to pick the times to broadcast, and the pages to broadcast at those times, in order to minimize the total time (over all requests) that requests/users have to wait in order to have their requests satisfied, subject to the constraint that there are at most k broadcasts.

For example, if $k = 3$ and the array R was:

time	1	2	3
page A	0	2	0
page B	1	3	4

One schedule (which is presumably optimal) is to broadcast pages A and B at time 3, and page B again at time 4. This gives k=3 total broadcasts. The waiting time for the 2 page requests to page A at time 2 would be 1. The page request to B at time 1 would wait 2. The 3 page requests to B at time 2 would wait 1. And the 4 page requests to B at time 3 would wait 1. This gives total waiting time 2*1 + 1*2 +3*1 +4*1 = 11.

Hint: As a warm up, start with the case that there is only one page.