Kevin Hoser

Alex Schendel

Spring 2020

23 March 2020

**CS 324 HW 6: Dynamic Programming & Greedy Algorithms**

**Part A (20 points): can be done individually or in pairs, pairs preferred**

You will implement a recursive and dynamic programming solution to the equal partition problem, similar to what you did for the 0-1 knapsack problem. Note that there are many solutions available to this problem on-line, but you should implement this from your own brain(s) to get practice with dynamic programming. You may choose any programming language that you wish.

**Problem**: Given a list of positive integers, can the set be partitioned into two subsets that the sums of the elements in each subset are the same.

Example:

L = {1, 5, 11, 5}

The answer is true, since the subsets can be {1, 5, 5} and {11}, both with a total of 11.

Example:

L = {1, 3, 5}

The answer is false, since there are no two subsets with the same total.

Here is the pseudo-code for the recursive version. Note that we are turning the problem into – *is there one subset that adds to total/2*?

// Note: using 0-indexed arrays since most programming languages index arrays this way

EQ\_SUBSETS(L): // may need to pass in length of L as well, depending on PL

If L has at least one non-positive integer, return false

T = sum of elements in L

If T is odd, return false

N = L.length

EQ\_SUBSETS\_R(L, N, T/2) // see if there is a subset that totals T/2

EQ\_SUBSETS\_R(L, N, S): // N is length of L

If S == 0, return true // we have found a subset (empty) that adds to 0

If N==0 and S > 0, return false // out of elements and S > 0

If S < 0, return false // S < 0, so nothing left in sum

If L[N-1] > S, return EQ\_SUBSETS\_R(L, N-1, S) // too big to put in subset

return EQ\_SUBSETS\_R(L, N-1, S) || EQ\_SUBSETS\_R(L, N-1, S – L[N-1])

// first call does not put element in subset

// second call puts element in subset

**Implement** these functions and main. Test with the following arrays and answer question 2 below:

{3, 1, 5, 9, 12} // should return true

{10, 7, 4, 8} // should return false

{4, 5, 1, 11, 10, 3} // should return true

{4, 5, 6, 11, 120, 4} // should return false

Now, let’s see how we can turn this recursion into a dynamic programming solution. We can create a table of Booleans (or ints) to represent if a subset exists that sums to an intermediate value. So, the table will have (T/2)+1 rows and N+1 columns. We fill in a table entry with true if there is a subset up to j-1 items that sums to the value of the row index. Note that the pseudo-code assumes L is 0-indexed, so the last item is at position L[N-1]. Here is the pseudo-code for EQ\_SUBSETS\_DP:

EQ\_SUBSETS\_DP(L): // may need to pass in length of L as well, depending on PL

If L has at least one non-positive integer, return false

T = sum of elements in L

If T is odd, return false

N = L.length

Create table with (T/2)+1 rows and N+1 columns

Initialize top row of table with true (there is a subset that sums to 0)

Initialize left column except for [0][0] with false

For i from 1 to (T/2):

For j from 1 to N:

Table[i][j] = Table[i][j-1] // same return value for subset that

// sums to i without item j-1

// can have a subset that sums to i if we had a subset without

// item j-1 that sums to (i-L[j-1])

if i >= L[j-1]

Table[i][j] = Table[i][j] or Table[i-L[j-1]][j-1]

Return Table[T/2][N]

**Implement** the dynamic programming function. Test with the same input lists as before and answer question 3 below.

Now, add code to the dynamic programming function to print the table in tabular form (one row per line, with tabs in between). Run just test case 3 {4, 5, 1, 11, 10, 3}.

Include the table printout below to question 4.

**Name(s)**: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (just one submission per pair is needed)

(you can delete all the text above for your submission)

Submit answers to part A to moodle separately from part B.

1. (15 points) Upload your code file to the separate moodle submission link. If you worked with a partner, put both names above and in the code comments.
2. (2 points) What are the returned values for the test arrays above for the recursive solution? Include a screenshot of your results below.
   1. True/False
   2. True/False
   3. True/False
   4. True/False
3. (1 point) Do your answers with the dynamic programming version match that of the recursive version?
   1. Yes/No
4. (2 points) Copy and paste your table from test case 3 {4, 5, 1, 11, 10, 3} below.