CISC4080 Spring 2019
Final Lab: a team project on Dynamic Programming

**Goal:**

- Working in a team (of 1-3 members) to come up a solution to a problem (Graph or Dynamic Programming)

- Designing test cases to test the correctness of your algorithm

- For simple input, you might be able to calculate the correct output by hand, and compare algorithm output against that. This is hard to do for larger input (problem instances). Consider implement a brute-force algorithm to the problem, and compare your DP algorithm against it.

- Documentation of the project.
  - Write a report that includes the following sections:
    - Problem Analysis: restate the problem being solved, what kind of problems (is it optimization problem?)? what will be the brute-force solution and its running time? and so on.
    - Dynamic Programming solution: describe the subproblem, optimal substructure and overlapping subproblems; Running time analysis
    - The testcases that you will test your algorithm with
    - Code that implements the brute-force algorithm and dynamic programming algorithm
    - Output as a screenshot.

- You can choose to work on any of the following problems.

**Adding To N:**

Given a list of n positive distinct integers $a_1, a_2, \ldots, a_n$; a positive integer S decides whether there is a subset of the integer $a_i$’s that add up to S, each of the number can only be used once. If the answer is yes, output the subset too.

For example, given a list of 5 integers 3, 4, 8, 2, and $S=14$, the output is yes, as $4+8+2=14$. 
Note that this problem can also be given as a change making problem: where \(a_1, \ldots, a_n\) are values of the coins, and \(S\) is the amount of change to make using these coins where each coin can be used at most once.

**Change-Making Problem with at most k coins:**

Given an unlimited supply of coins of values \(a_1, a_2, \ldots, a_n\), we wish to make change for a value \(v\) using at most \(k\) coins. This might not be possible. Your algorithm should report whether it’s possible or not, and if it’s possible, output the coins used to make the change.

**Input:** \(a_1, a_2, \ldots, a_n; k; v\)

**Output:** yes and the value of the coins (up to \(k\)) used to make the change (\(v\)) or no.

For example,

If there are two types of coins, \(a_1=4, a_2=10, k=5,\) and \(v=15,\) the output is No.

If there are three types of coin, \(a_1=4, a_2=10, k=6,\) and \(v=48,\) the output is yes, and coins used are: 10, 10, 10, 10, 4, 4.

**Road Trip:**

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 \(a_1 < a_2 < \ldots < a_n,\) 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 particular 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 algorithms that determines the optimal sequence of hotels at which to stop.

**Input format:**

\(n, a_1, a_2, \ldots, a_n\)

// (i.e., the number of hotels, then the mile posts of the hotels)
Output format:
the sequence of hotels that you are staying at each night of the road trip, e.g., 3 6 9 10, means you are saying at the third hotel (with mile post a_3), then the sixth, ninth and finally reach the last hotel (n=10).

**Printing neatly:**

Consider the problem of neatly printing a paragraph with a monospaced font (all characters having the same width) on a printer. The input text is a sequence of n words, the lengths of the words are given as c_1, c_2, ..., c_n, measured in number of characters.

We want to print this paragraph neatly on a number of lines that hold a maximum of M characters each. Our criterion of "neatness" is as follows. If a given line contains words i through j, where i ≤ j, and we leave exactly one space between words, the number of extra space characters at the end of the line is

\[ M - j + i - \sum_{k=i}^{j} c_k \]

which must be nonnegative so that the words fit on the line. We wish to minimize the sum, over all lines except the last, of the cubes of the numbers of extra space characters at the ends of lines.

Give a dynamic-programming algorithm to print a paragraph of n words neatly on a printer.

Input format: n c1 c2 ... cn M
// For example, 20 3 6 2 8 3 4 9 10 2 3 3 4 9 2 7 3 10 2 9 3 30

Output format:
The range of words to be displayed on each line, e.g., the following output represents that we print 1st word to 6th word on the first line, and 7th words to 13th word on second line, and 14th to 20th word on last line.
1 6
7 13
14 20