# Project 2: Dynamic programming

COT 4400, Spring 2019

Due March 31, 2019

## 1 Overview

For this project, you will develop two algorithms to count the number of ways to sum numbers up to a specified total. Designing and implementing these algorithms will require you to model the problem using dynamic programming, then understand and implement your model.

You are only allowed to consult the class slides, the textbook, the TAs, and the professor. In particular, you are not allowed to use the Internet. This is a group project. The only people you can work with on this project are your group members. This policy is strictly enforced.

In addition to the group submission, you will also evaluate your teammates' cooperation and contribution. These evaluations will form a major part of your grade on this project, so be sure that you respond to messages promptly, communicate effectively, and contribute substantially to your group's solution. Details for your team evaluations are in Section 7.2. You will submit the peer evaluations to another assignment on Canvas, labelled "Project 2 (individual)."

**A word of warning:** this project is team-based, but it is quite extensive and a nontrivial task. You are highly encouraged to start working on (and start asking questions about) this project early; teams who wait to start until the week before the due date may find themselves unable to complete it in time.

## 2 Problem Description

For this project, you will be given an array of positive integers, *data*, and a target value, $t$, and you will need to count the number of ways that the numbers in *data* can sum up to $t$. In the first variation of the problem, you will count the different orders of the sums as different, while in the second variation, you will count different orders of the same collection as the same. (This is similar to the difference between permutations and combinations.) In both problems, you are allowed to use the same number from *data* as many times as you want. You may assume that *data* is sorted and does not contain any duplicates.

## 3 Example

If you were given the array $data = [1, 3, 5]$ and $t = 7$, there are 12 ways to add 1's, 3's, and 5's together to reach 7 if order matters:

1. $1 + 1 + 1 + 1 + 1 + 1 + 1$

2. $1+1+1+1+3$
3. $1+1+1+3+1$
4. $1+1+3+1+1$
5. $1+1+5$
6. $1+3+1+1+1$
7. $1+3+3$
8. $1+5+1$
9. $3+1+1+1+1$
10. $3+1+3$
11. $3+3+1$
12. $5+1+1$

If order doesn't matter, though, there are only four sums, as everything other than 1, 2, 5, and 7 in the list above is a shuffled version of one of these four.

## 4 Modelling the problem recursively

In order to write a dynamic programming algorithm for this problem, you will need to develop recurrence to solve both versions of the problem. I recommend that you start by developing a recurrence to solve the "order matters" problem first. To come up with this recurrence, think about what choices you can make for your first decision in forming these sums, then use recursion to count the sums for each of those possible decisions.

For the "order doesn't matter" problem, I recommend counting how many *sorted* sums there are that add up to $t$. This will ensure that you don't consider two different orders of the same values as the same. For this version, you'll need to think about how you count the sums recursively so that the recursive part of the sum has to be sorted relative to the decision you made initially.

## 5 Project report

In your project report, you should include brief answers to 9 questions. Note that you must use dynamic programming to solve this problem; other solutions will not receive substantial credit.

1. How you can break down a problem instance of the "order matters" problem for a given target $t$ and array *data* into one or more smaller instances? Your answer should include how the solution to the original problem is constructed from the subproblems.
2. What are the base cases of the "order matters" recurrence?
3. How you can break down a problem instance of the "order doesn't matter" problem for a given target $t$ and array *data* into one or more smaller instances? Your answer should include how the solution to the original problem is constructed from the subproblems.

4. What are the base cases of this "order doesn't matter" recurrence?

5. What data structure would you use to recognize repeated problems for each problem? You should describe both the abstract data structures, as well as their implementations.

6. Give pseudocode for a memoized dynamic programming algorithm to find the number of sums when *order matters*.

7. What is the *worst-case* time complexity of your memoized algorithm for the *order matters* problem?

8. Give pseudocode for a memoized dynamic programming algorithm to find the number of sums when *order doesn't matter*.

9. Give pseudocode for an iterative algorithm to find the number of sums when *order doesn't matter*. This algorithm does not need to have a reduced space complexity.

# 6 Coding your solutions

In addition to the report, you should implement a dynamic programming algorithms that can find the number of sum that add up to $t$ *both* when order matters and when it doesn't. Your code may be iterative or recursive, but both solutions must use dynamic programming. Also, you may code your solution in C++ or Java, but it must compile and run in a Linux environment. If you are using C++ and compiling your code cannot be accomplished by the command

```
g++ -o mover *.cpp
```

you should include a Makefile that is capable of compiling the code via the `make` command.

If you choose to implement your code in Java, you should submit an executable jar file with your source. In either case, your source code may be split into any number of files.

Your code will not need to handle invalid input (e.g., negative weight boxes or problem instances with no boxes).

## 6.1 Input format

Your program should read its input from the file `input.txt`, in the following format. The first line of the file has a positive integer $x$ specifying the number of problem instances. The rest of the file contains $x$ pairs of lines. The first line of each pair contains two positive integers, $t$ (the target value) and $n$ (the size of *data*), in that order. The next line will contain the $n$positive integers, representing the values of *data* in increasing sorted order. You may assume *data* will not contain any duplicates.

## 6.2 Output

Your program should write its output to the file `output.txt`. For each problem, your program should output the number of sums when order matters, followed by the number of sums when order doesn't matter. The two numbers should be separated by a space, and you should print the solution for each problem on a line (e.g., if there are 10 input problems, your output file should have 10 lines with two numbers on each line).

# 7 Submission

Your submission for this project will be in two parts, the group submission and your individual peer evaluations.

## 7.1 Group submission

The submission for your group should be a zip archive containing 1) your report (described in Section 5) as a PDF document, and 2) your code (described in Section 6). If your code requires more than a simple command to compile and run then you must also provide a Makefile and/or shell script. You should submit this zip archive to the "Project 2 (group)" assignment on Canvas.

Be aware that your project report and code will be checked for plagiarism.

## 7.2 Teamwork evaluation

The second part of your project grade will be determined by a peer evaluation. Your peer evaluation should be a text file that includes 1) the names of all of your teammates (including yourself), 2) the team member responsibilities, 3) whether or not your teammates were cooperative, 4) a numeric rating indicating the proportional amount of effort each of you put into the project, and 5) other issues we should be aware of when evaluating your and your teammates' relative contribution. The numeric ratings must be integers that sum to 30.

It's important that you be honest in your evaluation of your peers. In addition to letting your team members whether they do (or do not) need to work on their teamwork and communication skills, we will also evaluate your group submission in light of your team evaluations. For example, a team in which one member refused to contribute would be assessed differently than a team with three functioning members.

You should submit your peer evaluation to the "Project 2 (individual)" assignment on Canvas.

# 8 Grading

| **Report** | **40 points** |
|---|---|
| Questions 1 and 3 | 9 each |
| Questions 6 and 9 | 5 each |
| Question 8 | 4 |
| Questions 2, 4, 5, and 7 | 2 each |
| **Code** | **30 points** |
| Compiles | 5 |
| Uses correct input and output format | 5 |
| Computes correct answer | 15 |
| Good coding style | 5 |
| **Teamwork** | **30 points** |

Note that if your algorithm is inefficient, you may lose points for both your pseudocode and your submission. Also, in extreme cases, the teamwork portion of your grade may become negative or greater than 30. In particular, if you do not contribute to your group's solution at all, you can expect to receive an overall grade of 0 on the project.