

CS 421 Algorithms (Summer 2018)

Lab Assignment #2

Definition due on 06/04/2018 (1:00 PM), Programs due on 06/10/2018 (11:00 PM)

Description:

This assignment asks you to use two approaches, pure recursion and dynamic programming, to solve one non-optimization problem and one optimization problem.

For comparison purposes, in you programs, you need to count and print out the number of recursive calls for the recursion approach and the number of table references for the dynamic programming approach.

I. Shuffle-ness: The problem description is as follows.

Suppose we are given three strings of characters: $X = x_1x_2 \dots x_m$, $Y = y_1y_2 \dots y_n$, and $Z = z_1z_2 \dots z_{n+m}$, where each z_i may or may not be one of the characters in X and Y .

Z is said to be a **shuffle** of X and Y if Z can be formed by interspersing the characters from X and Y in a way that maintains the left-to-right ordering of the characters from each string. For example, `cchocohilaptes` is a shuffle of `chocolate` and `chips`, but `chocochilatspe`, `chocolatechipz` are not.

The following are what you need to do for this problem.

1. Let X_i be the prefix of X with i characters (similarly, prefix Y_i for Y and Z_i for Z). The recursive definition of the boolean solution for this problem is as follows.

Let $t(i, j)$ be the boolean answer for the sub-problem X_i, Y_j, Z_{i+j} . Suppose that we have a boolean function $b(c_1, c_2)$ which takes two characters c_1 and c_2 as inputs and returns true (if $c_1 = c_2$) or false (if $c_1 \neq c_2$). Please recursively define $t(i, j)$. Write down the recursive definition of $t(i, j)$ in a file **Shuffle.def**, where

$$t(i, j) = \begin{cases} \text{true} & \text{if } i = 0 \text{ and } j = 0 \\ \text{_____} & \text{if } i > 0 \text{ and } j = 0 \\ \text{_____} & \text{if } i = 0 \text{ and } j > 0 \\ \text{_____} & \text{if } i > 0 \text{ and } j > 0 \end{cases}$$

2. Please write programs with 3 command line arguments: **X**, **Y**, **Z** to determine whether or not Z is a shuffle of X and Y (a yes-no output). Please check whether the three input strings are compatible at the beginning of your program. If not, prompts users the usage of your program. **X**, **Y**, **Z** are compatible if the length of **X** plus the length of **Y** is equal to the length of **Z**.
3. For the dynamic programming approach, your program implements step 3 and 4 in the dynamic programming development process as we discussed in class. Step 4 is trivial in this problem. Your program should print out the table you construct in step 3 and the final **yes-no** solution, as well as the number of table references.

4. For the pure recursion approach, your program should output a **yes-no** solution and the number of recursive calls.

II. 0-1 Knapsack: The problem description is as follows (or as the 2nd paragraph on page 382 of the textbook).

A thief robbing a store finds n items; the i th item is worth v_i dollars and weights w_i pounds, where v_i and w_i are integers. He wants to take as valuable a load as possible, but he can carry at most W pounds in his knapsack for some integer W . Which items should he take? (This is called the **0-1 knapsack problem** because each item must either be taken or left behind; the thief can not take a fraction amount of an item or take an item more than once.)

The following are what you need to do for this problem.

1. Let $m(i, j)$ be the value of the most valuable load for a knapsack carrying at most j pounds and selecting from items $\{1, 2, \dots, i\}$. Please recursively define $m(i, j)$.
2. Your program should have 4 command line arguments: **n**, **W**, **w.txt**, **v.txt**, where **n** is the number of items, **W** is the maximum weight a knapsack can carry, **w.txt** is a file containing each individual item's weight (one weight a line) and **v.txt** is a file containing each individual item's value (one value a line).
3. For the dynamic programming approach, your program implements step 3 and 4 of the dynamic programming development process as we discussed in class. Your program should print out the table you construct in step 3, the optimal value, the optimal solution, as well as the number of table references.
4. For the pure recursion approach, your program should output the the optimal value, the optimal solution and the number of recursive calls.

You can find two sample input files **w.txt** and **v.txt**, and the corresponding **sample_result** file in the directory

`/home/JHyeh/cs421/labs/lab2/files/`

What you need to submit

1. **Shuffle.def**
2. Four programs: two for the pure recursion approach (please name them **ShuffleRec.java** and **KnapsackRec.java**) and two for the dynamic programming approach (name them **ShuffleDP.java** and **KnapsackDP.java**).

Definition Submission

Submit your definition file from **onyx** by copying the file to an empty directory (with no subdirectories) and typing the following FROM WITHIN this directory:

```
submit jhyeh cs421 p2
```

Program Submission

Please use only **Java** to write your programs.

Before submission, you need to make sure that your program(s) can be compiled and run in **onyx**. Submit your program(s) from **onyx** by copying all of your files to an empty directory (with no subdirectories) and typing the following FROM WITHIN this directory:

```
submit jhyeh cs421 p2
```