A CASE STUDY INTRODUCING DYNAMIC PROGRAMMING IN CS2 *

James R. Daehn and Jamil Saquer
Computer Science Department
Missouri State University
901 S. National Ave
Springfield, MO 65897
JDaehn@MissouriState.edu, JamilSaquer@missouristate.edu

ABSTRACT

To assess students' understanding and the advantages gained in being introduced to the concepts of dynamic programming early in their computer science experience, students were introduced to dynamic programming in CS2. Two lab assignments, one homework assignment and one quiz were given following an introduction to the topic in our CS2 course, Data Structures with C++. Our results reveal that types of problems addressed were fully comprehensible and we expect this comprehension to lend itself to later success in an Algorithms and Advanced Data Structures course that explores dynamic programming in greater depth.

INTRODUCTION

Dynamic programming (DP) is an important topic that students usually study in an algorithms course. In the fall 2016 semester, the computer science department at Missouri State University introduced a new major option, called Software Development. Students in this option are not required to take the algorithms course. Nonetheless, the faculty members at the CS department felt that students should be exposed to DP before graduation. In addition, the 2013 ACM/IEEE Curriculum Guidelines for Undergraduate Degree Programs in Computer Science list Dynamic Programming under Core-Tier 1 topics [1]. This means that DP should be included in the curriculum for the Software Development option.

The faculty members in the CS department felt that CS2 is a good place to introduce DP to students. This agrees with recommendations in the 2013 ACM/IEEE curriculum Guidelines, which state that DP may be covered under Fundamental Data Structures and Algorithms [1]. Since students in CS2 who are in the original major option, called Computer Science, are still required to take the algorithms course, extensive coverage of DP in CS2 may not be appropriate. Moreover, it would not be possible to do comprehensive coverage of DP in CS2 due to lack of time. In addition, many students in the algorithms course find DP a hard subject to understand. So, the authors of this paper felt that a simplified version of DP in CS2 is appropriate. Simplified here means using interesting problems with easy DP solutions at a level appropriate for students in a CS2 course. This would make it easy for students in CS2 to understand DP. It should also make the topic easier in the algorithms course when students are fully exposed to it and when harder problems are considered.

BACKGROUND

Students come to CS2 after having completed CS0 and CS1. In CS0, students get an overview of different areas in computer science and are introduced to programming. The amount of programming in CS0 has increased over time and currently more than half the semester is used to learn Programming in Python. In CS1, our students learn more Python with an emphasis on object oriented programming, GUI development and higher order functions. Then a transition to C++ is made to prepare the students to the data structures course, CS2, which uses C++ as the main programming language. On average, about 6 weeks are usually spent learning C++ in CS1. In CS2, the emphasis is on learning basic data structures such as stacks, queues, linked lists, trees and graphs. Students are also introduced to big-O notation and program efficiency. Fall 2016 was the first semester that dynamic programming was introduced in CS2.

STUDY DESCRIPTION

Two 50-minutes lectures were given to introduce dynamic programming concepts. This included the analysis of four problems outlined in [4]. The goal here was to have students recognize the two characteristics of overlapping sub-problems and optimal substructure that a problem should have to be an ideal candidate for solution using a dynamic programming approach [2]. Students were also shown how an inefficient recursive solution can be converted to an efficient solution using a top-down DP approach, aka memoization, and a bottom-up DP approach, aka tabulation, [2]. More emphasis was given to the bottom-up approach because the authors felt that it was easier for students to understand.

Labs

Two 100-minutes labs were given over a period of two weeks during the section on DP. The first lab was given in-between the two lectures that covered the topic while the second lab was given a week later, after the two lectures.

Lab 1: Two Dynamic Programming Solutions for Finding Fibonacci Numbers

The primary goal of this lab was to familiarize students with DP. It required them to implement both a top-down and a bottom-up solution to a familiar problem, that of finding the n^{th} Fibonacci number [5]. The lab materials provided a discussion of the top-down and bottom-up techniques along with pseudocode outlining these approaches. Students were asked to use appropriate data structures and programming practices to realize the pseudocode. Since this lab was conducted in-between the two lectures discussing DP, we choose to use the familiar problem of finding the n^{th} Fibonacci number to focus on the higher-level concepts of the two different approaches to DP. Furthermore, using this familiar problem also allowed us to continue a study of useful data structures (the primary focus of this course) by introducing students to associative arrays in C++.

Problem statement: Provide a top-down and bottom-up implementation that finds the nth Fibonacci number using dynamic programming. To complete this lab, students were required to implement the following two functions:

```
/**
* Dynamic, top-down approach to finding the nth Fibonacci number.
*
* @param n the number (index) in the sequence
* @return The nth Fibonacci number is returned.
*/
int topDownFib(int n);
/**
* Dynamic, bottom-up approach to finding the nth Fibonacci number.
*
* @param n the number (index) in the sequence
* @return The nth Fibonacci number is returned.
*/
int bottomUpFib(int n);
```

For the top-down implementation, students were suggested to use a global variable

```
std::map<int, int> m;
```

For the bottom-up function, students were encouraged to use local variables.

Lab 2: Spending as little money as possible

The second lab placed an emphasis on recognizing the two characteristics of a problem that lends itself to DP: overlapping sub-problems and optimal substructures [6]. It also required the students to provide a DP implementation of an interesting problem adopted from the web site GeeksForGeeks.org [3]. To minimize the probability that a simple Internet search would disclose the solution to this problem, identifiers associated with this problem were changed.

Problem statement: We seek to find a solution to a problem where we want to find an optimal path through a "liability matrix." Each cell in the matrix contains a positive integer value representing the liability associated with visiting that cell. Optimal, in this sense, means minimizing our "liability" by finding a path with the lowest liability from the upper left corner of the matrix (i.e., position (0,0)) to some general location (m, n) in the matrix. We can travel from cell (0, 0) to cell (m, n) by either moving to the right, or down, or diagonally in the matrix, one cell at a time. That is, from a given cell (i, j), you can go to any of (i, j + 1), (i + 1, j) or (i + 1, j + 1), which represent movements to the right, down and diagonally, respectively. When we visit a cell, we incur a certain liability.

The total liability in moving from (0, 0) to (m, n) is the sum of all the liabilities on that path (including both origin and destination).

To complete this lab, students were required to implement three functions. The following function pertains to DP:

```
/**
* Calculate the minimum liability amassed in traveling from the * upper
left-hand corner of a table to cell (m, n)
*
* @param table the two-dimensional array whose contents are liabilities
* @param m the number of rows in the two-dimensional array
* @param n the number of columns in the two-dimensional array
* @return The minimum liability incurred in traversing the table from
* the upper left-hand corner to the destination cell (m, n)
* @post The contents of the table are unchanged.
*/
int maxNumCoins(int** table, const int& m, const int& n);
```

This lab is similar in nature to and expands on two of the examples used in class to discuss DP. It provides an excellent tool to measure students' understanding of the material.

Homework

A homework assignment was given right after the second lecture on DP. It consisted of an optimization problem that is easily solved using the material learned in the lectures and labs. It provided the authors with another tool to assess students' understanding of the material and the students with a way to reinforce learning the DP technique.

Problem Statement: A table is composed of m x n cells. Each cell contains several golden coins. An example is shown in Table 1.

5	3	9
7	2	2
2	3	1

Table 1: Number of golden coins in each cell of a table

You start at the upper-left cell and want to reach the bottom-right cell. At each step, you can go either down or right one cell. Write a program that uses dynamic programming to find the maximum number of golden coins you can collect. The output from your program should display the cache table where the maximum number of golden coins collected on a path from the top-left cell to cell (r, c) is stored in cell (r, c). Also, output a message indicating the maximum number of coins on a path from the top-left cell to the right-bottom cell. Sample output for the above grid is as follows:

```
5   8   17
12  14  19
14  17  20
Maximum number of golden coins we can collect is 20
```

Quiz

Before starting the second lab, students logged into Blackboard to take a short quiz related to basic concepts surrounding DP that they had been exposed to prior to coming to the lab. Students take such quizzes every other week at the start of lab. The quizzes are short and typically consist of multiple-choice or True/False questions. Below are the three questions that were given:

1. Multiple Choice (1 point): Two aspects of a problem that lends the problem to a solution using dynamic programming are:
   a. Optimal substructure
   b. Overlapping sub-problems
   c. Mutual exclusion
   d. Data abstraction
2. True/False (0.5 points): Dynamic Programming solutions are always recursive solutions.
3. Multiple Choice (0.5 points): The technique of storing solutions to sub-problems instead of re-computing them is called
   a. Memoization
   b. Memorization
   c. Recursion
   d. Stacking

Using the tools built in by Blackboard, the order of these questions and the order of the options for the multiple-choice questions are randomized.

RESULTS

Table 2 gives a summary of the grades earned in the labs, homework assignment and quiz given over the dynamic programming material. The row labeled "Adjusted Average" is computed by removing some 6-7 zeros (out of 48 students) that correspond to students that did not complete the assignment.

	Lab 1	Lab 2	HW	Quiz
Maximum	100%	100%	100%	100%
Minimum	0%	0%	0%	0%
Minimum Removing 0s	63%	95%	80%	25%
Median	100%	97%	100%	100%
Mean	87%	83%	77%	76%

Standard Deviation	0.31	0.35	0.40	0.35
Adjusted Average	97%	98%	98%	87%

Table 2: Dynamic Programming Assignment Results

DISCUSSION

It is noteworthy to mention that students in this course conduct their labs in a peer-programming format. They use version control (git on BitBucket) to manage and complete their assignments. Every other week, students are paired up with a new partner. The first week, one person owns the repository on BitBucket and is responsible for making the appropriate commit and pull requests. In the second week of this pairing, the other partner assumes these responsibilities. Regardless of the week, both members of the team must actively contribute to the solution of the lab. Thus, in the results shown above, the teams remained the same with the two labs discussed. Homework assignments and quizzes, however, are individual efforts. As such, half of the results come from individual efforts and the other half come from pair-programming efforts.

The high grades reflect students' understanding of the material related to dynamic programming. Finally, it is also worth mentioning that the students found this material exciting. There were two outcomes they found appealing:

1. The runtime improvements provided by these techniques.
2. The relative ease in implementing these changes, i.e., in going from a naïve recursive solution to an elegant solution based on DP principles.

CONCLUSIONS AND PLANS FOR FUTURE WORK

The results presented here suggest that the concepts presented were understandable and are promising. We expect that this early exposure to dynamic programming will afford the students greater chances for success in subsequent courses that may explore this technique in greater depth or as applied to more difficult problems. For future work, we plan to perform a study in the Algorithms and Advanced Data Structures course to compare the performance of students who were introduced to DP in CS2 with those who were not.

REFERENCES

[1] ACM/IEEE Joint Task Force on Computing Curricula, Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, https://www.acm.org/education/CS2013-final-report.pdf, retrieved November 23, 2016.

[2] Cormen, T. , Leiserson, C., Rivest, R. , Stein, C. , Introduction to Algorithms, 3rd ed., Cambridge, MA: MIT Press, 2009.

[3] Geeks For Geeks, Dynamic Programming - Set 6 (Min Cost Path), http://www.geeksforgeeks.org/dynamic-programming-set-6-min-cost-path/, retrieved November 25, 2016.

[4] Saquer, J., Smith, L., Simplifying dynamic programming, *Proceedings of the Fifth International Conference on Informatics and Applications*, 159-164, 2016.

[5] Saquer, J., Daehn, J., Lab 9 - Two dynamic programming solutions for finding Fibonacci numbers, https://bitbucket.org/professordaehn/msu-csc232-lab09, November 1, 2016.

[6] Saquer, J., Daehn, J., Lab 10 - Spending as little money as possible, https://bitbucket.org/professordaehn/msu-csc232-lab10, November 8, 2016.

[7] Saquer, J., Daehn, J., HW 4: Dynamic programming - maximum number of gold coins, https://bitbucket.org/professordaehn/msu-csc232-hw04, November 6, 2016.