Project 4: Dynamic and Exhaustive

Author: Diego Vela

e-mail: dvela2@csu.fullerton.edu

Instructor: Zhiyuan Yang

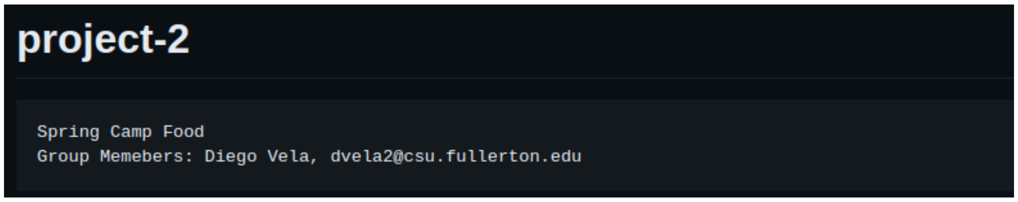
May 8, 2024

INTRODUCTION

A summer camp is expecting lots of students over the summer and needs to order canned, fresh, or frozen items. Each product has a certain weight in ounces and a total calorie content. We are given a set of these products and are asked to design algorithms to pick a subset of food products that fit within a given the number of calories one can obtain by eating them while maximizing total weight.

The objective of the report is to compare our previous algorithm in which we discovered exhaustive algorithms are feasible to implement and produce correct outputs but algorithms with exponential running times are too slow to be practical. A dynamic and exhaustive algorithm will be compared and analyzed in their step counts and time complexity using visual and mathematical data to prove the efficiency and success of dynamic programming.

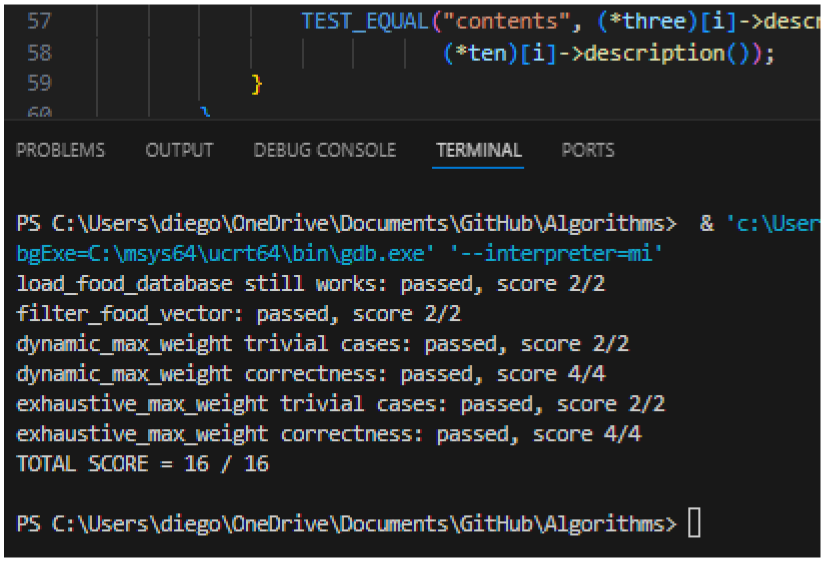
README.MD



project-2

```
Spring Camp Food
Group Memebers: Diego Vela, dvela2@csu.fullerton.edu
```

CODE EXECUTION



```
                TEST_EQUAL("contents", (*three)[i]->descr
                        (*ten)[i]->description());
            }
```

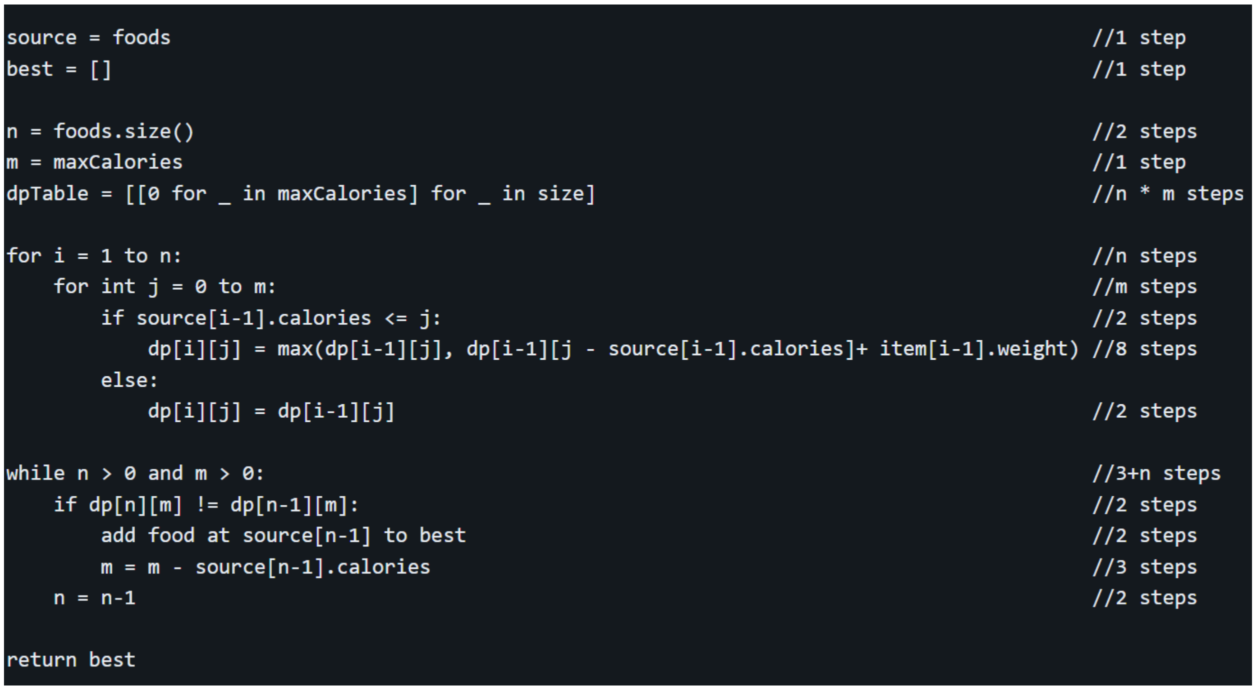
```
PS C:\Users\diego\OneDrive\Documents\GitHub\Algorithms>  & 'c:\User
bgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
load_food_database still works: passed, score 2/2
filter_food_vector: passed, score 2/2
dynamic_max_weight trivial cases: passed, score 2/2
dynamic_max_weight correctness: passed, score 4/4
exhaustive_max_weight trivial cases: passed, score 2/2
exhaustive_max_weight correctness: passed, score 4/4
TOTAL SCORE = 16 / 16

PS C:\Users\diego\OneDrive\Documents\GitHub\Algorithms> 
```

MATHEMATICAL ANALYSIS

Assumptions

- The sum_food vector function will be counted as 1 step.
- The max function will be counted as one additional step.
- The size function will be counted as one additional step.
- Adding an item to best will count as one additional step.
- n > m so n*m will be reduced to n^2 in mathematical analysis.

Dynamic Algorithm: PseudoCode



```
source = foods                                                                  //1 step
best = []                                                                       //1 step

n = foods.size()                                                                //2 steps
m = maxCalories                                                                 //1 step
dpTable = [[0 for _ in maxCalories] for _ in size]                              //n * m steps

for i = 1 to n:                                                                 //n steps
    for int j = 0 to m:                                                         //m steps
        if source[i-1].calories <= j:                                           //2 steps
            dp[i][j] = max(dp[i-1][j], dp[i-1][j - source[i-1].calories]+ item[i-1].weight) //8 steps
        else:
            dp[i][j] = dp[i-1][j]                                               //2 steps

while n > 0 and m > 0:                                                          //3+n steps
    if dp[n][m] != dp[n-1][m]:                                                  //2 steps
        add food at source[n-1] to best                                         //2 steps
        m = m - source[n-1].calories                                            //3 steps
    n = n-1                                                                     //2 steps

return best
```

Dynamic Algorithm: Mathematical Analysis

$$BigO = 1 + 1 + 2 + 1 + n * m + (\sum_{1}^{n}\sum_{1}^{m} max(2 + 8, 2)) + max(n, m) + 3$$

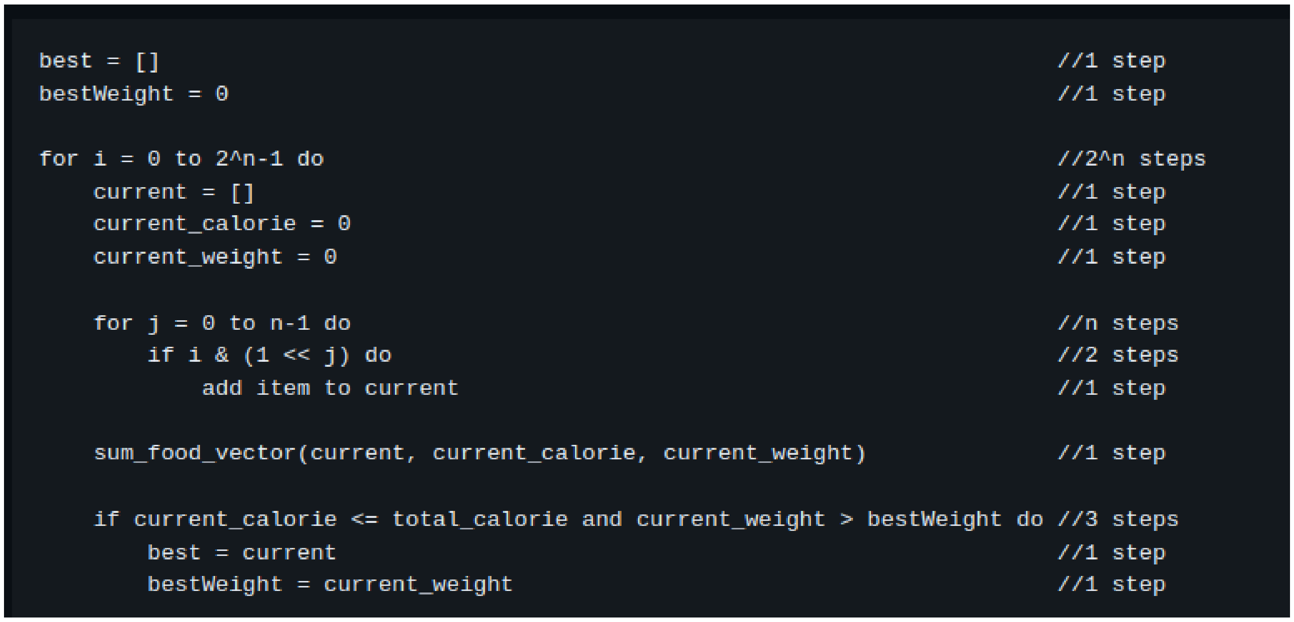
$$=> 5 + n * m + (\sum_{1}^{n} 10m) + n + 3$$

$$=> 8 + n * m + 10(n * m) + n$$

$$=> 11n * m + n + 8$$

$$11n^2 + n + 8 = O(n^2)$$

Exhaustive Algorithm: PseudoCode



```
best = []                                                               //1 step
bestWeight = 0                                                          //1 step

for i = 0 to 2^n-1 do                                                   //2^n steps
    current = []                                                        //1 step
    current_calorie = 0                                                 //1 step
    current_weight = 0                                                  //1 step

    for j = 0 to n-1 do                                                 //n steps
        if i & (1 << j) do                                              //2 steps
            add item to current                                         //1 step

    sum_food_vector(current, current_calorie, current_weight)           //1 step

    if current_calorie <= total_calorie and current_weight > bestWeight do //3 steps
        best = current                                                  //1 step
        bestWeight = current_weight                                     //1 step
```

Exhaustive Algorithm: Mathematical Analysis

$$BigO\ =\ 1 + 1 + \sum_{1}^{2^n} 1 + 1 + 1 + (\sum_{1}^{n} 2 + 1) + 1 + 3 + 1 + 1$$

$$=>\ 2 + \sum_{1}^{2^n} 9 + \sum_{1}^{n} 3$$

$$=> 2 + \sum_{1}^{2^n} 9 + 3n$$

$$=> 2 + 9 * 2^n + 3n * 2^n$$

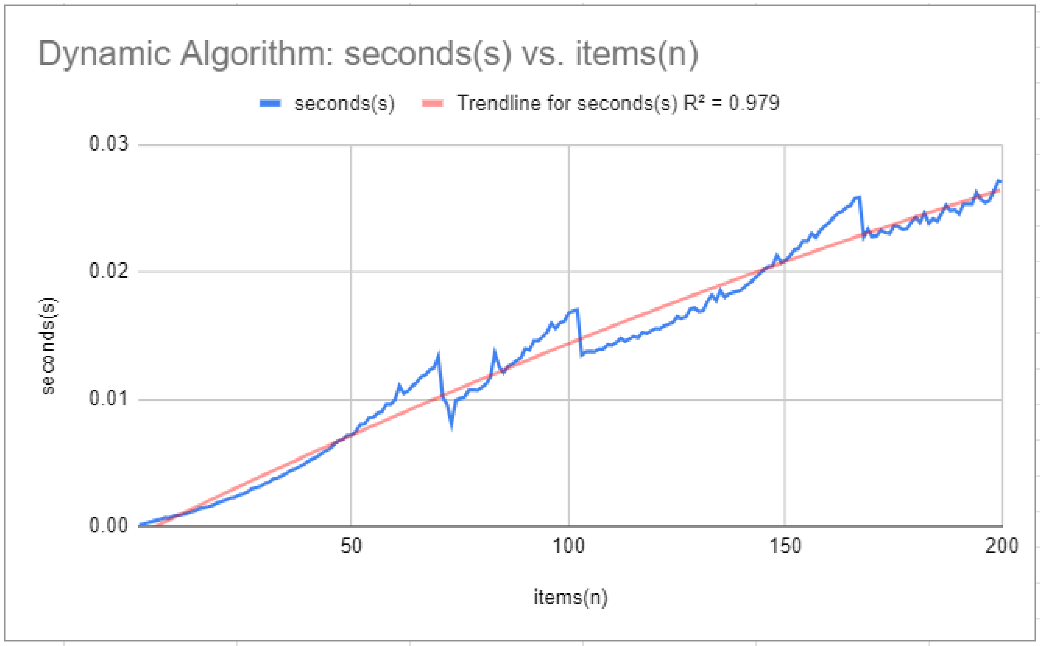
$$3n * 2^n + 9 * 2^n + 2 = O(2^n * n)$$

Summary

The dynamic programming algorithm has a lower time complexity than the exhaustive algorithm. The exhaustive algorithm is expected to run at an exponential growth rate while the dynamic programming algorithm is expected to run at a polynomial growth rate of n^2.

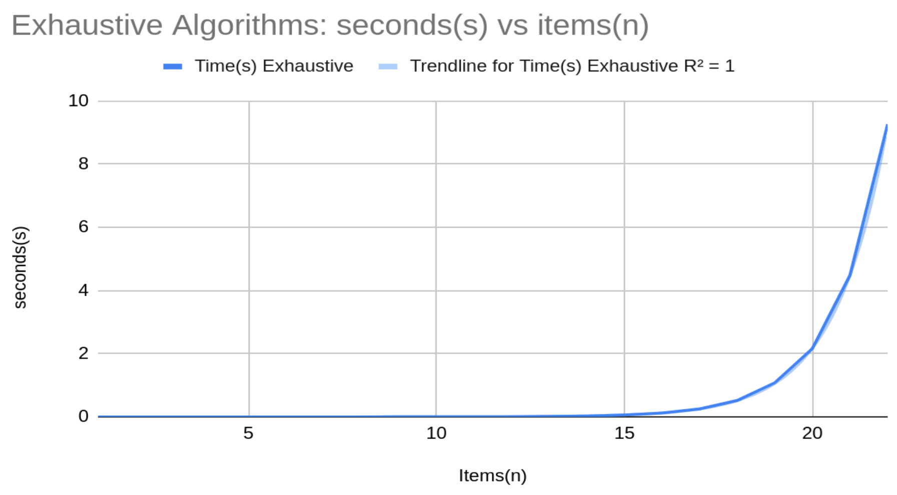
SCATTERPLOTS

Greedy Scatterplot



- The trendline that best fit was a linear trendline which is consistent with the time complexity analysis. The coefficient of determination is 0.979 which signifies a strong correlation with a polynomial time complexity.

Exhaustive Scatterplot



- The trendline that best fit was an exponential trendline which is consistent with the time complexity analysis. The coefficient of determination is 1.

CONCLUSION

In this project, the time complexity of an exhaustive algorithm was compared to a dynamic programming algorithm in a real world problem. Given a set of food items with a calorie and weight value, the algorithms found the subset to maximize the weight given a max amount of calories. Using this problem, an exhaustive algorithm was proven to be accurate and simple in terms of generating the best results and in implementation respectively. It was also re-iterated that although an exhaustive algorithm can be accurate, the computational time it takes to generate outputs is too slow to be feasible in large sets of data. Dynamic programming is able to achieve similar accuracy while cutting down the time complexity by a significant amount. The substantial difference in efficiency between the exhaustive and dynamic programming algorithms highlights how algorithmic efficiency becomes increasingly crucial as the size of the input data grows.