Design and Analysis of Algorithm for Maximizing the Profit of Producing a given Number of Items based on Profit and Production Rates of Producers

Nairit Banerjee
IIT2016505

Kamal Nayan Chaubey
IIT2016510

Ashutosh Vishwakarma
IIT2016017

Mandeep Chakma
IIT2016012

April 15, 2018

Abstract

This document is a final report of our study on maximizing the overall profit of producing n given items with the help of p producers such that one producer produces only one item. The profit is maximized on the basis of two matrices given. First matrix, describing rate of profit of producing each item by each producer. Second matrix, describing rate of production of each item by each producer. Here rate refers to production per unit time.

Keywords: *Bit Masking,Recursion,Matrix, Dynamic Programming*

1 Introduction And Literature Survey

The approach we are using to solve the problem is using Bit Masking and Dynamic Programming. Mask in Bitmask means hiding something. Bitmask is nothing but a binary number that represents something. Here we mask the producers array to represent which item is alloted to which producer. We will have a profit matrix of size p*n in which there will be profit for each item that each producer produces. This matrix is achieved by multiplying the rates of productions to the rate of profit for each producer-item pair. Applying Dynamic Programming brings down the running time by a significant amount. Where the brute-force algorithm has complexity O(n!), the dynamic programming approach has complexity

$$O(2^n n)$$

but at the cost of space complexity of

$$O(2^n)$$

2 Algorithm Design

Now the approach is simple that we are recursively going in to the matrix starting from column 1 and row 1 and we are taking at each recursive step the maximum value of profit till that call. For assigning the items to the producers, we are masking the producers with an binary array of bits. That is if the ith bit of mask is set than the ith producer has produced his one item. We then compute the profit for this allotment scheme, if not already computed, compare with the current maximum and update it, if required.

2.1 Pseudo Code

Algorithm 1 MAXIMUM

```
1:  procedure MAXIMUM(id, mask)
2:      if (mask == (1 << p) − 1)||(id > n) then
3:          return 0
4:      end if
5:      if dp[id][mask]! = 1000000000 then
6:          return dp[id][mask]
7:      end if
8:      i ← 0
9:      ans ← 0
10:     while i < p do
11:         if mask AND (1 << i) then
                continue
12:         end if
13:         ans ← max(ans, profit[i][id − 1] +
        MAXIMUM(id + 1, mask|(1 << i)))
14:     end while
15:     dp[id][mask] ← ans
16: end procedure
```

Algorithm 2 Main_Program

```
1: procedure MAIN
2:     i ← 0
3:     j ← 0
4:     x ← 0
5:     y ← 0
6:     while x < 100 do
7:         while y < 100 do
8:             dp[x][y] ← 1000000000
9:         end while
10:    end while
11:    while i < p do
12:        while j < n do
13:            scan x
14:            y ← ceil((1.0)/x)
15:            profit[i][j] = profit[i][j] * y
16:        end while
17:    end while
18:    Print(MAXIMUM(1,0))
19: end procedure
```

3 Analysis and Discussions

3.1 Time Complexity

The time complexity for the above stated algorithm can be computed by adding units of time taken by each statement multiplied by the number of times the statement is executed. Hence,

$T = \Sigma(cost(i) * no.of.times(i))$,for all statements

The running time will mainly depend on the number of producers given. We mask the producers to represents the various allotments of items to be produced. Hence if we take p as the number of producers and n as the number of items, we get,

$$T_{best} \propto npc_0 + 2^p pc_1 + c$$

$$T_{avg} \propto npc_0 + 2^p pc_2 + c$$

$$T_{worst} \propto npc_0 + 2^p pc_3 + c$$

Hence the order of growth for best case, average case and worst case are

$\Omega(2^p p), \Theta(2^p p), O(2^p p)$ respectively.

There isn't any best, average or worst case as such, as we compute results for all possible allotments irrespective of input. Thus all three expressions are structurally similar. As we are applying Dynamic Programming, the number of times we look for already computed results may affect the running time. Though we cannot have a clear distinction, we can say the number of hits to the look-up table may result in best, worst or average cases.

This is a table containing running time units for values of p ranging from 3 to 7. These values should be kept small as exponential time complexity results into huge running times for bigger values.

Table 1: Data for plotting running time graph

p	Units of Time
3	503
4	1362
5	3402
6	8935
7	21304

We will get the following running time graph for n ranging from 3 to 7. The x-axis represents the number of producers and y-axis represents the units of time taken.

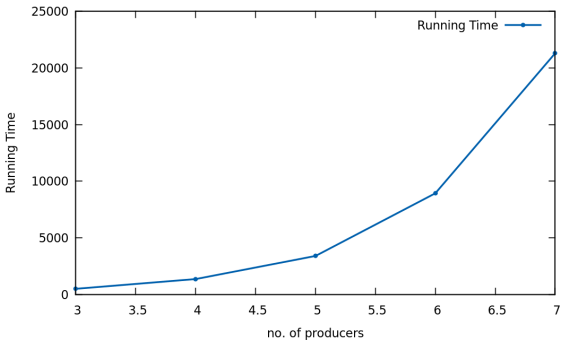

Figure 1: Running time graph.

3.2 Space Complexity

Here we have two matrices of size n*p. These are given as input, so we are not taking them into consideration while computing time complexity. We are using extra space to store the results we obtain at each step in order to implement Dynamic Programming. Hence,

$$Space(n) \propto 2^n$$

4 Experimental Study

Basically all the cases that is best, average and worst will take almost the same time but there might be some differences in their execution time because of accessing of the 2d Dynamic Programming array.

Best Case: If the input is such that the Dynamic Programming table is accessed a lot then clearly it will take the least time as it do not need to take a lot of time to calculate it again.

Average Case: The input which take intermediate level of accessing of Dynamic Programming table will constitute the average case scenario.

Worst Case:When the Dynamic Programming table is least used to calculate the result. This type of input will constitute the worst case.

ation is required to either minimize or maximize a function.

5 Conclusion

In this paper we proposed an algorithm to assign items to be produced to some producers, so that profit generated can be maximized.

This has many real life applications, like optimizing productivity, or profit, job assignment, optimizing salaries to employees and several such scenarios, where optimization is required to either minimize or maximize a function.

References

[1] Introduction To Algorithms (Second Edition) by Thomas H.Cormen, Charles E.Leiserson , R.L.Rivest and Clifford Stein.

[2] How to Solve it by Computer by R.G.Dromey.

[3] The C Programming Language by Brian Kernighan and Dennis Ritchie.

[4] Dynamic Programming and Bit Masking: https://www.hackerearth.com/practice/algorithms/dynamic-programming/bit-masking/tutorial/