National Institute of Technology
Silchar

Department of Computer Science and Engineering

Processing Power Assembly Line Scheduling Using Tabulation Dynamic Programming Algorithm

mini Project set 7 - Q.86

mini-Project submitted to:
Dr. Shyamosree Pal

Bachelor of Technology Mini-Project
Algorithm
CS-206

Swagat S. Bhuyan
18-1-5-059

August 2020
Silchar, Assam

Contents

0.1 Abstract

Dynamic Programming is an Algorithmic Paradigm of Coding which is usually used to solve problems that have a tendency to overlap its sub-sections, hence making it tedious and far more complex than it initially shows itself to be. Dynamic Programming approach to solving such problems helps overcome overlapping, hence making the code more efficient and the time of the code less complex and faster. It is usually handy when solving to find an optimal solution to a problem, i.e.- finding the most cost efficient solution out of all the feasible solutions. This is an Optimization Problem, and algorithms like Dynamic Programming, Greedy Algorithms, etc. have a knack for crushing such problems.

Dynamic Program depends on finding optimal sub problems of the given problem till each sub-section is solvable, all while keeping tabs of repetitions and overlapping in a tabulation format in order to avoid running the same calculations multiple times. Memoization and Tabulation Methods are used efficiently in Dynamic Programming to avoid overlapping and solve the problem in a much simpler time complexity.

The frequently encountered "Assembly Line Scheduling" problem is an interesting problem that can be solved using Dynamic Programming. Here, 2 Assembly lines are present each with a stipulated service time, stations, enter/exit times, assembly line transfer cost, etc. The idea is to find the most cost effective path via machine transfers and assembly line scheduling using Tabulation in Dynamic Programming. This way, path finding is made less complex as optimal paths from a given station at the assembly line are already tabulated for later use, eradicating the possibility of overlapping.

Chapter 1

Question Q.86

Given a sequence $a_1, a_2, a_3.....a_n$ and $b_1, b_2, b_3.....b_n$ where $a_i > 0, b_i > 0$ are processing power of machines A and B in the ith minute respectively. Given a sequence of n minutes, a plan is specified by a choice of A, B or 'move' for each minute, with the property that choices A and B cannot appear in consecutive minutes. For example, if your job is on machine A in minute i and you want to switch to machine B then your choice for minute i+1 must be 'move' and then your choice for minute i+2 can be B. The value of a plan is the total number of steps that you manage to execute over the n minutes: so it's the sum of a_i over all minutes in which the job is on A plus the sum of b_i over all minutes in which the job is on B. Give an efficient algorithm that takes values for $a_1, a_2, a_3.....a_n$ and $b_1, b_2, b_3.....b_n$ and returns the value of an optimal plan (i.e., a plan of maximum value).

Chapter 2

Project Description

In the problem given, it has been clearly stated that there are two Arrays given, say A and B. These are said to store the processing powers of machines A and B in the ith minute. We can correlate this to a combination of Job Scheduling and Assembly Line Scheduling. The catch being, machine transfer is allowed only under the condition that we sacrifice a unit time slot to switch from machine A to B, or vice versa. We thus have to find the best possible set of Processing Power Value sum that can be obtained from a given table of processing powers of machines A, B.

This is clearly an Optimization problem, as we are given the task to find out the max value feasible of processing powers from machines A and B, taking into condition the circumstances of machine transfers. For this we use Dynamic Programming Algorithm as this will effectively find the required optimized path without the possibility of overlapping, as we will be tabulating our taken optimized paths in a separate table. Meaning, we will be avoiding taking the same optimization paths time and again, reducing the complexity of this code drastically.

A trade-off has therefore been established between time complexity and space complexity as we will be requiring extra space for tabulation. This will also ensure building up from the bottom a tabulation keeping record of all optimal paths taken between the assembly lines of processing powers of machines A and B.

Chapter 3

Algorithm

3.1 ALGORITHM STEPS

1. We already have Machine Processing Power Sequences at each time slot up to n, int the arrays A, B. We take two more tabulation arrays X, Y and store the initial values of A and B in X and Y respectively. This takes O(n) time complexity, as we iterate through the 1D arrays A and B once.

2. Let each unit time slot (1 minute) in the assembly lines be S_{ij}, where i denotes Machine A if 0, Machine B if 1. consider them as stations.

3. **BASE CASE:** Considering X to be denoting the Assembly line for power processing machine A, Y be denoting the Assembly line for power processing machine B. Hence, if we consider time S_{0j}, where $0 < j < n$, the only possible backtracking paths from the position are if:

   - There was no change in Machine, hence the previous slot was S_{0j-1}. No Machine Transfer occurred here, therefore no change in value of i.
   - There was a Machine Transfer, hence it came from slot S_{0j-2}. Notice how two indexes are skipped backtracking, as we have already established that machine transfer requires a minute, hence a slot has to be skipped.

*X[0], Y[0], A[0], B[0] = INT_MIN for smooth functioning of code

4. **ITERATIVE CASE:** We then iterate through both the arrays X and Y from the second column tabulate from bottom up the optimal paths to be backtracked from the position of i. Let current position of index for X and Y be i, then:

$$\text{X[i]} \leftarrow X[i] + max(X[i-1], Y[i-2])$$
$$\text{Y[i]} \leftarrow Y[i] + max(Y[i-1], X[i-2])$$

5. After iterating through the the whole table, and updating the values of X[i], Y[i] accordingly, we find that the final indexes, i.e.- X[n], Y[n] now contain two optimal possibilities of backtracking paths. Meaning, X[n] contains the optimal path containing X[n], Similarly, Y[n]. therefore, to find maximum Processing Power Schedule value, we find the max optimized paths out of the two.

$$\text{backtrack} = \max(\text{X[n]}, \text{Y[n]})$$

6. After Backtracking, we can further make the algorithm more efficient by also forward propagating and comparing with previous optimal solution for better optimal solution if possible. (rare case).

$$\text{X[i]} \leftarrow X[i] + max(X[i+1], Y[i+2])$$
$$\text{Y[i]} \leftarrow Y[i] + max(Y[i+1], X[i+2])$$
$$\text{forward\_propagation} = \max(\text{X[n]}, \text{Y[n]})$$

3.2 ALGORITHM PSEUDO-CODE

Algorithm 1 Find Optimal Schedule Plan Value for Machine A and B

int ScheduleTimeProcessingPower(int* A, int* B, int n)

1. **for** i = 0 to n **do**
   X[i] = A[i]
   Y[i] = B[i]
2. **end for**
3. **for** i = 2 to n **do**
   X[i] $\leftarrow X[i] + max(X[i-1], Y[i-2])$
   $Y[i] \leftarrow Y[i] + max(Y[i-1], X[i-2])$
4. **end for**
5. backtrack = max(X[n], Y[n])
6. **for** i = 0 to n **do**
   X[i] = A[i]
   Y[i] = B[i]
7. **end for**
8. **for** i = n-1 to 1 **do**
   X[i] $\leftarrow X[i] + max(X[i+1], Y[i+2])$
   $Y[i] \leftarrow Y[i] + max(Y[i+1], X[i+2])$
9. **end for**
10. forward = max(X[n], Y[n])
11. return max(backward, forward)

3.3 TIME COMPLEXITY ANALYSIS

1. Here, a single 1D iteration occurs to store values of A, B in X, Y respectively. **COMPLEXITY:** $\theta(n)$
2. $\theta(1)$

3. This for loop runs n-2+1 times = n - 1 time. Both codes inside the loop consume constant complexity as they are just assigning calculations with $\theta(1)$. **COMPLEXITY:** $\theta(n)$

4. $\theta(1)$

5. Just Assigning value to variable. **COMPLEXITY:** $\theta(1)$

6. Here, a single 1D iteration occurs again to store values of A, B in X, Y respectively again for forward tracking. **COMPLEXITY:** $\theta(n)$

7. $\theta(1)$

8. This for loop runs n-2+1 times = n - 1 time. Both codes inside the loop consume constant complexity as they are just assigning calculations with $\theta(1)$. **COMPLEXITY:** $\theta(n)$

9. $\theta(1)$

10. $\theta(1)$

11. $\theta(1)$

Therefore, Total Time Complexity of this Dynamic Programming Algorithm is:

$\theta(n) + \theta(1) + \theta(n) + \theta(1) + \theta(1) + \theta(n) + \theta(1) + \theta(n) + \theta(1) + \theta(1) + \theta(1)$

FINAL TIME COMPLEXITY: $= \theta(n)$

BEST/WORST CASE: Here, Since the loop will iterate through irrespective of machine transfers, there is very little difference between best and worst case, hence we can safely use theta $\theta(n)$ as time complexity for all cases. *we can say that non transferred pathways show the best results as optimal solution is in the same assembly line throughout.

Chapter 4

Input/Output

According to the code, that will be presented later, everything has been randomized: Slots in Assembly Line (n), Processing Powers of Machine A in ith minute and Processing Powers of Machine A in ith minute. This way, a very versatile range of inputs can be expected, as a RANGE_LIMITER is also in place to be set by the user for better user experience.

1. Here, n = 8, Range of A: 4, Range of B: 5

```
Enter Flow Schedule Time-span: 8

Enter MAX_RANGE of Assembly Line Processing Powers for Machine A: 4

Enter MAX_RANGE of Assembly Line Processing Powers for Machine B: 5

Given Processing Power Table at ith minute for machines A, B:
        A: 4 3 4 4 2 2 1 1
        B: 2 5 5 1 5 3 4 1

Required Time Flow Scheduling Plan with Machine Transfers:
B B B B B B B B
Optimal Plan Value: 26

Press any key to continue . . .
```

2. Here, n = 14, Range of A: 20, Range of B: 20

```
Enter Flow Schedule Time-span: 14

Enter MAX_RANGE of Assembly Line Processing Powers for Machine A: 20

Enter MAX_RANGE of Assembly Line Processing Powers for Machine B: 20

Given Processing Power Table at ith minute for machines A, B:
        A: 4 1 8 4 9 15 19 16 18 10 4 4 17 8
        B: 8 2 4 19 4 1 11 8 10 2 16 8 8 14

Required Time Flow Scheduling Plan with Machine Transfers:
B B B B B B B move A move B move A A
Optimal Plan Value: 147

Press any key to continue . . .
```

3. Here, n = 30, Range of A: 10, Range of B: 10

```
Enter Flow Schedule Time-span: 30

Enter MAX_RANGE of Assembly Line Processing Powers for Machine A: 10

Enter MAX_RANGE of Assembly Line Processing Powers for Machine B: 10

Given Processing Power Table at ith minute for machines A, B:
        A: 8 3 5 7 8 7 6 9 6 6 1 4 6 10 7 2 1 7 5 7 10 4 9 3 4 6 8 9 5 3
        B: 1 2 9 2 3 6 9 1 10 8 3 8 4 9 9 2 1 4 5 2 6 5 4 8 7 7 8 5 8 10

Required Time Flow Scheduling Plan with Machine Transfers:
B move A A A move B move A A move B B B B B B B B B B B B move A move B B move
Optimal Plan Value: 184

Press any key to continue . . .
```

Chapter 5

Proof of Correctness of the Algorithm

It is equally important to prove our findings to validate this mini project. For this, we will be approaching with the Loop Invariant methodology.

1. **INITIALIZATION:** For Back Propagation, we start from i = 2. Therefore the First Slot in the Assembly Line is S_{i2} where i = 0, 1 depending on whether the selected slot in on the Machine A Assembly Line or Machine B Assembly Line.

2. **MAINTENANCE:** At any given Time Slot S_{ij}, we can be sure that S_{ij-1} for both i = 0, 1 (machine A and B) have been tabulated at their optimized Assembly Line Processing Power Pathway, Meaning, We can easily refer to S_{ij-1} for future calculations of optimization instead of traversing all the way to the beginning of the assembly line to obtain the optimal Schedule Plan.

3. **TERMINATION:** As the loop reaches n, the backtracking can be terminated successfully with X[n], Y[n] being the optimal values for each pathway. We simply find the larger of the two to return as the final optimal solution to the Processing Power Assembly Line Scheduling Using Tabulation Dynamic Programming Algorithm problem.

Chapter 6

Justification

This bottom up approach of building a separate tabulation resembling to the Assembly Line Scheduling Problem Model is really efficient as it has for the most part provided results in an exceptionally low complexity of O(n).

But there were other algorithms in the vicinity of trials as well, such as a hybrid of the Greedy Algorithm Johnson's, which may have been helpful if the question was not rigidly related to the time slot conditions. Moreover, it has little to no accommodations for facilitating Machine transfers between the two machines used in this algorithm.

Another Algorithm that was similar for usage was the Flow Shop Scheduling model of problem, which is a Greedy approach to this problem. but this too had its drawbacks as although it had facilities for Assembly Line Transfers, it was too flexible with its time slots as they were fluidly manipulated ac cross the assembly line keeping very little conditions as constants. moreover, trying to inculcate this problem forcibly on to the Flow Shop Scheduling Algorithm would've resulted in higher complexity.

Basic Brute Force methods were also tried, such as checking the immediate next pathways at every junction and taking a decision. But this usually lead to misleading paths and non optimal solutions, as finding a better path was too vague to be just sidetracked by a simple look 1 step ahead methodology.

Chapter 7

Results and Discussion

Lets take an example:

A: 2, 8, 2, 1, 1, 1.

B: 3, 2, 1, 8, 8, 8.

Table 7.1: Processing Power Table - ORIGINAL

_	**1**	**2**	**3**	**4**	**5**	**6**
A	2	8	2	1	1	1
B	3	2	1	8	8	8

Now after the Loop starts from i = 2, the X Y Tabulation is updated as.....

1. i = 2

Table 7.2: Processing Power Dynamic Tabulation - 1

_	**1**	**2**	**3**	**4**	**5**	**6**
X	2	10	2	1	1	1
Y	3	5	1	8	8	8

2. i = 3

3. i = 4

4. i = 5

5. i = 6

Table 7.3: Processing Power Dynamic Tabulation - 2

_	**1**	**2**	**3**	**4**	**5**	**6**
X	2	10	12	1	1	1
Y	3	5	6	8	8	8

Table 7.4: Processing Power Dynamic Tabulation - 3

_	**1**	**2**	**3**	**4**	**5**	**6**
X	2	10	12	13	1	1
Y	3	5	6	18	8	8

Table 7.5: Processing Power Dynamic Tabulation - 4

_	**1**	**2**	**3**	**4**	**5**	**6**
X	2	10	12	13	14	1
Y	3	5	6	18	26	8

Table 7.6: Processing Power Dynamic Tabulation - 5

_	**1**	**2**	**3**	**4**	**5**	**6**
X	2	10	12	13	14	19
Y	3	5	6	18	26	34

Clearly from the above Tabulation Transitions, we can see that after termination, X[n] = 19, Y[n] = 34. therefore the optimal solution for Assembly Line Scheduling for Processing Power Optimization Problem is: max(19, 34) = **34**.

If we would have taken the Naive Greedy method of looking forward two steps, we would have ended up with optimal sequence: B B B B B B (No Machine transfer). Then the optimal Solution obtained would've been 31, which is lesser then 34 which was obtained using Assembly Line Scheduling Dynamic Programming Approach.

Chapter 8

Drawbacks

Although this approach is seemingly the best option there is to solve this type of problem, it doesn't come free of drawbacks:

1. May show anomalies if Forward Propagation is not done also, as in many rare cases, different results are possible.

2. In some cases when printing the Schedule along with finding optimal solution, representing the letters A and B becomes a tedious job as the algorithm works in a bizarre way that makes it difficult for you to tabulate the Schedule along with the optimal pathways.

3. Works on simple iterations, hence is susceptible to large data types and larger ranges of data.

4. Compromises the trade-off on Space Complexity as a separate Space has to be kept for tabulating in order to avoid overlapping. Hence Time Complexity is traded off to Space Complexity.

Chapter 9

GITHUB

Please refer to the following GITHUB LINK for the repository based on Processing Power Assembly Line Scheduling Using Tabulation Dynamic Programming Algorithm.

The Repository has the main Solution Code ProcessingPower-LineScheduling.cpp along with a Flow Shop Scheduling effort that was not very fruitful. I/O has been set to randomize but the type of inputs can be customized by user.

GITHUB: https://github.com/SwagatSBhuyan/Processing-Power-Assembly-Line-Scheduling-Using-Tabulation-Dynamic-Programming-Algorithm