Problem set 5

by Maksim Al Dandan

April 21, 2024

A company is running Big Data ™ Analysis using cloud computing infrastructure. It has three big analysis tasks at the moment and five computational cores available to allocate among these tasks. Therefore, the company managers need to determine how many cores (if any) to allocate to each of these regions to minimize the total cost of the analysis (measured in CPU hours). Each core can only be dedicated to one task, so the number of cores allocated to each task must be an integer. The following table gives the estimated cost for each task for each possible allocation of cores.

1 Task 1

1.1 Statement

Describe a general algorithm for any number C of available cores, any number A of analysis tasks and table E with estimates:

(a) Summarize the idea for a naïve recursive algorithm.

(b) Clearly identify overlapping subproblems (provide an explicit example).

(c) Write down pseudocode for the dynamic programming algorithm that solves the problem (top-down or bottom-up). It is enough to compute the minimum cost without keeping track of the computing core allocation.

1.2 Solution

(a) The naive recursive algorithm would try all possible combinations of core allocations for each task. It starts with the maximum number of cores for the first task and recursively calls itself for the remaining tasks and cores. It then reduces the number of cores for the first task and repeats the process until it has tried all combinations. The minimum cost is kept track of during this process.

(b) Overlapping subproblems occur when the algorithm computes the cost for the same number of remaining tasks and cores more than once. For example, if we first try 2 cores for task A and 3 cores for task B, we would then compute the cost for 0 remaining tasks and 0 remaining cores. Later, we might try 3 cores for task A and 2 cores for task B, and again compute the cost for 0 remaining tasks and 0 remaining cores.

(c) Pseudocode for the dynamic programming algorithm:

```
function MinCost(C, A, E)
    dp = create a table with dimensions (C+1) x (A+1)
    for i from 0 to C
        dp[i][0] = 0
    for j from 0 to A
        dp[0][j] = 0
    for i from 1 to C
        for j from 1 to A
            minCost = infinity
            for k from 0 to i
                cost = E[k][j] + dp[i - k][j - 1]
                if cost < minCost
                    minCost = cost
            dp[i][j] = minCost
```

```
return dp[C][A]
```

2 Task 2

2.1 Statement

Provide asymptotic worst-case time complexity with justification
(a) for the naïve recursive algorithm
(b) for the dynamic programming algorithm

2.2 Solution

(a) The naive recursive algorithm tries all possible combinations of core allocations for each task. It starts with the maximum number of cores for the first task and recursively calls itself for the remaining tasks and cores. It then reduces the number of cores for the first task and repeats the process until it has tried all combinations. The minimum cost is kept track of during this process. The time complexity of this algorithm is $O(C^A)$, where C is the number of available cores and A is the number of analysis tasks. This is because for each task, the algorithm tries all possible combinations of cores, and there are C possible combinations for each task.

(b) The dynamic programming algorithm computes the minimum cost for each combination of tasks and cores by iterating over all tasks and cores and all possible allocations of cores for each task. The time complexity of this algorithm is $O(C^2 \cdot A)$, where C is the number of available cores and A is the number of analysis tasks. This is because for each combination of task and core, the algorithm iterates over all possible allocations of cores, and there are C possible allocations. Therefore, the total number of operations is proportional to the number of tasks times the square of the number of cores.

3 Task 3

3.1 Statement

Apply the dynamic programming algorithm to an instance of the problem below. You must provide the table with solutions for subproblems that are computed in the algorithm, as well as give the final answer to the problem.

Number of cores	Task A	Task B	Task C
0	∞	∞	∞
1	12.0CPU hours	13.0CPU hours	15.0CPU hours
2	10.5CPU hours	10.0CPU hours	11.0CPU hours
3	9.0CPU hours	8.0CPU hours	7.5CPU hours
4	7.0CPU hours	7.0CPU hours	4.5CPU hours
5	4.5CPU hours	5.0CPU hours	2.0CPU hours

3.2 Solution

First, we initialize a table dp with dimensions $(C + 1) \times (A + 1)$, where C is the number of cores and A is the number of tasks. We fill the table with infinity, except for the first row and column which are filled with 0.

Then, we iterate over the tasks (from 1 to A) and for each task, we iterate over the cores (from 1 to C). For each combination of task and core, we calculate the minimum cost by iterating over all possible allocations of cores for the current task (from 0 to the current number of cores). The cost for each allocation is the cost of the current task with the current allocation plus the cost for the remaining tasks and cores, which we get from the dp table. We store the minimum cost in the dp table.

Finally, the minimum cost for all tasks and cores is in dp[C][A].

The table with solutions for subproblems would look like this:

Cores/Tasks	Task A	Task B	Task C
0	0	0	0
1	12.0	23.0	38.0
2	10.5	20.5	31.5
3	9.0	17.0	24.5
4	7.0	14.0	18.5
5	4.5	11.5	11.5

The final answer to the problem is 11.5 CPU hours.