Algorithms and Data Structures - Assignment 3

Bottom-up dynamic programming approach

To explain the bottom-up dynamic programming approach we first quickly repeat the constraints that we know from the assignment text.
- The water bags and the drones must be used in order
- One cannot use multiple drones at the same time since the drones are operated by one person - One cannot use a drone twice on one day since it need to be recharged
- No more than a specific liter budget may be used per day
- Process of transportation/emptying cannot be split across two days

Using these constraints we can construct a bottom-up strategy to fill a 2D-array, called **optimal_cost**, which is indexed by drones on the rows and bags on the columns. We can start with filling the first top-left element with 0 and build up the solutions column-wise.
When we fill the next possible entry in *optimal_cost*, we calculate all possibilities to split up the bags between all drones. So for the third bag of the first drone, we would compare the costs of not splitting the costs at all and transporting all bags with one drone, splitting it after the first bag and transporting two succeeding bags with a new drone and possibly new day. The third option is to split it after the second bag and transport solely the last bag with a new drone.

The first possibility to transport all bags can be done using equation 1 in the Assignment text, thus to sum the **usage_cost** and the **idle_cost** for this bag. For the last two bags though, you need to add the *optimal_cost* for the previous bag to the *usage cost* and *idle_cost*, since the total optimal cost is the sum of all partial optimal costs. Here we use the bottom-up strategy. The overlapping subproblem in this case is the *optimal_cost* of the previous bag. Since we started filling the *optimal_cost* bottom up starting with the first bag and drone and using constraint 1 that the bags and drones must be used in order, we can always use the *optimal_cost* of the previous bag when we split the drones/days.

Recurrence Equation

Simple case with one drone

To implement the logic described in the previous section into code we define a recurrence equation of transporting water bags [0:i]. For simplicity we begin with the easy case, thus with only one drone. We use the simplifications to be able to properly write down the equation:

- $usage\_cost(x, y) = compute\_sequence\_usage\_cost(x, y, d)$
- $idle\_cost(x, y) = compute\_idle\_cost(0, i, idle\_time\_in\_liters)$

The recurrence equation for transporting water bags [0:i] for the constant drone d is:

$$\begin{aligned} optimal\_cost\_simple(i, d) = \min(&optimal\_cost[0, d] + usage\_cost(0, i) + idle\_cost(0, i),\\ &optimal\_cost[1, d] + usage\_cost(1, i) + idle\_cost(1, i), ...,\\ &optimal\_cost[i-1, d] + usage\_cost(i-1, i) + idle\_cost(i-1, i)) \end{aligned} \tag{1}$$

Next we will look at the more complicated case with multiple drones.

Difficult case with multiple drones

For the case with multiple drones we have to additionally consider the multiple drones. Firstly, we can calculate the optimal costs for each drone similarly as for the simplified sub-problem. The only difference is that it can be that performing the same transportation with another drone might be more efficient. To implement this, we compare the entry calculated with equation 1 with the entry calculated with the previous drone. Thus we get

$$optimal\_cost(i, d) = \min(optimal\_cost[i, d-1], optimal\_cost\_simple(i, d)) \tag{2}$$

for the difficult case with multiple drones.

Time complexity

Determining N as the number of bags (*num_bags*) and M as the number of drones (*num_drones*).

Then, the nested loops in the *dynamic_programming* method iterate over the bags twice (l. 175, 177) and drones (l.174), resulting in a time complexity of $\Theta(3*N^2*M*3)$. This holds for O-notation and Ω-notation because the upper and lower bound for time complexity are the same. The loop runs over all drones and bags and then again over the loop for the candidate solutions with three subfunction of complexity N. Furthermore, considering O-notation the loop will only calculate all bags twice and drones, but not more than that. Hence, one can deduct $\Omega(3*N^2*M*2) \leq \Theta(3*N^2*M*k) \leq O(3*N^2*M*3)$. Hence, $2 \leq k \leq 3$. Within each iteration of the nested loops, the functions *compute_sequence_usage_cost*, *compute_idle_cost*, and *compute_sequence_idle_time_in_liters* are called with a time complexity of N.

Investigating in detail the dynamic function one can see what happens and how often what happens inside the inner loop. It is useful to calculate the computation time in a cost function: $T(n, m)$.

Code line number	cost	repetition (O)	repetition (Ω)
190	c_1	$3*N^2*M$	$3*N^2*M$
193	c_2	M	M
196f	c_3	M	M
200f	c_4	M	0

Table 1: The time complexity in terms of a cost function:
$T(N, M) = c_1*(3*N^2*M) + c_2*M + c_3*M + c_4*M$

From T(N,M) (big O)-notation we can conclude:

$$\lim_{N,M\to\infty} \frac{T(N, M)}{f(n^2, m)} = \frac{c_1*(3*N^2*M) + c_2*M + c_3*M + c_4*M}{N^2*M} = 3_{cm} + 3_{cm}$$

Thus, this is in line with the initial evaluation: $\Omega(3*N^2*M*3) \leq \Theta(3*N^2*M*k) \leq O(3*N^2*M*2)$ where $2 \leq k \leq 3$.

Space complexity

Likewise to time complexity the primary data structures are again relying on the variable number size of the bags and the drones respectively N and M. The primary space-consuming data structures are *usage_cost*, *idle_cost*, and *optimal_cost*. There are also other data structures for example *liter_cost_per_km*, *backtrace_solution* and *liter_budget_per_day* and all other simple parameters listed in the *__init__* function that are holding proportional memory space. Those other data structure hold memory space proportional to the input O(N).

All the three main space-consuming data structures are saving information about the drones and the bags. The *usage_cost* saves exactly all the cost combinations of bags and drones exclusively. Hence, the space complexity is $O(N*M)$ because there are at most all possible combinations of bags and drones. Also, the lower bound is $\Omega(N*M)$ (The same argument holds for the linear parameters) as no drone can be neglected for the usage cost as every drone-bag combination might occur. The same holds for *optimal_cost* and *idle_cost*. The above is visualized in the following table with the specific cost function of the space complexity

Code line number	cost	repetition (O)	repetition (Ω)
23-27	c_1	N	N
28	c_2	$N*M$	$N*M$
40	c_3	$N*M$	$N*M$
44	c_4	$N*M$	$N*M$
48	c_5	N	N

Table 2: The time complexity in terms of a cost function:
$S(N, M) = c_1*(N) + c_2*(N*M) + c_3*(N*M) + c_4*(N*M) + c_5*(N)$

From S(N,M) (big O)-notation we can conclude:

$$\lim_{N,M\to\infty} \frac{S(N,M)}{f(n,m)} = \frac{c_1*(N)+c_2*(N*M)+c_3*(N*M)+c_4*(N*M)+c_5*(N)}{N*M} = 3_c n, m$$

Therefore, the overall space complexity of the method is $\Omega(3*N*M) = \Theta(k*N*M) = O(3*N*M)$ where $3 \leq k \leq 3 \rightarrow k = 3$.

Backtracing solution

With the function dynamic_programming, we filled the 2D optimal_cost datastructure and got the minimal value using the function lowest_cost. We want to deduce an optimal schedule to use the drones for every water bag from the 2D memory. For that we have to figure out which bags are transported on which day and which drone is used for which bag. We can figure out which bags are used on which day only by using the last column of the optimal_cost datastructure. One can do so, since the algorithm compares the value that is calculated with the current drone with the one calculated with the previous drone. The optimal_cost decreases within the column until the next day since the idle_cost (which is significant for the optimal_cost because of its power 3 impact) decreases with more bags being used on one day. Thus one can search within the last column of the 2D memory for the entries, which have an increase and no decrease. We store these entries as the left-most entries of one day.

We did one minor change to the code in dynamic_programming to also figure out which bag is transported by which bag. We do that using a global array backtrace_memory of length bags. We have a case distinction between if the calculated cost is smaller than the calculated cost to transport the same bag of the previous drone or not. If that is the case, we store the optimal_cost value of the previous drone for the new drone as well. If not we actually store the calculated cost as the optimal_cost for this bag. The change to the code in order to find out which drones are used takes place here. We also store the drone for the bag in the array backtrace_memory. Thus, only when the optimal_cost is not filled by a previous drone (which would already have filled the array at this bag position), the array at the bag position would be written with the drone.

Summary and Discussion

Goals

In the assignment, we aimed to find a bottom-up dynamic programming solution for calculating the optimal cost for transporting water bags with drones in order to extinguish a fire. We should learn how to construct a data structure with purpose to store the entries of previous calculations in order to reuse them for further entries and to minimize a cost function. The function should be analyzed in terms of time and space complexity and one should write a backtracing algorithm that deduces the optimal schedule and which drones are used for every water bag.

Results

We developed an optimization function that uses a 2D memory object to build up the datastructure from the bottom and reuse its entries to save runtime. We found that:

- The algorithm for the costs can be described by the recurrence equations described in equation 1 and 2
- The algorithm has a time and space complexity of $O(3*N^2*M*2)$ and $O(N*M)$ respectively
- The backtracing algorithm can derive the daily schedule regarding the optimal cost

Contribution

- David Wünsch (s3665534) & Jesse Kroll (s3666778):
  - Contribution to code (both)
  - Bottom-up dynamic programming approach (D)
  - Recurrence Equations (D)
  - Time and Space complexity (J)
  - Summary and Discussion (both)