Algorithms and Data Structures - Assignment 3

Bottom-up approach explanation

In our bottom-up approach, we are filling up the table, in which columns represent drones and rows represent bags. So, for every row i and column j we will have the optimal total cost of spending some amount of days carrying bags and ending the last day having used the first i bags and the first j drones. However, it's important to mention, that all the rows besides the bottom one will have to be optimized by the total cost which includes the idle time loss on the last day as well, since the rows below will need to factor that in. The optimal cost that we actually need to find will be the minimal number of the last row, since we have to use all the bags. Thus, we are using slightly different formulas to calculate the last row and all the others. To explain the algorithm in simple terms imagine that we are filling the cell $[i, j]$ and we know all the previous costs with the final day loss. Also, we have two categories of days: the last one and all the others. And now the question is how many bags we will drop on the last day: we may carry only the bag i alone on the last day, or the bag i with previous $k-1$ bags, where k is the maximal number of bags we could carry that day in that situation (it depends on the transportation costs of the bags). So, for instance, if we have 5 bags with transportation costs (travelling both directions and emptying the bag) [1, 15, 4, 5, 11] and the limit per day is 20, then during the consideration of i=4 we will have the following scenarios: take bag 4 alone (9 liters left), take bag 4 and 3 (4 liters left), or take bags 4, 3, and 2. We may not take bags from 4 to 1 as it will exceed the permitted limit. So, our number k is 2 and in our table, we will consider rows from 2 to 4. As for the columns, we will consider columns from 0 to j, as we may use drones only in ascending order.
So we start to step through the possible options described in the example. Firstly, we consider when we are taking only bag 4 on the last day. In that case, to calculate the cost, which excludes last day idle time loss, we are adding the cost of using drone j to transfer that bag to the minimal cost we will find in the row $i-1$ (up to j). To find minimal value with the final day idle loss we just add a cubed difference between the limit and the time of work on the last day to the 'normal' cost. Record this value as one of the possible options. When we are taking bags 4 and 3, to find the cost we add the cost of transporting these two bags with the drone j to the minimal cost in the row i-2 (as we are carrying two bags). Afterwards, find the cost with the last day's idle time loss. Record these values as well and repeat the steps until you check all the possible options for this cell. Important to note, that we are also recording the coordinates of the previous bags, with witch we obtain the minimal loss. After that, if you are not on the last cell, pick the option, where the cost with the final day is minimal. Otherwise, pick the one with the minimal cost without the last day.
Finally, we will have the filled table, in which the last row is filled with the tuples, in which the 'normal' cost is optimal. In all other rows the tuples with the cost, which includes the last day idle time loss, will be presented. To find the minimal 'normal' cost we simply may look for the minimal one among the tuples in the last row.

Recurrence equation for 1 drone

The recurrence equation for 1 drone is following:

$$Optimal\_cost(i) = \begin{cases} usage\_cost[0,0] & i = 0 \\ \min \begin{cases} usage\_cost[i,0] + ocwld[i-1] \\ usage\_cost[i,0] + usage\_cost[i-1,0] + ocwld[i-2] \\ \dots \\ usage\_cost[i,0] + \dots + usage\_cost[i-m,0] + ocwld[i-m] \end{cases} & i > 0 \end{cases}$$

Where $usage\_cost$ is the table, in which $usage\_cost[i][j]$ is the cost of using drone j for bag i. $travel$ is the list, in which $travel[i]$ is the full cost of travelling to the bag and back and releasing the water. $limit$ is the max amount of liters we can use per day. And $ocwld[i]$ is a shortcut for $optimal\_cost\_with\_the\_last\_day[i]$ - the optimal cost for bringing i bags if we count the idle time loss on the last day as well.
That is the formula for the last row of our table (in the program we also use it for the 0 row). The formula for the cost, including the loss of the idle time on the last day, is almost the same, but we add the idle time loss after the $usage\_cost$:

$$ocwld(i) = \begin{cases} usage\_cost[0,0] + (limit - travel[i])^3 & i = 0 \\ \min \begin{cases} usage\_cost[i,0] + (limit - travel[i])^3 + ocwld[i-1] \\ usage\_cost[i,0] + usage\_cost[i-1,0] + (limit - travel[i] - travel[i-1])^3 + ocwld[i-2] \\ \dots \\ usage\_cost[i,0] + \dots + usage\_cost[i-m,0] + (limit - travel[i] - \dots - travel[i-m])^3 + ocwld[i-m] \end{cases} & i > 0 \end{cases}$$

The logic behind these two formulas is simple: if we have one bag, we take the cost from the *usage_cost* directly. Otherwise, we check for all possible distributions of bags between the last day and all others and take the one with the minimal cost (with the last day or without), as it was discussed in the previous section.

Recurrence equation for many drones

For many drones, we have (notation is the same):

$$Optimal\_cost(i,k) = \begin{cases} \min \begin{cases} usage\_cost[0,0] \\ \dots \\ usage\_cost[0,k] \end{cases} & \text{i} = 0 \\ \min \begin{cases} usage\_cost[i,k] + min(ocwld[i-1,0] + \dots + ocwld[i-1,k]) \\ usage\_cost[i,k] + usage\_cost[i-1,k] + min(ocwld[i-2,0] + \dots + ocwld[i-2,k]) \\ \dots \\ usage\_cost[i,k] + \dots + usage\_cost[i-m,k] + min(ocwld[i-m,0] + \dots + ocwld[i-m,k]) \end{cases} & i > 0 \end{cases}$$

$$ocwld(i,k) = \begin{cases} \min \begin{cases} usage\_cost[0,0] + (limit - travel[i])^3 \\ \dots \\ usage\_cost[0,k] + (limit - travel[i])^3 \end{cases} & \text{i} = 0 \\ \min \begin{cases} usage\_cost[i,k] + (limit - travel[i])^3 + min(ocwld[i-1,0] + \dots + ocwld[i-1,k]) \\ usage\_cost[i,k] + usage\_cost[i-1,k] + (limit - travel[i] - travel[i-1])^3 + min(ocwld[i-2,0] + \dots + ocwld[i-2,k]) \\ \dots \\ usage\_cost[i,k] + \dots + usage\_cost[i-m,k] + (limit - travel[i] - \dots - travel[i-m])^3 + min(ocwld[i-m,0] + \dots + ocwld[i-m,k]) \end{cases} & i > 0 \end{cases}$$

So the logic here is the same, except the fact, that we have to take min twice. That's because when we had one drone, it was obvious what we have for $optimal\_cost\_with\_the\_last\_day[i-m]$. Now we have multiple drones, so we have to calculate $optimal\_cost\_with\_the\_last\_day[i-m]$ for every drone from 0 to k and take the minimum from that. All the rest stays the same.

Time complexity

For the time complexity, we will assume that all preparatory functions have been done and we are executing only the dynamic_programming function. For the notation, we are using i for the number of bags, j for the number of drones, and l for the maximal amount of liters per day.
Now we have to fill up the table with the shape of $i \times j$. For every cell, we have to compute the number of rows we may step back. As we have integer numbers, the maximal number of rows we are considering is simply the maximal amount of liters per day, l. And for every row, we are looking for the minimal tuple, so we are traversing through the row from the 0 column to the column we are on now (k). Maximally, we may traverse j columns, as that's the width of the table. Finally, we have the formula for the big-$\mathcal{O}$ complexity:

$$\mathcal{O}(Optimal\_cost(i,j,l)) = i * j * l * j = i * j^2 * l \tag{1}$$

As we are getting fewer drones, l will be approaching i, if we have a significant limit. So in that case, for the higher (or equal) order of growth, we may write:

$$\Omega(Optimal\_cost(i,j,l)) = i * j * i * j = (i * j)^2 \tag{2}$$

Finally, the function with lower (or equal) order of growth will be the same as big-$\mathcal{O}$:

$$\Theta(Optimal\_cost(i,j,l)) = i * j * l * j = i * j^2 * l \tag{3}$$

Space complexity

To calculate the space complexity of the method we will use the same notation as in the previous section. We have to fill the table of tuples, which contains 2 numbers and a tuple with 2 numbers. Moreover, for every cell, we are creating a list with possible options, max length of which is i and it is filled with 2 values and a tuple with 2 values. Finally, we have the following equation:

$$\Theta(Optimal\_cost(i,j,l)) = i*j*4 + i*4 \Rightarrow \Theta(Optimal\_cost(i,j,l)) = i*j \tag{4}$$

If we are to calculate the space complexity of the solution (class) overall, it will have the following form:

$$\Theta(Optimal\_cost(i,j,l)) = i*j*4 + i*4 + i + 2*i + i*j \Rightarrow \Theta(Optimal\_cost(i,j,l)) = i*j \tag{5}$$

The two additional i's and $i*j$ are the space of the lists of weights of bags, their locations and the table $usage\_cost$ ($usage\_cost[n][m]$ is a cost if using drone m for bag n).

Optimal schedule and backtracing

In the first section (where we explained the algorithm) we mentioned, that along with the number we also record the coordinate in the table of the last bag used on the previous day. It was done with the purpose of making backtracing easier. To find the schedule of the drones and bags we first have to find the last drone and bag we used, which is exactly what is in the tuple in the last row with the minimal 'normal' cost. The 3rd object in this tuple is the coordinate of the last bag on the previous day. And in the tuple of this last bag, its 3rd object is the coordinate of the last bag on the day before it. So this way we go way back until the y-coordinate in the tuple becomes -1. When it does so, it means, that we are on the last bag of the first day. We may end our traversing. After that, we reverse the recorded list, as we went from the back, and add the last bag of the last day (the one we found in the beginning). Subsequently, we transform the obtained schedule into the format we need to output. We travel our schedule day by day (so going through every tuple in the list) and appending the numbers to the needed lists. On the first tuple we simply append 0 to the sequence of the 1st bags carried every day as we have no other option. On the other tuples, we append the number in the previous tuple + 1 (so the bag we end the previous day + 1). As for the list of the sequence of drones, we take the drone number from the tuple and append it to the list n times, where n is the difference between the numbers of the last drone we use on the i-th day and on the day before that. For the first tuple, we simply append the number of times equal to the number of the last drone on the first day - 1 (- 1 because we start naming drones from 0).
This way our backtracing algorithm has a linear time of completion. The number of iterations we have to make does not exceed the number of bags we have. Such complexity is justified by the architecture of the table that we fill up using dynamic programming.

Summary and Discussion

The goal of the assignment was to optimize the schedule of drones and bags they are carrying to obtain the minimal cost function. Firstly, it was important to note, that the loss of the idle time function and the cost of usage function are not independent. Therefore, it's not optimal to just optimize the schedule of bags for every day, and simply assign the drones to days afterwards. In other words, the dynamic programming algorithm has to take the cost function overall to calculate the solution, but not its parts separately. Also, from this observation, it also follows, that it is not optimal to put as many bags on a day as we can carry (so use a greedy approach to optimize a bags' schedule). Afterwards, another important concept to introduce was the division of the days into 2 categories: the last one and all the others. Basically, this is the key idea of our bottom-up approach. Lastly, the idea of recording the coordinates of the bag-drone pair, used at the end of the previous day, is the basis for implementing backtracing. With that backtracing algorithms has a linear complexity.
Overall, we developed an algorithm with the time complexity of $\mathcal{O}(n^4)$, and with the minimum quadratic space complexity $\Omega(n^2)$.

Contributions

Maksym Lytovka (s3705609): Code, Report
Ivan Banny (s3647951): Code, Report