Programming Assignment 3 — Dynamic programming

There is a large forest fire (see Figure 1) that we want to extinguish in the most efficient way. For this, we use drones that can pick up water bags spread across different locations, and empty them in the forest. Our goal is to devise an optimal schedule for these drones so that the forest damage is minimal.

Figure 1: The forest fire.

Description

The problem is illustrated in Figure 2. As can be seen in the left part of the figure, we have a sequence of n water bags of different contents $\mathbf{w} = [w_0, w_1, \ldots, w_{n-1}]$, where w_i is the amount of water inside of bag i, measured in liters. We want to use these water bags to extinguish the forest fire. Importantly, the water bags should be used in order, that is, first we use water bag 0, then water bag 1, and so on. Our drone housing company is very close to the location of the forest (x_f, y_f) so the travel distance is negligible. The water bags all have (possibly different) locations $[(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})]$. Our goal is to minimize the forest damage as measured by a cost function that we will define below. **Importantly, throughout the description, all costs will be expressed in liters, including the travel cost.** Travel costs are transformed to liters as follows. Suppose the distance to a water bag w_i from the forest (or drone housing company) is d (measured in kilometers). Then, the usage cost $\ell_{trans}(w_i) = d \cdot p$, where p is the liter cost per kilometer (`liter_cost_per_km` in the code).

We use drones to pick up the water bags and empty them above the forest to extinguish the fire. We have a set of m drones, i.e., $\mathbf{d} = [d_0, d_1, \ldots, d_{m-1}]$. For every (water bag, drone) pair (w_i, d_k), we have a usage cost $c(i, k)$ that is influenced by the type of drone, the weight of the water bag, and other factors. This information is given by the problem. Importantly, the drones should also be used in order, and it is not allowed to use multiple drones on a single day. That is, a drone with index k cannot be used after using a drone with an index $m > k$. For example, it is **not** allowed to use drone 1 for water bag 0 and afterward drone 0 for water bag 1, because drone 0 cannot be used after using any other drone.

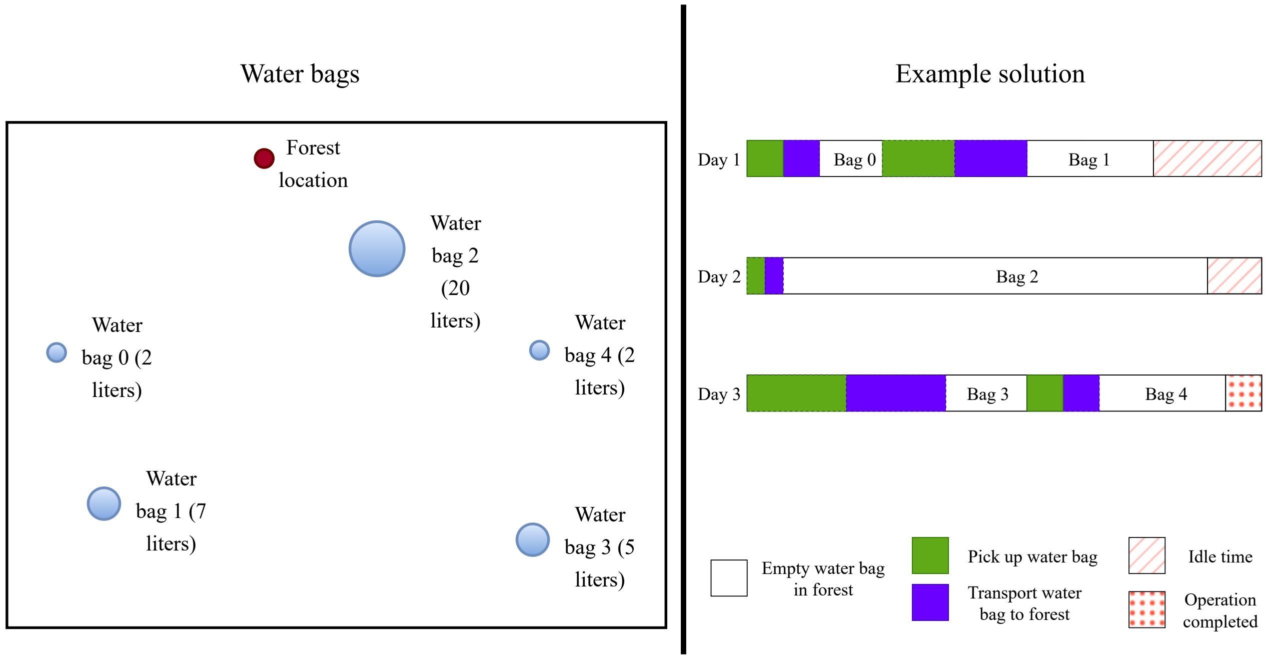

Figure 2: Example of a forest fire problem. Note that the travel distance to pick up a water bag is equivalent to the distance to travel back to the forest.

As mentioned before, the goal is to find a water bag transportation schedule such that the damage to the forest caused by the fire is minimal. An important constraint here is that per day, we cannot do more work (transporting and emptying water bags) than X liters (`liter_budget_per_day` in the code). This "work" includes the sum of the contents of the water bags transported on the day plus the cost of transportation ℓ_{trans} (also measured in liters)—note that this cost is **not** the usage cost. An example of such a solution is shown on the right in Figure 2. Here, we transport water bags 0 and 1 on day 1, water bag 2 on day 2, and the remaining bags (3 and 4) on day 3.

We want to empty all water bags above the forest location whilst minimizing a cost function based on the usage cost of the drones and the idle time: time not spent on transporting water or emptying a bag in the forest. An important constraint here is that the process of transportation/emptying a bag cannot be split across two days. Thus, every water bag should be transported and emptied within the same day. Note that drones cannot be used in parallel (at the same time) because they are operated in real-time by a single person (Larry); Larry has no other colleagues due to staff cuts.

Candidate solution representation A candidate solution can be represented as a tuple (water bag list, drone list). Assuming that the candidate solution has a time span of T days, the water bag list $s = [idx(1), idx(2), ..., idx(T)]$ is a list where $idx(t)$ is the index of the water bag that is the first to be transported on day t. The water bag list in the example solution (right plot in Figure 2) could be represented as [0,2,3] because on day 1, we start with water bag 0, on day 2 with water bag 2, and on day 3 with bag 3. In addition to the water bag list, we also need to indicate what drones are used to transport the bags. This is the drone list, $d = [d(0), d(1), \ldots, d(n-1)]$ where $d(j)$ is the drone index of the drone used to transport water bag j (w_j). Assuming that water bags 0, 1, and 2 are transported with drone 0, bag 3 with drone 1, and bag 4 with drone 2, the drone list would be $d = [0, 0, 0, 1, 2]$.

The cost function The cost formula that we aim to minimize is the following. Suppose that a candidate solution transports water bags for a total of 3 days as in the example solution in Figure 2. Let $idle(t)$ be the idle time on day t for $t \in \{1, 2, 3\}$ for a valid candidate solution. Then, the cost formula that we aim to minimize is given by

$$Cost = \underbrace{\sum_{i=1}^{n} c(i, d(i))}_{\text{usage cost}} + \underbrace{\sum_{t=1}^{T-1} idle(t)^3}_{\text{idle cost}} . \tag{1}$$

The first term is the sum of the costs of using drone $d_{d(i)}$ to transport bag w_i, that is, $c(i, d(i))$ (`usage_cost` in the code), measured in liters. Note that this is independent of the distances to the bags: this is purely the cost of operating the drone. The second term is the sum of the idle cost of all days, **except for the final day** (as the mission is then completed and the remainder of the day cannot be interpreted as idle time). The idle cost is computed as the difference between the length of a day, X (measured in the amount of liters that can be maximally transported and emptied per day) and the time (measured in liters) spent transporting and emptying water bags. Note that negative idle times are not allowed (we cannot start the transportation on one day and finish it on the next day). Note that in this example, the solution consisted of 3 days. Supposing that the budget per day is 25 liters, the idle time on day 1 is 25-2(liters in water bag 0) - 3 (travel time to bag 0 and back to the forest) - 7 (liters in water bag 1) - 5 (travel time to bag 1 and back to the forest) = 8. Thus, the idle cost of day 1 would be $8^3 = 512$. Here, we assumed that the travel times for water bags 0 and 1 are 3 and 5 liters respectively; these numbers are made up and could not be inferred by the figure alone. In general, a solution consists of as many days as required.

Programming

Your task is to devise a *bottom-up* dynamic programming strategy for creating a transportation schedule that minimizes the cost function as shown in Equation 1. To implement the dynamic programming algorithm for the problem described above, you will implement a class called `FireExtinguisher`. Since this is a relatively challenging task, we have broken it down into smaller manageable pieces. First, we recommend you to implement the utility functions (everything except dynamic_programming, lowest_cost, and backtrace_solution). After that, it is crucial to come up with a recurrent equation that will translate itself into a dynamic programming strategy. We highly recommend you do this in the following two steps.

Step 1: recurrent equation for a single drone

To begin with, assume that we have a simplified version of the problem, where we only have a single drone that should transport all the water bags, minimizing the cost. Try to identify the recurrent structure in the problem: suppose you have the optimal cost for placing water bags[0:i] ($\{w_0, \ldots, w_{i-1}\}$), that is $Optimal\_cost(i)$ (and all preceding optimal costs ($Optimal\_cost(0)$, $Optimal\_cost(1)$, ..., $Optimal\_cost(i-1)$). Then, how can we reuse these quantities to derive the optimal cost for water bags[0:i+1] ($\{w_0, \ldots, w_i\}$), that is, $Optimal\_cost(i+1)$? In other words, think of a formula that computes $Optimal\_cost(i+1)$ based on $Optimal\_cost(j)$ for $j < i+1$. Also, think about the base case: what is the endpoint of the recursive/recurrent definition?

Step 2: recurrent equation for the entire problem

Next, think about what changes to the recurrent structure by having multiple drones at our disposal. In this case, the recurrent equation becomes a function of two variables, namely $Optimal\_cost(i, k)$. This quantity tells us the optimal cost of transporting water bags[0:i] ($\{w_0, \ldots, w_{i-1}\}$) when we have drones[0:k+1] ($\{d_0, \ldots, d_k\}$) to our disposal. Note that you are not forced to use all drones. For example, the optimal cost $Optimal\_cost(i, k)$ could be obtained by using only a subset of the k drones (taking into account the constraints).

Template program

We provided a template program. It consists of two files:

- `dynprog.py`: The main file in which you will implement the solution.
- `test_usecases.py`: A Unit Test file with various test cases that help you check whether your implementations of the different functions are correct.

The file `dynprog.py` contains various functions that are not implemented yet. Read the written documentation about these functions, and implement these functions. Most of these functions require less than 10 lines of code. **Do not change the headers of the functions provided.** You are allowed to add functions yourself if you feel that that makes it easier. Note, however, that points are deducted if we think that they are unnecessary. Make sure to document these consistently.

The file `test_usecases.py` consists of several test functions. We provided these functions to help programming and testing the code. By running the following command, the unit tests can be executed:

```
python -m unittest test_usecases.py
```

Make sure to have the right packages installed. Do not use other packages than those present in the template. Feel free to add more unit tests. Do not alter the unit tests that are provided. If your program does not succeed on all unit tests that are provided, it is likely that there is still a problem in your code. Make sure that all other unit tests succeed, before submitting the code.

Additionally, also hand in a file named `test_private.py`. In here, you can create additional unit tests to verify the working of your program. Write at least 3 additional unit tests, and hand these in along with the assignment.

Also keep in mind that all unit tests should be able to run within a matter of seconds on any computer.

Report

Write a report in LaTeX(at most 3 pages) using the provided template (see Brightspace), addressing the following points / research questions:

- Give a complete explanation of your final bottom-up dynamic programming approach in the case that we have multiple drones in words. You do not need to write down recurrence equations here, focus on conveying the intuition. Introduce relevant notation that you use throughout the report.

- Write down the recurrence equation in the case that we have a single drone (step 1). This means that you have to express the optimal cost *Optimal_cost*(i) of transporting water bags[0:i] ($\{w_0, w_1, \ldots, w_{i-1}\}$) in terms of the optimal cost *Optimal_cost*(j) for $j < i$ **and explain** what this equation means.

- Write down the recurrence equation for *Optimal_cost*(i, k) in the case where we have multiple drones at our disposal (step 2). How does the recurrent formulation change compared to the one in the previous question?

- What is the time complexity of the dynamic programming method? Do the analysis in terms of Ω, Θ, and O-notation. Make sure to introduce and define the necessary variable(s) to do this.

- What is the space complexity of the method? Use Θ-notation.

- How can we deduce the optimal schedule and which drones are used for every water bag from the 2D memory that we obtained after running our dynamic programming function? (Explain backtrace_solution in natural language—without using code in your report!)

- Summary and Discussion. What was the goal of the assignment? What have you done and observed? Do not write about your personal experience and stories. Keep it scientific and simply summarize the report, making observations about the algorithms.

Work distribution

At the end of the report, include a distribution of the work: who did what? By default, we give both group members the same grade, but if there is a serious imbalance in the workload distribution, we may have to take action. The work distribution does not count towards the page limit.

Submission

Submit your assignment through Brightspace by submitting the following files:

- report.pdf (the report)

- dynprog.py (your solution code)

- test_private.py (your unit tests)

The deadline for this assignment is **the 19th of May 2023, 23:59 CET.**