Treasure Quest: The Straw Hat Crew!

COL106 Assignment 3

September 2024

1 Background

The Straw Hat pirates, always in search of adventure and treasure, have gathered an impressive collection from their journey across the seas. As the crew's Treasurer, it is your responsibility to ensure all this treasure is properly managed. With the help of your crewmates, each member volunteers to handle the treasure, making sure that every piece of treasure is organized efficiently.

While each crewmate can work on one treasure pile at a time, they must carefully decide which to tackle first, balancing the workload and ensuring that no treasure gets forgotten. With countless treasures arriving at different times and in varying sizes, it is up to you to lead the crew in managing this task smoothly. Can you ensure the Straw Hat pirates stay organized on their journey to the One Piece?

2 Modelling

In this problem, we model the task of managing the treasure collected by the Straw Hat pirates as a scheduling problem involving multiple crewmates. The key elements of this model are defined as follows:

2.1 Definitions

- **m**: the number of crewmates available to manage the treasure.
- Each piece of treasure j is characterized by:
  - **Treasure ID**(id_j): a unique positive integer for identifying each piece of treasure.
  - **Treasure Size**($size_j$): the time (a positive integer) required to process the treasure, measured in units. For example, if $size_j = 5$, it takes 5 units of time to complete.
  - **Arrival Time**($arrival_j$): the time (a positive integer) at which the treasure becomes available for processing. For example, if $arrival_j = 7$, it can only be processed after time 7.

2.2 State of the System

At any given time t:

- Each crewmate maintains a list of treasure pieces assigned to them.
- The remaining size of a treasure piece j is given by:

  $$\text{Remaining Size} = \text{Original Size}(size_j) - \text{Processed Time}$$

  where the processed time is the amount of time already spent managing that piece. Once the remaining Size of a treasure is zero, no more time can be spent on it for processing.
- The load on a crewmate is defined as the total remaining size of the treasures in their queue.
- Note that at any given moment, **a crewmate can only process one treasure i.e. a crewmate cannot process 2 or more treasures in parallel**

2.3 Scheduling Policy

When a new piece of treasure arrives, the following scheduling policies are applied:

- **Treasure Assignment**: The newly arrived treasure is assigned to the crewmate with the least current load. This is calculated based on the total remaining size of treasures assigned to each crewmate. In case more than one crewmate has the least current load then the treasure can be assigned to any one of them.
- **Treasure Processing**: At any time t, each crewmate will process the treasure j for which:

  $$\text{Priority}(j) = (\text{Wait Time}(j) - \text{Remaining Size}(j))$$

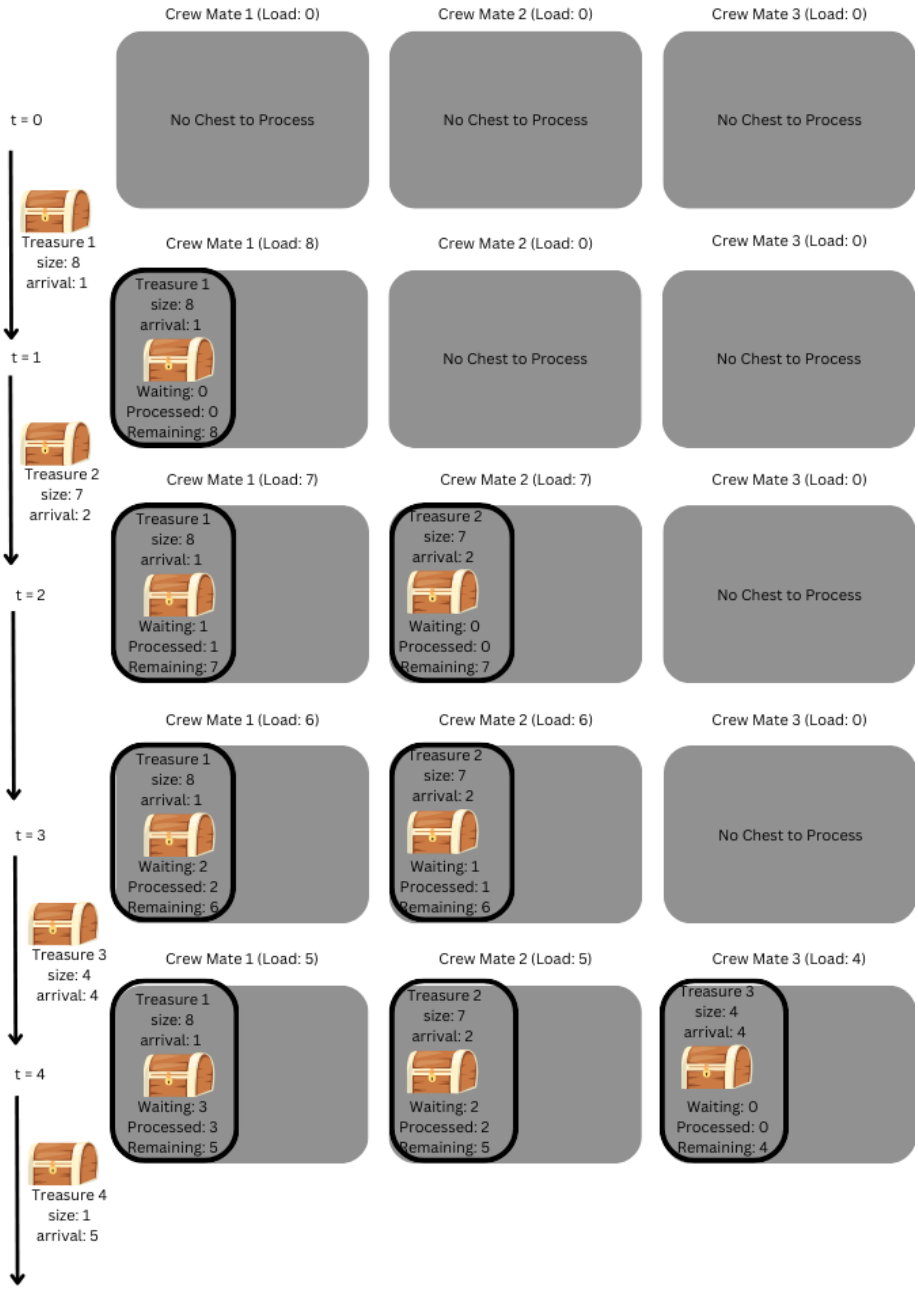
  is maximized. Here, Wait Time(j) is the total time the treasure has been waiting since its arrival time, i.e., at a time t after arrival of the treasure j,

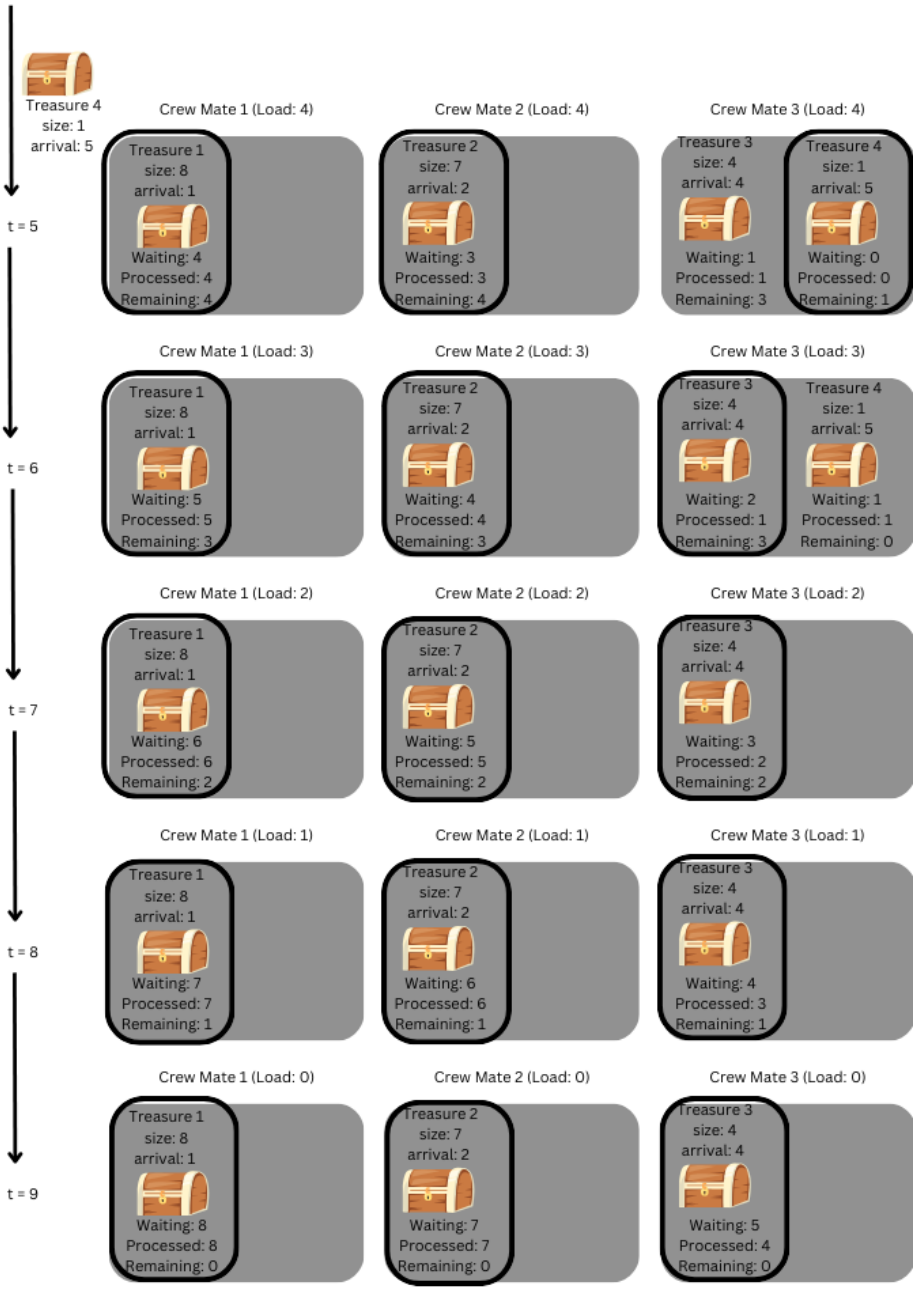
  $$\text{Wait Time}(j) = t - arrival_j$$

  . In case more than one treasure has maximum priority, then the crewmate will process the treasure with least id (having the maximum priority).

2.4 Example

Consider a scenario where there are 3 crewmates managing 4 pieces of treasure represented by a flow diagram. The black outline in the diagram represents the treasure which is being processed at the corresponding time.


Crew Mate 1 (Load: 0)
Crew Mate 2 (Load: 0)
Crew Mate 3 (Load: 0)
t = 0
No Chest to Process
No Chest to Process
No Chest to Process
Treasure 1
size: 8
arrival: 1
Crew Mate 1 (Load: 8)
Crew Mate 2 (Load: 0)
Crew Mate 3 (Load: 0)
t = 1
Treasure 1
size: 8
arrival: 1
Waiting: 0
Processed: 0
Remaining: 8
No Chest to Process
No Chest to Process
Treasure 2
size: 7
arrival: 2
Crew Mate 1 (Load: 7)
Crew Mate 2 (Load: 7)
Crew Mate 3 (Load: 0)
Treasure 1
size: 8
arrival: 1
Waiting: 1
Processed: 1
Remaining: 7
Treasure 2
size: 7
arrival: 2
Waiting: 0
Processed: 0
Remaining: 7
No Chest to Process
t = 2
Crew Mate 1 (Load: 6)
Crew Mate 2 (Load: 6)
Crew Mate 3 (Load: 0)
Treasure 1
size: 8
arrival: 1
Waiting: 2
Processed: 2
Remaining: 6
Treasure 2
size: 7
arrival: 2
Waiting: 1
Processed: 1
Remaining: 6
No Chest to Process
t = 3
Treasure 3
size: 4
arrival: 4
Crew Mate 1 (Load: 5)
Crew Mate 2 (Load: 5)
Crew Mate 3 (Load: 4)
Treasure 1
size: 8
arrival: 1
Waiting: 3
Processed: 3
Remaining: 5
Treasure 2
size: 7
arrival: 2
Waiting: 2
Processed: 2
Remaining: 5
Treasure 3
size: 4
arrival: 4
Waiting: 0
Processed: 0
Remaining: 4
t = 4
Treasure 4
size: 1
arrival: 5


Treasure 4
size: 1
arrival: 5
t = 5
t = 6
t = 7
t = 8
t = 9
Crew Mate 1 (Load: 4)
Crew Mate 2 (Load: 4)
Crew Mate 3 (Load: 4)
Treasure 1 size: 8 arrival: 1 Waiting: 4 Processed: 4 Remaining: 4
Treasure 2 size: 7 arrival: 2 Waiting: 3 Processed: 3 Remaining: 4
Treasure 3 size: 4 arrival: 4 Waiting: 1 Processed: 1 Remaining: 3
Treasure 4 size: 1 arrival: 5 Waiting: 0 Processed: 0 Remaining: 1
Crew Mate 1 (Load: 3)
Crew Mate 2 (Load: 3)
Crew Mate 3 (Load: 3)
Treasure 1 size: 8 arrival: 1 Waiting: 5 Processed: 5 Remaining: 3
Treasure 2 size: 7 arrival: 2 Waiting: 4 Processed: 4 Remaining: 3
Treasure 3 size: 4 arrival: 4 Waiting: 2 Processed: 1 Remaining: 3
Treasure 4 size: 1 arrival: 5 Waiting: 1 Processed: 1 Remaining: 0
Crew Mate 1 (Load: 2)
Crew Mate 2 (Load: 2)
Crew Mate 3 (Load: 2)
Treasure 1 size: 8 arrival: 1 Waiting: 6 Processed: 6 Remaining: 2
Treasure 2 size: 7 arrival: 2 Waiting: 5 Processed: 5 Remaining: 2
Treasure 3 size: 4 arrival: 4 Waiting: 3 Processed: 2 Remaining: 2
Crew Mate 1 (Load: 1)
Crew Mate 2 (Load: 1)
Crew Mate 3 (Load: 1)
Treasure 1 size: 8 arrival: 1 Waiting: 7 Processed: 7 Remaining: 1
Treasure 2 size: 7 arrival: 2 Waiting: 6 Processed: 6 Remaining: 1
Treasure 3 size: 4 arrival: 4 Waiting: 4 Processed: 3 Remaining: 1
Crew Mate 1 (Load: 0)
Crew Mate 2 (Load: 0)
Crew Mate 3 (Load: 0)
Treasure 1 size: 8 arrival: 1 Waiting: 8 Processed: 8 Remaining: 0
Treasure 2 size: 7 arrival: 2 Waiting: 7 Processed: 7 Remaining: 0
Treasure 3 size: 4 arrival: 4 Waiting: 5 Processed: 4 Remaining: 0

3 Requirements

Here's a description of the classes and functionalities you are expected to implement :

3.1 Heap

Refer to `heap.py` in the starter code. You are required to implement a heap from scratch that accommodates any general comparison function. The heap should be a min-heap based on the comparison function, meaning the top of the heap should always represent the minimum element according to the function's criteria. The implementation should adhere to the following specifications:

- **init:** Implement the `init` function with a time complexity of $O(n)$, where n is the size of the `init_array`. This function should initialize the heap using the provided array.
- **insert:** Implement the `insert` function with a time complexity of $O(\log n)$, where n is the number of elements currently in the heap. This function should add a new element to the heap while maintaining the heap property.
- **extract:** Implement the `extract` function with a time complexity of $O(\log n)$, where n is the number of elements currently in the heap. This function should remove and return the top element of the heap while maintaining the heap property.
- **top:** Implement the `top` function with a time complexity of $O(1)$. This function should return the top element of the heap without removing it.

3.2 StrawHatTreasury

Refer to `straw_hat.py` in the starter code. You are required to implement the class `StrawHatTreasury`. Note that m is the number of crewmates and n is the number of treasures assigned. The implementation should adhere to the following specifications:

- **init:** Implement the `init` function with a time complexity of at most $O(m)$. This function should initialize the treasury and the crewmates involved in processing the treasures.
- **add_treasure:** Implement the `add_treasure` function, which adds a treasure to the class and schedules its processing accordingly. The time complexity should be at most $O(\log n + \log m)$. You can assume that the input treasure is always valid and follows the constraints outlined in this document. The arrival times of the treasures will be in strictly increasing order. For example, if the first call to function occurs with arrival time 5, the next can only occur with arrival time 6 or later, and so on.

- **get_completion_time:** Implement the `get_completion_time` function, which returns a list of treasures objects in the increasing order of their Ids after updating the completion time in the treasure object. If multiple valid outputs are possible, any one of them can be returned. This function should be implemented with a time complexity of $O(n(\log n + \log m))$.

  Note: This method may be called multiple times during autograding. At each call, you are required to return all the Treasure objects (with updated completion time) added before the call, assuming no further treasures will be added. For example, if 10 treasures have been added using the `add_treasure` method, calling `get_completion_time` should return the list of these 10 treasures with their updated completion times, in the order of their ID, assuming no additional treasures will be added. If 5 more treasures are subsequently added and `get_completion_time` is called again, the function should return the list of all 15 Treasure objects with their updated completion times, again assuming no further treasures will be added. It's important to note that the completion times of the original 10 treasures may change between the two calls, as the 5 newly added treasures could be scheduled earlier for processing.

4 Clarifications

1. You are allowed to do cross imports between the files provided but you are not allowed to import any additional library.

2. You are not allowed to use any hash based data structures like dictionary (map), set etc. This assignment is primarily around heaps and you should not use any other structures such as tress, hash maps etc. We are expecting an array based implementation of heap. In case of any doubts, you should check with the instructors.

3. Note that the treasure size and arrival time can be arbitrarily large. To avoid timeout errors, the time complexity of your algorithm should not be dependent on these values.

4. Remember to return the output from functions instead of printing it.

5. You can assume that you will always receive valid inputs. You don't have to check for their validity.

6. You are allowed to use inbuilt python sort function, only for sorting the output of the `get_completion_time` method of the `StrawHatTreasury` class. **Do not use inbuilt sort function for anything else**.

7. You don't need to be concerned about the recursion limit as long as your code runs within the expected time complexity. We will adjust the recursion limit in the autograder.

8. You are not allowed to create any additional file. We have provided the file named `custom.py` which can be used for any additional functions and classes. You can also define any additional function and classes in other files. You are allowed to define additional methods for any class. But you are not allowed to change function signature for the methods defined in starter code. You are also not allowed to modify the init method of Treasure class.

5 Submission

1. You are expected to complete this assignment in the next 2 weeks during your lab hours. (Deadline : 6th October 11:59 PM)
2. The starter code for this assignment is available on MoodleNew.
3. The submission should contain following files
   (a) crewmate.py
   (b) custom.py
   (c) heap.py
   (d) straw_hat.py
   (e) treasure.py
4. Late submission policy is available on the course website.
5. For any queries refer to piazza thread. Please go through the entire thread (and the assignment doc) carefully before raising a new query. Any updates to the assignment will also be posted in this thread.
6. Friendly reminder: all assignments will be checked for plagiarism. We trust that you will adhere to the IITD Honor Code.