Galactic Cargo Management System

COL106 Assignment 2

August 27, 2024

1 Background

In the vast expanse of the galaxy, interstellar shipping companies face a critical challenge: efficiently packing cargo into their space cargo bins. Each cargo bin on a starship has a specific capacity, and each shipment of cargo comes in varying sizes and colors. Efficiently managing this cargo is essential for smooth space operations and avoiding costly delays. A new and upcoming company called The Galactic Cargo Management System (GCMS) needs your help to tackle this challenge.

The GCMS assigns unique integer IDs to both bins and objects. It handles cargo differently based on the color of the cargo, which represents special handling instructions:

1. **Blue Cargo** (`Compact Fit, Least ID`): represents standard shipments, the system uses the **Compact Fit** Algorithm. This algorithm assigns the cargo to the bin with the smallest remaining capacity that is still sufficient to hold the item. If there are multiple bins with this remaining capacity, the one with the **least ID** is chosen.

2. **Yellow Cargo** (`Compact Fit, Greatest ID`): The Compact Fit algorithm is used for this color as well with the caveat that the bin with the **greatest ID** is chosen in case of multiple bins with the same remaining capacity.

3. **Red Cargo** (`Largest Fit, Least ID`): represents delicate items, the system uses the **Largest Fit** Algorithm. This algorithm assigns the cargo to the bin with the largest remaining capacity. If there are multiple bins with this remaining capacity, the one with the **least ID** is chosen.

4. **Green Cargo** (`Largest Fit, Greatest ID`): The Largest Fit algorithm is used for this color as well with the caveat that the bin with the **greatest ID** is chosen in case of multiple bins with the same remaining capacity.

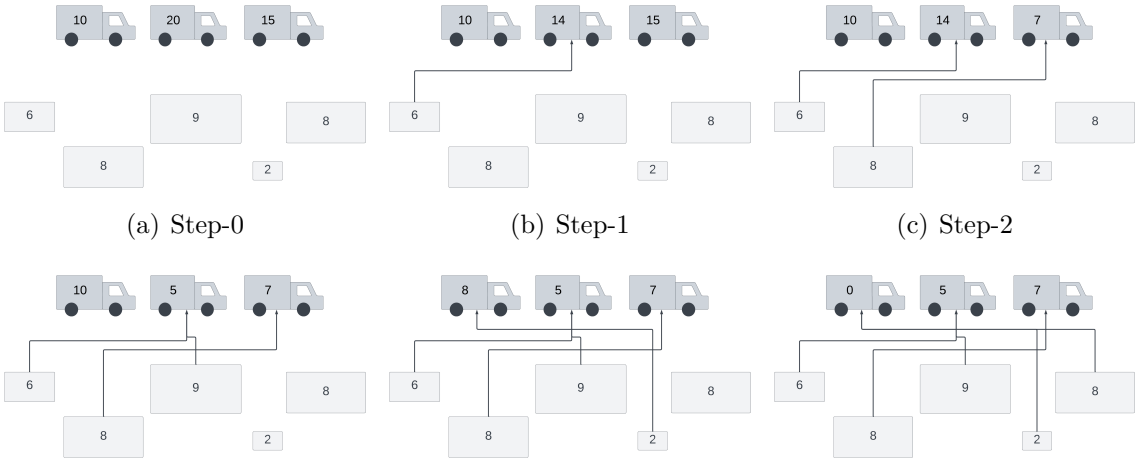
2 Modelling

The problem in our context can be described as follows: there are n bins where each bin has a **capacity** and an **ID** which are both integers. We are also given objects in order where each object has a **size** and a **ID** which are again integers along with a **color** as mentioned above.

2.1 Largest Fit Algorithm

The Largest Fit Algorithm is a well-known approach for solving the bin packing problem. When adding an object to a bin, this algorithm selects the bin with the largest remaining capacity.

To illustrate the largest fit algorithm, suppose there are 3 bins with capacities 10, 20, and 15, respectively. Suppose there are 5 objects with sizes 6, 8, 9, 2, and 8 (added in this order). The best fit algorithm works as follows:



(a) Step-0 (b) Step-1 (c) Step-2

(d) Step-3 (e) Step-4 (f) Step-5

Figure 1: Working of the Largest Fit Algorithm

1. Add the first object (size 6) to the bin with the largest remaining capacity. The bins' remaining capacities are [10, 20, 15]. The object goes to the second bin. Remaining capacities: [10, 14, 15].

2. Add the second object (size 8) to the bin with the largest remaining capacity. The bins' remaining capacities are [10, 14, 15]. The object goes to the third bin. Remaining capacities: [10, 14, 7].

3. Add the third object (size 9) to the bin with the largest remaining capacity. The bins' remaining capacities are [10, 14, 7]. The object goes to the second bin. Remaining capacities: [10, 5, 7].

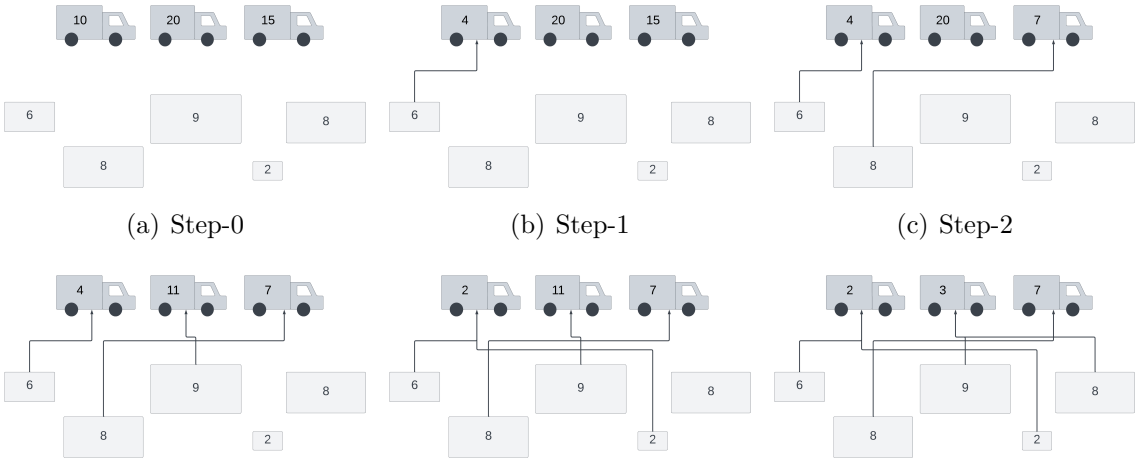
4. Add the fourth object (size 2) to the bin with the largest remaining capacity. The bins' remaining capacities are [10, 5, 7]. The object goes to the first bin. Remaining capacities: [8, 5, 7].

5. Add the fifth object (size 8) to the bin with the largest remaining capacity. The bins' remaining capacities are [8, 5, 7]. The object goes to the first bin. Remaining capacities: [0, 5, 7].

2.2 Compact Fit Algorithm

The Compact Fit Algorithm is another algorithm for the bin packing problem.
When adding an object to a bin, this algorithm selects the bin with the smallest remaining capacity that can accommodate this particular object.

To illustrate the compact fit algorithm, we consider the earlier example with 3 bins of capacities 10, 20, and 15, respectively and 5 objects with sizes 6, 8, 9, 2, and 8 (added in this order). The compact fit algorithm works as follows:



(a) Step-0 (b) Step-1 (c) Step-2

(d) Step-3 (e) Step-4 (f) Step-5

Figure 2: Working of the Compact Fit Algorithm

1. Add the first object (size 6) to the bin with the largest remaining capacity. The bins' remaining capacities are [10, 20, 15]. The object goes to the first bin. Remaining capacities: [4, 20, 15].

2. Add the second object (size 8) to the bin with the largest remaining capacity. The bins' remaining capacities are [4, 20, 15]. The object goes to the third bin. Remaining capacities: [4, 20, 7].

3. Add the third object (size 9) to the bin with the largest remaining capacity. The bins' remaining capacities are [4, 20, 7]. The object goes to the second bin. Remaining capacities: [4, 11, 7].

4. Add the fourth object (size 2) to the bin with the largest remaining capacity. The bins' remaining capacities are [4, 11, 7]. The object goes to the first bin. Remaining capacities: [2, 11, 7].

5. Add the fifth object (size 8) to the bin with the largest remaining capacity. The bins' remaining capacities are [2, 11, 7]. The object goes to the second bin. Remaining capacities: [2, 3, 7].

3 Requirements

The GCMS will provide a unique identifier (ID) for each object and each cargo bin which can be any arbitrary integer. Note that two bins or two objects can have the same size.

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

3.1 GCMS

Refer to `gcms.py` in the starter code. The constructor of this class initializes an empty system of cargo bins and objects. You need to implement the `GCMS` class with the below functions :

- **add_bin(bin_id, capacity)**: Add a new bin with a specified ID and capacity.
- **add_object(object_id, size, color)**: Add a new object with a specified ID, size and color choosing the appropriate algorithm based on the color of the object.
  Example :- (12345, 50, Color.RED)
  Note : The bin capacity is reduced by the size of the added object in this operation.
- **delete_object(object_id)**: Delete the object with the specified ID from its assigned bin.
- **object_info(object_id)**: Print the ID of the bin the object has been placed into.
- **bin_info(bin_id)**: Return a tuple with first element as the remaining capacity of the bin and second element as list of object IDs currently stored in the bin.
  Example :- (44, [1432, 1340, 1500])

3.1.1 Clarifications

We assume that there are `n` bins and `m` objects in a system.

1. You are not allowed to use Dicts (Hashing) or Sets internally implemented in Python.
2. The program should have a space complexity of $O(n + m)$.
3. The program should also have a time complexity of at most $O(log(n) + log(m))$ for the first four functions above and $O(log(n) + S)$ for the `bin_info` function where `S` is the number of objects in the corresponding bin.
4. If the object cannot be placed into any bin as per the algorithm, the program should raise the `NoBinFoundException`.
5. You must not import from any new Python libraries and implement the functionality from scratch.