Project 2
MA 444
Chris Sadler and Trevor Morton

<u>Problem</u>

Running a fleet of container ships has a number of complex modeling problems to consider; for instance, route planning, finding the optimal cargo container size, or optimizing fuel and weight costs. Our problem is related to cargo ships, specifically efficiently transferring containers on to and off of the ship at each port. The problem is to create a method of loading and unloading these containers while minimizing the number of containers that are shifted.

We define shifting a container as temporarily moving a container from the ship and replacing it back on the ship (not necessarily in the same location). Loading a container onto a ship does not count as a shift; similarly, removing a container from a ship at its destination port is not a shift. Because containers can be stacked on a ship, some containers whose destination is not the current port may need to be shifted to get to containers whose destination is the current port. We assume that we can assign a cost of 1 to moving a container from the top of a stack of containers. For example, the cost of removing a container buried under 3 other containers (that do not need to be removed) is 3. There is no additional cost incurred if there was another box we needed under this stack of three, because the cost can be thought of as having to take the box off the ship. This means that the boxes will already be off the ship if you had to get at any boxes under them, regardless of how many there are.

We assume that we are given the N ports the ship stops at, the capacity of the ship is $X = R * C$, where $R$and $C$are the number of rows and columns of a rectangular grid to pack the containers into. We also assume that we have a transportation matrix T_{ij} $(i \in 1 .. N-1, j \in 2 .. N)$that defines how many containers are going from port i to port j.

<u>Algorithm</u>

Step 1:
Starting with stop number 1, find all the packages that leave from the current stop.
Step 2:
Find the package that starts at the current stop and has the destination that is the farthest away.
Step 3:
Find a package that is already loaded in the cargo hold that has room above it and is destined for the same destination or a farther destination and place the current package on top of it.
Step 4:
If a spot was not found for the current package in the last step, find an empty column in the cargo hold and place the current package at the bottom of it.
Step 5:

If a spot was not found for the current package in the last step, place the current package on top of the package with an open spot above it whose destination is coming up first.
Step 6:
If a spot was not found for the current package in the last step, the problem is invalid.
Step 7:
Repeat from step 2 until there are no more boxes that need to be packed leaving from the current stop.
Step 8:
Move on to the next stop, so now the current stop is one more than it was. Remove all boxes from cargo hold that had a destination of this stop. If there are any boxes above a box for this stop, increase the cost by 1 for each box that is above one for this stop that is not destined for this stop. Each box that had a cost increased for it will be added to the boxes loaded on at this port and removed from the bay temporarily.
Step 9:
Start at step 1 treating the current stop as what it was in the last step and adding the temporarily removed packages from the last step to the pool of packages that will get loaded at this stop. If there are no more stops to go to, return the cost as the answer.

Design

To come up with this algorithm, we began by considering the brute force case: lay each container with the farthest destination port in the bottom most available row of the cargo hold. We soon realized that it was much more efficient to arrange the containers by column instead, since less containers would then block a potential container from being able to be removed. This is because containers with the same destination will be stacked on top of each other, which reduces the cost of moving a container underneath another container with the same destination.

After this, we began to deal with the cases of where to put incoming containers. We realized that to avoid placing later destination containers on top of earlier containers we could just fill empty columns with incoming containers when their number was higher. A minor optimization we realized in this was that we could stack a container on top of another container in an existing pile if the destination of the first container is equal to or nearer than the destination of the second container. In this way we could more efficiently pack space. Finally, we had to handle the case that it was impossible to find an empty column to add additional containers to. We decided to stack these containers on top of a container with the nearest destination so that we can avoid stacking more containers on top of that one that will be taken off the one at the base. Because we redistribute containers that are temporarily moved, this allows them to be potentially placed in a more optimal position at the next port.

Test results

Data File 1, Cost = 0
Data File 2, Cost = 0
Data File 3, Cost = 1

Data File 4, Cost = 1
Data File 5, Cost = 3
Data File 6, Cost = 2
Data File 7, Cost = 0

See Results files for more details on where each box must be loaded in each case to achieve these costs.