Train shunting

Nariman Haidar

Spring Term 2023

Introduction:

We will be in charge of shunting wagons of a train that has a "main" track and "one" and "two" shunting tracks in this assignment. A move occurs when a wagon switches from one set of tracks to another. Data structures like lists and tuples will be used for this and we won't use any library modules instead, we'll write each function from scratch.

Modeling:

We will first determine whether the train wagons are to the left or the right and how they moves. The wagon closest to the path leading to the main track will be the first item on the list for tracks one and two. The leftmost wagon on the track is the first item in the list to symbolize the main track. Then the moves are a binary tuple. Here is how one state move is transformed into another: The n rightmost wagons from track "main" to track "one" are moved if the move is : one, n and n is greater than zero. More than n wagons on track ":main" result in the remaining wagons. The n left-most wagons are moved from track "one" to track "main" if the move is : one, n and n is less than zero. The additional wagons stay on track "one" if there are more than n wagons there. The move : $one, 0$ has no effect.

Some train processing:

In this section I am going to explain the code that I wrote: The first file is train.ex: In this file I defined the module called Train, then define the function take that returns the train's first "n" elements. The complete train is returned if "n" is bigger than the length of the train. Then define the function drop that removes the first 'n' components from the train and then returns the rest. An empty list is returned if "n" is bigger than the length of the train. then define the append function that appends two trains together. Then define the function member to determines whether an element is a part of the train. Then define the function position that returns the first position

(1 indexed) of y in the train train. You can assume that y is a wagon in train. Then split function that return a tuple with two trains, all the wagons before y and all wagons after y. Then the last function is the main function that returns the tuple $k, remain, take$ where k is the number of wagons that must still be present in order to have n wagons in the taken part.

Applying moves:

Then we are going to define a module called Moves that has two functions called single and sequence that, respectively, represent individual moves and sequences of moves. First define the single function that receives a tuple indicating a move and the train's current state returns the train's new state following the move. A tuple of the type "track, n" is used to express the move. Track can either be one or two, and n is an integer that represents the number of wagons that need to be moved. The list of wagons on the main track, the list of wagons on track one, where main is a list of wagons on the main track, one is a list of wagons on track one, and two is a list of wagons on track two. Then we define the sequence function that takes a list of moves and a starting state of the train, and returns a list of states, each representing the state of the train after executing a sequence of moves.

The shunting problem:

Finally, in order to address the shunting issue, a file called shunt.ex will be created. "Shunt" module, which includes a number of functions for shunting a train of wagons. Wagons are shuffled from the main track to one of two parallel tracks, and then they are shuffled back to the main track in a different arrangement. The first function, "find," has two arguments: a list of the current wagons (represented by "xs") on the main track and a list of the desired order of the remaining wagons (represented by "ys") on the main track after shunting. The function produces a list of tuples that represents the series of shunting maneuvers required to arrange the wagons in the desired order. The function performs the shunting operations by repeatedly calling itself with an updated train. The second function, "few," is comparable to "find" but is designed to move a small number of wagons. Recursion is also used to carry out the shunting operations. The third function, "count," counts the number of wagons in a train as a convenience function. The actual counting is carried out by the private function "do count".