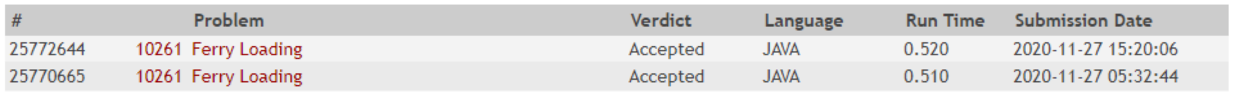
Programming Assignment 2

Ferry Loading Problem Report

(Feier Zhang, 8589976)

Part 1: Big Table

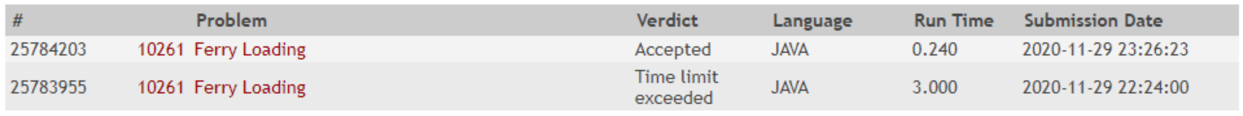
For backtracking using big table memorization, my code passed all the test posted on the Brightspace and was accepted by the online judge. Below is the screen shot of the big table implementation after the online judge, and the runtime is 0.520 and 0.510 separately. Actually, I did not change my code between these two times.



#	Problem	Verdict	Language	Run Time	Submission Date
25772644	10261 Ferry Loading	Accepted	JAVA	0.520	2020-11-27 15:20:06
25770665	10261 Ferry Loading	Accepted	JAVA	0.510	2020-11-27 05:32:44

Part 2: Hash Table

For backtracking using hash table memorization, my code passed all the test posted on the Brightspace and was accepted by the online judge. Below is the screen shot of the hash table implementation after the online judge, and the best runtime is 0.240.



#	Problem	Verdict	Language	Run Time	Submission Date
25784203	10261 Ferry Loading	Accepted	JAVA	0.240	2020-11-29 23:26:23
25783955	10261 Ferry Loading	Time limit exceeded	JAVA	3.000	2020-11-29 22:24:00

Actually, this work implemented two kinds of Hashing techniques: 1. Chaining; 2. Linear Probing. The discussion would be based on the observation from the development and its observation.

1. Chaining:
- **Main idea**: Firstly, I created empty chains. Then, when the state(key, value) is visited, I will store the value to the chain according the key. For example, when adding car to the port, then the key is “currK+1” and the value is “newS”.
- Using ArrayList to store chains
- **Hash function** is: index = key.
- Disadvantages:
    a. The search and insert function are too time consuming.
    b. Too much collision
- Therefore, the chaining technique passed the tests posted on the Brightspace, but it failed in the online judge, because of exceeding time limit.

2. Linear Probing:

- **Main idea**: Firstly, I initialized each value in the hash table(1-D array) equals “-1”. Then, when the state(key, value) is visited, I will store the sum of the “key” and the “value”(The motivation is because the number of collision is very small) to the hash table according to the hash function. For example, when adding car to the port, then the key is “currK+1” and the value is “newS”.
- Using 1-D Array as the Hash Table;
- **Hash function** is: (key+value) % hashTableSize.
- **Hash Table Size** is: total_number_of_cars + length_of_ferry;
- Therefore, when I need to check whether the state has been visited or not, I just need to check the sum of the key and value is in the hash table or not. And the runtime is 0.240.