Assignment Two

The ferry loading problem

Stefan Todorovic

University of Ottawa

CSI 2110

WonSook Lee

December 4, 2020

A. Status of Part 1 – BigTable

For Part 1 of the assignment, the code implemented a backtracking algorithm using a Bigtable memorization method. In specific a 2d Boolean array was used to store the visited locations. The code has passed all test provided in the documentation for this assignment. Every generated outputX.txt file matches the provided outputX.txt. In addition, the code passed Online judge certification and was accepted.

Below is the attached Online Judge screenshot for 3 runs of BigTable. The average runtime is 0.4566 s.

#	Problem	Verdict	Language	Run Time	Submission Date
25800492	10261 Ferry Loading	Accepted	JAVA	0.430	2020-12-03 05:05:23
25800490	10261 Ferry Loading	Accepted	JAVA	0.480	2020-12-03 05:05:11
25800487	10261 Ferry Loading	Accepted	JAVA	0.460	2020-12-03 05:04:53

Figure 1: OnlineJudge Runtimes for Part 1

B. Status of Part 2 – HashTable

For Part 2 of the assignment, the code implemented a backtracking algorithm using a HashTable memorization method. In specific a HashMap array was used to store the visited locations. The code has passed all test provided in the documentation for this assignment. Every generated outputX.txt file matches the provided outputX.txt. In addition, the code passed Online judge certification and was accepted.

To determine the best size of HashTable, code was inserted into the program to print the time in milliseconds that it took to an entire input. Below is the information that has come out from this research. This was then used to run on online judge.

Table Size	*Run 1*	*Run 2*	*Run 3*	*Average*
500	52	53	46	50.33333
1000	51	44	44	46.33333
2000	41	41	41	41
4000	43	46	51	46.66667
8000	52	47	47	48.66667
15000	51	43	52	48.66667

Table 1: Runtimes from local testing of various HashMap sizes

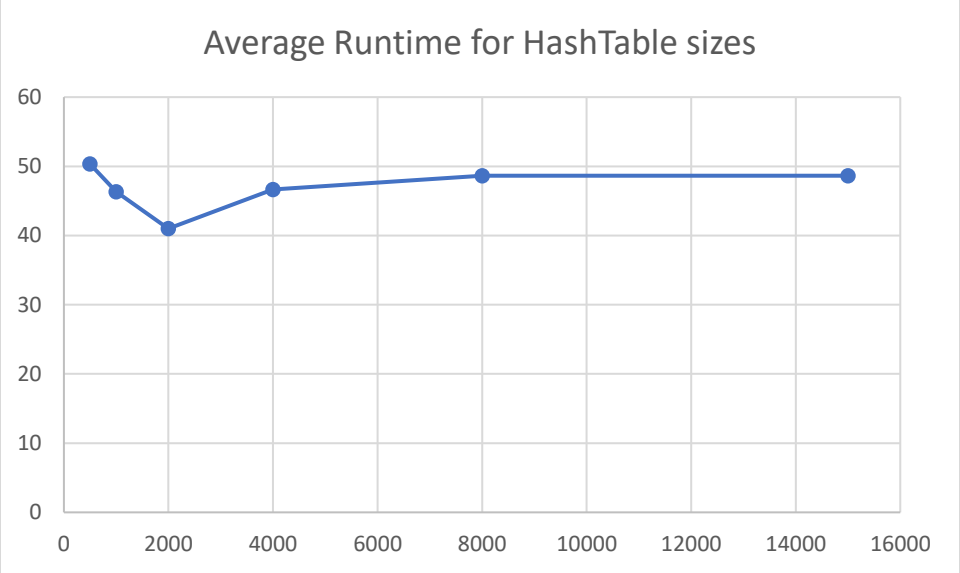

Figure 2: Runtimes from Table 1 as a graph

Below is the attached Online Judge screenshot for 3 runs of HashTable. The average runtime is 0.14 s.

#	Problem	Verdict	Language	Run Time	Submission Date
25811578	10261 Ferry Loading	Accepted	JAVA	0.130	2020-12-05 03:43:40
25811577	10261 Ferry Loading	Accepted	JAVA	0.150	2020-12-05 03:43:35
25811570	10261 Ferry Loading	Accepted	JAVA	0.140	2020-12-05 03:43:01

Figure 3: OnlineJudge Runtimes for Part2

C. Specifying the hash function of Part2

For Part 2 the HashMap was set to an initial size of 2000. As Java's Built in HashMap function already accounts for collections, there was no problem solving necessary on that front.

For the hash to be calculated the following formula was used (k+s)%2000. A function called setHash(int varK, int varS) was made in order to calculate the hash based on the currK and currS/newS used from the BigTable solution.

As I was aware that using this type of Hash already delt with collisions minimum problem solving was required. The only thing left was to determine a hash based on the table and runtime from Table 1.