Big Table

Thought Process

I decided to add an addition parameter to the recursive function to store the right-side value so it would not need to be recomputed every time. I then implemented two Boolean values to determine if the left and right side have space. If both the left side and right side are full or if all the cars are added, then no more cars are added, and this is the base case. The code then checks to see if bestX and bestK should be updated. I made my code in this way because I thought it would be better to only clone the currX into bestX at the end of the runs since running unnecessary cloning functions is very inefficient. I also made the currX and bestX into Booleans since they can only have 0 or 1 as values which can be substituted with false and true.

Initial Issues

I had a lot of issues with making my submission work with the online judge. I was able to isolate the problem to the input/output section of my code by running the test cases provided and verifying the outputs. Since the outputs were immaculate it meant that the recursive function was not the issue, rather the online judge does not like my output. To solve the input/output issue I remade the output section to fix the common issue with blank lines. I also switched from scanner to buffered reader because buffered reader is slightly more efficient.

Overall Code Correctness

After fixing issues with online judge submission the code worked with both the provided tests and the online judge. See below for online judge and the outputs are provided in the zip folder.

Online Judge

25794774	10261 Ferry Loading	Accepted	JAVA	0.160	2020-12-02 02:14:51
25794769	10261 Ferry Loading	Accepted	JAVA	0.130	2020-12-02 02:12:26
25794764	10261 Ferry Loading	Accepted	JAVA	0.170	2020-12-02 02:11:45

Hashtable (HashMap)

Original Recursion Table Size: (n*(n+1))/2+1

Hashtable Calls:

!visited. getOrDefault ((currK+1)+","+newLS, false)

visited.put((currK + 1)+","+(newLS), true)

Change Made	Timing (avg 3 runs)
DEFAULT	2.42
Table Size Changes	
n*sideLength i.e same as bigtable	2.26
n	2.49

The first thing I did was just to run it using Java's STL HashMap and I experimented with changing the sizes. There seemed to be little difference when I changes the table sizes leading me to believe that table size is not as important as other aspects of the code and I should tweak those instead. However, it did work with the online judge and passed all the outputs, so it was a good start.

Upon Further Thought

I have decided to use Gausses formula +n for the initial table size because each state can have either all n cars on that side n-1 cars etc to 0. Gausses formula calculates the sum of all numbers from 1-n but the issue is since I am including zero there is an extra case each time hence the + n. This should be the most memory efficient way that also avoids remaking the table. This however did not really change the run time, so I am just going to trust my math on this one and move onto optimizing other stuff.

Big Changes

So, I decided to change the key from a String to a Double. Before the decimal point is the length and afterwards is the current number of cars. I thought a double would be more efficient since double is a primitive value wrapped with the object Double so it works in collections. I changed the getOrDefault to containsKey to see if that would be faster, but it did not really make a difference and the speed is now averaging 2.2.

Comeback

Discouraged but not defeated, I decided to give it another crack the next day. I thought to expand on my existing idea of using better keys and I made my own class called "visit" that stores the length of the left size and the number of cars as ints. I continued to use the java hashmap because I know its been optimized to levels I could only dream of. However, I did some research and I found I could override the existing hashcode function for my object. I auto-generated new hashcode and equals functions using the eclipse IDE auto generation function.

Tinkering

I found the equals method contained some unnecessary checks like if obj == null since I knew there were no null objects in my hashmap I removed it. However, one line that did cause a significant improvement in run time was the this == obj. I assume there were some calls of the same object to the hashmap and that saved some time. I also obviously checked if the length and car number were the same. The hashCode implementation is the basic eclipse one and should be quite good at avoiding collisions because unless the two cases are equivalent at least one of the two variables, cars and length, should be unique. Intuitively, I know that using a bunch of different numbers in the same equation generally yields different results, so I was happy with the function. Now the only think left to check is if changing the size does anything to this new implementation. I was averaging .62 with my current size and with n*sideLength I got around .5. I know that there will be no table rehashing at that size, so I knew my current size was not perfect. I decreased the table size to n*sideLength/2 and I had it a time of .560. I then tried it /50 and I got .610.

HashMap Conclusions

My code works with both the online judge and the provided tests. See below for online judge and in the zip folder for the outputs of the provided tests. There is not much left to be said except what size I settled on. I decided to go with sideLength*n/50 because it would most consistently return .6 while the Gauss one would be all over the place. However, I knew from the discrepancies between different runs that some table rehashing did exist.

25804159	10261 Ferry Loading	Accepted	JAVA	0.720	2020-12-03 20:41:52
25804154	10261 Ferry Loading	Accepted	JAVA	0.630	2020-12-03 20:40:55
25804152	10261 Ferry Loading	Accepted	JAVA	0.600	2020-12-03 20:40:45