CSC263 Project Part 3

Arun Jolly Marlier, Eric Hasegawa, Faustine Leung

September 8, 2020

For both implementations,
- Let m be the number of restaurants in the graph
- Let n be the number of cars in the graph
- getDrivingTime() runs in $\Theta(1)$

1 Randomized Operation Implementations

addCar(Car c)
Time Complexity:
- BC: $\Theta(m)$
- WC: $\Theta(m)$

Space Complexity:
curr_restaurant takes $\Theta(n)$ of space because restuarants[r_index] is an array of size n. The space needed to append the Weight_Key_Pair is $\theta(1)$ and same for inserting into car_dict. This is repeated m times. In addition, c takes $\Theta(6)$ of space, $\Theta(1)$ for each of the 6 attributes. Therefore, the space complexity is $\Theta(mn)$.

removeCar(Car c)
Time Complexity:
- BC: $\Theta(1)$
- WC: $\Theta(n+m)$

Space Complexity: found takes $\Theta(1)$ of space. We repeat the following n times: target_id takes $\Theta(1)$ of space and changing the value of found to True takes $\Theta(1)$ of space. In addition, c takes $\Theta(6)$ of space, $\Theta(1)$ for each of the 6 attributes. Therefore the space complexity is $\Theta(n)$.

addRestaurant(Restaurant r
Time Complexity:
- BC: $\Theta(n)$
- WC: $\Theta(n)$

Space Complexity:
Since the empty list we are appending will be of size n, let's say that the space complexity of appending an empty list to adj_mat is $\Theta(n)$. The following is repeated n times: curr_car_id takes $\Theta(1)$ of space, curr_car takes $\Theta(6)$ of space for the 6 attributes that Car has (each taking $\Theta(1)$ of space), Weight_Key_Pair taks $\Theta(2)$ of space for its 2 pointers, and appending r to restaurants takes $\Theta(4)$ of space for its 4 pointers. In addition, r takes $\Theta(4)$ of space, $\Theta(1)$ for each of its 4 attributes. Therefore, the space complexity is $\Theta(n+n(1+6+2+4)+4) = \Theta(n)$.

removeRestaurant(Restaurant r)
Time Complexity:

- BC: $\Theta(1)$
- WC: $\Theta(m)$

Space Complexity:
Setting found to False takes $\Theta(1)$ of space. Since we only set found to True once, as it is under the if statement and only happens once, this takes $\Theta(1)$ of space.In addition, r takes $\Theta(4)$ of space, $\Theta(1)$ for each of its 4 attributes. Therefore, the space complexity is $\Theta(1)$.

setEdge(Car c, Restaurant r, int new_time)
Time Complexity:
- BC: $\Theta(1)$
- WC: $\Theta(mn)$

Space Complexity:
The parameters include a Car object with takes $\Theta(6)$ of space, a Restaurant object which takes $\Theta(4)$ of space, and an integer which takes $\Theta(1)$ of space. Additionally, the checking boolean found, and both loop variables take $\Theta(1)$ space each. Therefore space complexity is $\Theta(1)$.

getEdge(Car c, Restaurant r)
Time Complexity:
- BC: $\Theta(1)$
- WC: $\Theta(mn)$

Space Complexity:
The parameters include a Car object with takes $\Theta(6)$ of space, and a Restaurant object which takes $\Theta(4)$ of space. Additionally, the checking boolean found, both loop variables, and the temporary car id take $\Theta(1)$ space each. Therefore space complexity is $\Theta(1)$.

getCars()
Time Complexity:
- BC: $\Theta(1)$
- WC: $\Theta(1)$

Space Complexity:
This returns an existing list already and has no parameters, therefore space complexity is $\Theta(1)$.

getRestaurants()
Time Complexity:
- BC: $\Theta(1)$
- WC: $\Theta(1)$

Space Complexity:

This returns an existing list already and has no parameters, therefore space complexity is $\Theta(1)$.

delivery(Restaurant r, int cook_time)

Time Complexity:
$BC \in \Theta(nlogn)$

Space Complexity:
- Firstly, the algorithm finds the target restaurant and then sorts the associated sublist. The loop variable only adds a a constant amount of space, but the sorting algorithm uses temporary lists to hold the two partitions of the given list. This means that, in total, n items will be stored temporarily. The rest of the sorting algorithm only adds constant space so therefore n items are stored in total. Adding this to the n items stored as a parameter in the first call of sort, we get $2n$ total items stored.

- We then initialize an empty list and populate it with the first 3 elements of the now sorted sublist.

- Therefore Space Complexity $\in \Theta(n)$.

Expected Running Time Analysis

We know that the number of comparisons at each level relies on the selected pivot point.

Aside from our sorting algorithm, the rest of delivery runs at the same speed no matter the input. Therefore the average runtime for n seperate random split points is $\frac{m}{n}$.

Now looking at the sorting algorithm, no matter our chosen split point, we will still have to make the same amount of comparisons as normal merge sort, so $nlogn$.

Therefore $EC_{Delivery}$ =(Probability of selecting a specific point to split at)$(\frac{m}{n} + nlogn) = \frac{1}{n}(\frac{m}{n} + nlogn) \in \Theta(m + logn)$.

Worst Case Running Time Analysis

Upper Bound
For delivery to be in worst case, the parameter Restaurant r needs to be the last element of the list restaurants.

This implies the for loop on line 2 iterates m times.

Once the target is found, we sort that sublist using an augmented merge sort, where we choose the split point at random, rather than at the midway point.

Since this augmentation does not change the overall runtime of merge sort, then merge sort will run in the worst case at $nlogn$.

Therefore $WC_{Delivery}$ takes $m + nlogn \in O(m + nlogn)$.

Lower Bound
Let r be the last element in restaurants.

Then the for loop in line 2 iterates m times.

Once the target is found, we sort that sublist using an augmented merge sort, where we choose the split point at random, rather than at the midway point.

Since this augmentation does not change the overall runtime of merge sort, then merge sort will run in the worst case at $nlogn$.

Therefore $WC_{Delivery}$ takes $m + nlogn \in \Omega(m + nlogn)$.

Therefore $WC_{Delivery} \in \Theta(m + nlogn)$.

2 Amortized Analysis Implementations

Space Complexity Generalizations:
A Restaurant object takes space $\Theta(logn + logm)$, $logn$ for the pointer to the first Car Node object in its linked list of cars and $logm$ for the pointer to the next Restaurant object.
A Car object takes space $\Theta(logn)$, $logn$ for the pointer to the next Car object.

addCar(Car c)
Time Complexity:
- BC: $\Theta(m)$
- WC: $\Theta(mn)$

Space Complexity:
curr_rest is a Restaurant object, so it takes $\Theta(logn + logm)$ of space. The inner while loop is repeated n times, where curr_car is a Car object that takes $\Theta(logn)$ of space. In total, this takes $\Theta(nlogn)$ of space. curr_car.next is also a Car object and takes $\Theta(logn)$ of space. curr_rest is a Restaurant object and takes $\Theta(logn + logm)$ of space. This process starting from the inner while loop is repeated m times. In addition, c takes $\Theta(logn)$ of space. Therefore, the space complexity is $\Theta(logn + logm + m(nlogn + logn + logn + logm) + logn) = \Theta(logn + logm + mnlogn + m(logn + logm)) = \Theta(mnlogn + mlogm)$.

removeCar(Car c)
Time Complexity:
- BC: $\Theta(m)$
- WC: $\Theta(mn)$

Space Complexity:
curr_rest takes $\Theta(logn + logm)$ space. The inner while loop repeats n times and prev_car.next, curr_car, and prev_car takes $\Theta(logn)$ of space, as they all are Car objects. curr_rest takes $\Theta(logm)$ since it is a Restaurant. This whole process starting from the inner while loop is repeated m times. In addition, c takes $\Theta(logn)$ of space. Therefore, the space complexity is $\Theta(logn + logm + m(n(logn)+logm))+logn) = \Theta(logn+logm+mnlogn+mlogm) = \Theta(mnlogm+mlogm)$.

addRestaurant(Restaurant r
Time Complexity:
- BC: $\Theta(n)$
- WC: $\Theta(n)$

Space Complexity:
The first three steps take $\Theta(logn)$ of space. The steps inside the for loop including the iterator car take $\Theta(logn)$ as all the variables are Car objects. The for loop is repeated n times. In addition, r takes $\Theta(logm+logn)$ of space. Therefore, the

space complexity is $\Theta(logn + nlogn + logm + logn = \Theta(logn + nlogn + logm) = \Theta(nlogn + logm)$.

removeRestaurant(Restaurant r)
Time Complexity:
- BC: $\Theta(1)$
- WC: $\Theta(m)$

Space Complexity:
curr takes $\Theta(logn + logm)$ of space since it is a Restaurant. The steps inside the while loop take $\Theta(logn + logm)$ of space, where the while loop runs m times in the worst case. The last step in the if statement takes $\Theta(logn + logm)$ of space. In addition, r takes $\Theta(logn + logm)$ of space. Therefore, the space complexity is $\Theta(2(logn + logm) + m(logn + logm) + logn + logm) = \Theta(logn + logm + mlogn + mlogm) = \Theta(mlogn + mlogm)$.

setEdge(Car c, Restaurant r, int new_time)
Time Complexity:
- BC: $\Theta(1)$
- WC: $\Theta(n)$

Space Complexity:
curr takes $\Theta(logn)$ of space since it is a Restaurant. The while loop runs n times in the worst case and the step inside takes $\Theta(logn)$ of space. The last step in the if statement takes $\Theta(1)$ of space. In addition, the parameters take $\Theta(logm + logn)$ of space. Therefore, the space complexity is $\Theta(logn + nlogn + 1 + logn + logm) = \Theta(logn + nlogn + logm) = \Theta(nlogn + logm)$.

getEdge(Car c, Restaurant r)
Time Complexity:
- BC: $\Theta(1)$
- WC: $\Theta(n)$

Space Complexity:
curr takes $\Theta(logn)$ of space since it is a Restaurant. The while loop runs n times in the worst case and the step inside takes $\Theta(logn)$ of space. The last step in the if statement takes $\Theta(1)$ of space. In addition, the parameters take $\Theta(logm + logn)$ of space. Therefore, the space complexity is $\Theta(logn + nlogn + 1 + logn + logm) = \Theta(logn + nlogn + logm) = \Theta(nlogn + logm)$.

getCars()
Time Complexity:
- BC: $\Theta(1)$
- WC: $\Theta(1)$

Space Complexity:

No extra space is needed. Therefore, the space complexity is $\Theta(1)$.

getRestaurants()
Time Complexity:
- BC: $\Theta(m)$
- WC: $\Theta(m)$

Space Complexity:
curr is a Restaurant, so it takes $\Theta(logn + logm)$ of space. restaurants will be an array of size m, so let's say it takes $\Theta(m)$ of space. The while loop runs m times in the worst case with each step taking $\Theta(logn + logm)$. Therefore, the space complexity is $\Theta(logn + logm + m + m(logn + logm)) = \Theta(logn + logm + mlogn + mlogm) = \Theta(mlogn + mlogm)$.

delivery(Restaurant r, int cook_time)
Time Complexity:
- BC: $\Theta(n)$

Space Complexity:
The parameters takes $\Theta(logn+logm)$ of space due to r. Since values and car_ids will be of size n, they take $\Theta(n)$ of space.

The next while loop is repeated n times with each step taking $\Theta(logn)$ of space since curr is a Car Node. In total, this portion takes $\Theta(nlogn)$ of space.

A Node object takes $\Theta(logn)$ of space as it has a pointer to the next Node object and there are n nodes in total. Thus, head takes $\Theta(logn)$ of space. The for loop is repeated n times and setting head takes $\Theta(logn)$ of space as it is a Node. We will examine the space complexity for push().

push() has parameters that take $\Theta(logn)$ of space. new takes $\Theta(logn)$ of space and the rest of the helper function takes $\Theta(nlogn + logn)$ of space. Therefore, the space complexity of push() is $\Theta(nlogn + logn)$ of space.

Thus, the for loop that calls push() takes $\Theta(n(logn+nlogn+logn) = \Theta(nlogn+ n^2logn) = \Theta(n^2logn)$.

pop() takes $\Theta(logn)$ of space. The for loop that calls pop() takes $\Theta(logn)$ of space.

The next block of code takes $\Theta(n)$ of space as the steps in the for loop take $\Theta(1)$ of space. The last block of code takes $\Theta(1)$ of space.

Therefore, the space complexity is $\Theta(logn+logm+n+nlogn+n^2logn+logn+ n+1) = \Theta(n^2logn + logm)$.

Amortized Analysis
We will use the aggregate method to compute the amortized cost.

All parts of delivery() are trivial for the amortized analysis except for the helper function push(Node head, int new_val), as the rest of the operation clearly consists of steps with constant time or loops. Therefore, we will find the amortized cost of push().

We want to find the worst-case sequence complexity $WCSC$, as amortized cost $AC = \frac{WCSC}{k}$ for a k-sequence of calls to push().

In total, for a k-sequence of calls to push(), how many times do we have to swap the nodes in the linked list to conserve the values in the linked list being in increasing order? We have that $WCSC \leq k^2$, as for each insert there are at most k comparisons to check if the linked list is in increasing order and we have a sequence of k calls to push().

Therefore, $AC = \frac{k^2}{k} = k$.

Combining this analysis with the analysis for delivery(), the first while loop where we store the values in a list runs in $O(n)$. When we insert all the values in the linked list, we call push() n times and thus, this step runs in $O(n^2)$ based on our amortized analysis of push(). When we find the ids of the three cars with the smallest value, the loop runs $len(values)$ = length of linked list of cars = n times so this step runs in $O(n)$. Moreover, the rest of the steps run in constant time, including pop() that runs in $O(1)$.

Then, we have

$$\begin{aligned} \text{delivery} &\in O(n + n^2 + n) \\ &\in O(n^2) \end{aligned}$$

Worst Case Running Time Analysis

Upper Bound
The first while loop that appends each absolute value to a list clearly runs in $O(n)$. WCRT of push() is $O(n)$ when values is in decreasing order. We call push() n times, so this step runs in $O(n^2)$. pop() always runs in $O(1)$, so finding the three smallest values runs in $O(3 * 1 = 3)$. The loop to find the ids of the three cars with smallest absolute value clearly runs in $O(n)$.

Then, we have

$$\begin{aligned}\text{delivery} &\in O(n + n^2 + 3 + n) \\ &\in O(n^2)\end{aligned}$$

Lower Bound

The first while loop that appends each absolute value to a list clearly runs in $\Omega(n)$. WCRT of push() is $\Omega(n)$ when values is in decreasing order, as we must swap the nodes a minimum number of times to get the linked list in increasing order. We call push() n times, so this step runs in $\Omega(n^2)$. pop() always runs in $\Omega(1)$, so finding the three smallest values runs in $\Omega(3 * 1 = 3)$. The loop to find the ids of the three cars with smallest absolute value clearly runs in $\Omega(n)$.

Then, we have

$$\begin{aligned}\text{delivery} &\in \Omega(n + n^2 + 3 + n) \\ &\in \Omega(n^2)\end{aligned}$$

Therefore, WCRT for delivery is $\Theta(n^2)$.