Assignment1: Runtime Analysis during worst-case.

Using Direct Addressing:

How to Run: Open the unzipped assignment1 project in the solution. Change the input filename variable as per the data file in your local pc then execute the **Source** Class under the package “**directAddressing**” to perform simple operations on sample data. **ElectionCount** class has the implementations of ADD, FIND and COUNT operations.

Synopsis: Direct addressing has been implemented by creating an array (VoterArray) of 9 lakh size to fit all the 6 digit voterId’s. Hence each block of array stores the candidateId for that particular VoterId. I have also created an array (CandidateArray) to store the counts of each candidateId while adding the voterId. **For convenience, I am considering the number of possible VoterId’s as ‘n’ and the number of possible CandidateId’s as ‘m’(max 1000).**

**Add function complexity: As per the implementation, there are no loops for adding the data to** VoterArray**. So the Worst case runtime of the ADD operation is O(1). The other operation involved in my add function is, adding the data to** CandidateArray**. As there are no loops and the computation is very minimal for incrementing the value of the count, the worst-case runtime of this ADD operation is O(1). Hence the aggregated worst-case runtime is O(1) + O(1) = O(1).**

**Find function complexity: As per my implementation, there are no loops and accessing the array plus the subtraction operation to reduce the size of the array is minimal. Hence the worst-case runtime of FIND operation is O(1).**

**Count function complexity: As per my implementation, by having an array for storing the counts of CandidateId’s in a separate** CandidateArray **of 1000 slots which is negligible. There are no loops and accessing the array and comparison operations are considered negligible. Hence the worst-case find operation of the count operation is O(1).**

**Hence, I am able to achieve the O(1) worst-case for the ADD, FIND and COUNT operations by just having two arrays of which CandidateArray is only 1000 slots which is negligible. But, the VoterArray takes 9lakh slots which is a bit huge, is the only drawback of this approach.**

*Using Hash Table:*

How to Run: Open the unzipped assignment1 project in the solution. Change the input “filename” variable as per the data file in your local and then execute the **Source** Class under the package “hashTable” to perform simple operations on sample data. ElectionCount class has the implementations of ADD, FIND and COUNT operations on HashTable.

Synopsis: Hash table has been implemented by creating a VoterData Hashtable that takes VoterId as key and CandidateId as value pair. And, a CandidateData HashTable which takes CandidateId as the key and the count as value pair. We use the standard functions provided by the HashTable to achieve the required functionality. **For convenience, I am considering the number of possible VoterId’s as ‘n’ and the number of possible CandidateId’s as ‘m’(max 1000).**

**Add function complexity: As per the implementation, there are no loops for adding the data to** VoterData**. So the Worst case runtime of the HashTable ADD operation is O(n) as n is the max possible count of voterId’s. The other operation involved in my add function is, adding the data to** CandidateData**. As there are no loops and the computation is very minimal for incrementing the value of the count, the worst-case runtime of this HashTable ADD operation is O(m) as m is the possible candidatId’s which is 1000 in this case. Hence the aggregated worst-case runtime is O(n) + O(m) = O(n) \*here m is 1000 which is negligible when compared to 9,00,000 voterId. Worst Case - O(n), Average Case – Θ(1).**

**Find function complexity: As per my implementation, there are no loops and accessing the HashTable using a Get operation plus the containsKey operation is O(n) + O(n) = O(2n) which is O(n). Hence the worst-case runtime of FIND operation is O(n). Worst Case - O(n), Average Case – Θ(1).**

**Count function complexity: As per my implementation, by having a HashTable for storing the counts of CandidateId’s in a separate** CandidateData **of 1000 slots which is negligible. There are no loops and accessing the array and comparison operations are considered negligible. Hence the worst-case find operation of the count operation is worst-case find operation on a CandidateData Hashtable which is having max possible value of m which O(m) much less than O(n). Please note m is different from n. m is only 1000 in this case which is much smaller(0.0011) than the n of 9,00,000. Worst Case - O(m), Average Case Θ(1).**

**Hence, I am able to achieve the O(n) worst-case for the ADD and FIND. But the O(m) COUNT operations by having two hashtables i.e. CandidateData is only 1000 possible unique values, which is negligible.**

**Though the worst case of HashTable implementations looks O(n) , as mentioned above the average cases of the HashTables are very effective and nearer to Θ(1).**