MTH 4320 Homework 6

Yaohui Wu

March 20, 2024

Problem 1

Solution. If the cell is empty then we can insert with constant time. If the cell is not empty then there is a collision so we need to iterate over the other cells to find an empty one. Therefore, the time complexity of insertion without collision is $O(1)$ and with collision is $O(n)$. If the element is in cell i then we can search in constant time. If it is not in cell i then we need to iterate over the other cells to find it. Therefore, the time complexity of searching without collision is $O(1)$ and with collision is $O(n)$. ■

Problem 2

Solution. The algorithm is:

1. Let H be a hash table with k cells. The time complexity is $O(k)$.
2. For every element in L: Insert the element to the kth cell where k is the key of the element. The time complexity is $O(1)$. If the cell is not empty then chain the element. The time complexity is $O(n)$.
3. Make a new sorted L by appending the elements in every cell of H in order. We have n elements and k keys so there are at most $\frac{n}{k}$ values in every cell of H. The time complexity is $O(k) \cdot O(\frac{n}{k}) = O(k \cdot \frac{n}{k}) = O(k)$.

The time complexity of the algorithm is $O(n + k)$. ■

Problem 3

Solution. The algorithm using the sliding window approach is:

1. Start from the first element in L and iterate to add every element to the sum until the sum is at least k. This is our sliding window with start as the first element of L and the end is the last elementof the subsequence.
2. Move the start of the sliding window to the right until the sum of the subsequence is less than k.

3. Move the end of the sliding window to the right until the sum of the subsequence is at least k.

4. Repeat steps 2 and 3 until the end of the sliding window reaches the end of L. In these steps, if the subsequences with sum greater than or equal to k are shorter then update the shortest subsequence. At the end we have the shortest subsequence in L that sums to at least k.

We move the start of the sliding window at most n times so it takes $O(n)$ time. We move the end of the sliding window n times so it takes $O(n)$ time. Checking and updating the shortest subsequence in total takes $O(n)$ time and all other operations take $O(1)$ time. The time complexity of the sliding window algorithm is $O(n)$. ■

Problem 4

Solution. The algorithm using the sliding window approach is:

1. Let the beginning price on day one be the start of the sliding window representing the buy price and the beginning price on day two be the end representing the sell price.

2. Move the end of the window from left to right to find the maximum sell price and calculate the profit as the maximum profit.

3. Move the start of the window from left to right until the end of the window to find the minimum buy price and update maximum profit if neccessary.

4. Repeat step 3 until the start of the window is the end of the window.

5. Now we get the possible pair of minimum buy price and the maximum sell price so we have the maximum profit.

The time complexity of the sliding window algorithm is $O(n)$. ■