CSC111 Assignment 1: Linked Lists

Zaina Azhar, Katherine Luo

February 4, 2021

Part 1: Faster Searching in Linked Lists

1. Complete this part in the provided `a1_part1.py` starter file. Do **not** include your solution in this file.
2. Complete this part in the provided `a1_part1_test.py` starter file. Do **not** include your solution in this file.
3. (a) **Running Time Analysis:** `LinkedList` $\rightarrow \Theta(n \cdot m)$

    - Let n $\in N$. Let `linky = LinkedList([0, 1, ..., n - 1])` with a length n. Let `item = n - 1`

    Running time of the `linky.__contains__` method:
    - The while loop takes n iterations, and each iteration takes 1 step. Total = n steps
    - `curr = self._first` takes 1 step
    - $(n-1) \in$ `linky`, therefore the `return False` line never runs
    - The total running time of the `linky.__contains__` method is $n+1$ steps, which is $\Theta(n)$

    Running time of the for loop:
    - The for loop takes m iterations.
    - Each iteration calls `linky.__contains__(n-1)`, which has a running time of $\Theta(n)$
    - Therefore, the for loop takes $n \cdot m$ steps.
    - The total running time would be $n \cdot m$ steps, which is $\Theta(n \cdot m)$

    (b) i. **Running Time Analysis:** `MoveToFrontLinkedList` $\rightarrow \Theta(n+m)$
    - Let $n \in N$. Let `linky = MoveToFrontLinkedList([0, 1, ..., n - 1])` with a length of n
    - Let `item = n - 1`
    - First analyze the running time of the `linky.__contains__(n-1)` call.

    Analysis for the First Search Operation
    - n - 1 is the last node, so the else branch in the `__contains__` method is executed.
    - The while loop takes n iterations to reach n - 1. Each iteration takes constant time, and n - 1 is reassigned to `self._first`. This is a total of n steps.
    - (n - 1) $\in$ `linky`, so the `return False` line never runs.
    - The if and elif branches each take 1 step to evaluate, for a total of 2 steps.
    - Therefore for the first search operation, it takes n + 2 steps, which is $\Theta(n)$.

    Analysis for the Subsequent Search Operations
    - n - 1 has been moved to `self._first` after the first search operation
    - The if condition is evaluated and takes 1 step.
    - The elif condition is first evaluated and then returns True, which takes 2 steps.
    - The else branch never executes since n - 1 is now `self._first`.
    - Therefore for subsequent search operations, it takes 3 steps, which is $\Theta(1)$.

- This is the running time for each iteration of the for loop, excluding the first one.

Running Time of the For Loop

- For iteration 1, the operation in the for loop has a running time of $\Theta(n)$, or n steps.
- For subsequent iterations (m - 1 iterations), the operation in the for loop has a constant running time, which repeats (m - 1) times for a total of (m - 1) steps
- The total running time is $n + (m - 1)$, which is $\Theta(n + m)$

ii. **Running Time Analysis:** `SwapLinkedList`

- Let $n \in N$. Let `linky = MoveToFrontLinkedList([0, 1, ..., n - 1])` with a length of n
- Let `item = n - 1`
- The running time of Heuristic 2 can be split into two cases: when $m \leq n$, and when $m > n$

Case 1: *Running time analysis when $m \leq n$*

- n - 1 is the last node, so the else branch in the `__contains__` method is executed.
- The assignment statement takes 1 step.
- The while loop iterates less and less each time `__contains__` is called, as n - 1 moves up in the list with each iteration. Thus, the while loop iterates (n - i) times, where i indexes through the list starting at 1 (for n - 1). Each iteration takes constant time, for a total of (n - i) steps.
- The for loop then iterates m times; i increases until it reaches m, as we know $m \leq n$.
- Steps taken by the for loop:

$$\sum_{i=1}^{m}(n - i) \tag{1}$$

$$= n \cdot m + \frac{-m^2 - m}{2} \tag{2}$$

$$= nm - \frac{1}{2} \cdot (m^2 - m) \tag{3}$$

- Thus, when $m \leq n$, the running time of this loop is $(nm - \frac{1}{2} \cdot (m^2 - m)) + 1$, which is $\Theta(nm - m^2)$.

Case 2: *Running time analysis when $m > n$*

- Similar to Case 1, for each iteration of the for loop, the while loop in the `__contains__` method takes (n - i) steps, where i indexes through the list starting at 1 (for n - 1).
- Since $n < m$, these steps will iterate n times until the last node is shifted to the beginning of the list.
- Steps taken by the for loop:

$$\sum_{i=1}^{n}(n - i) \tag{4}$$

$$= \frac{n^2 - n}{2} \tag{5}$$

- However, since $n < m$, any iterations after m has reached the length of the list n will be constant time. This is because (n - 1) has moved to the first node.
- So for any (m - n) iterations, it would take constant time as the elif branch would execute and return. Thus, this is $(m - n)$ steps.
- The total running time is $\frac{n^2 - n}{2} + m - n$ steps, which is $\Theta(n^2 + m)$

iii. **Running Time Analysis:** `CountLinkedList` $\rightarrow \Theta(n + m)$

- Let $n \in N$. Let `linky = CountLinkedList([0, 1, ..., n - 1])` with a length of n
- Let `item = n - 1`
- First analyze the running time of the `linky.__contains__(n-1)` call.

Analysis for the First Search Operation

- n - 1 is the last node, so the else branch in the `__contains__` method is executed.
- The while loop takes n iterations to reach n - 1. Each iteration takes constant time. The access count of n - 1 increases and thus is reassigned to `self._first`. This is a total of n steps.
- (n - 1) $\in$ `linky`, so the `return False` line never runs.
- The if and elif branches each take 1 step to evaluate, for a total of 2 steps.
- Therefore for the first search operation, it takes n + 2 steps, which is $\Theta(n)$.

Analysis for the Subsequent Search Operations

- n - 1 has been moved to `self._first` after the first search operation
- The if condition is evaluated and takes 1 step.
- The elif condition is first evaluated, increases the access count, and then returns True. This takes 3 steps.
- The else branch never executes since n - 1 is now `self._first`.
- Therefore for subsequent search operations, it takes 4 steps, which is $\Theta(1)$.
- This is the running time for each iteration of the for loop, excluding the first one.

Running Time of the For Loop

- For iteration 1, the operation in the for loop has a running time of $\Theta(n)$, or n steps.
- For subsequent iterations (m - 1 iterations), the operation in the for loop has a constant running time, which repeats (m - 1) times for a total of (m - 1) steps
- The total running time is $n + (m - 1)$, which is $\Theta(n + m)$

4. a) Let lst be a list of numbers from 0, 1, 2, .., n - 1. Consider the following m sequences of search operations on lst, where $m > n$:

```
n = lst.__len__
for _ in range(1, m + 1):
    if m <= n:
        lst.__contains__(n - m)
    else:
        lst.__contains__(0)
```

b) **Running Time Analysis for Heuristic 1:** `MoveToFrontLinkedList`

- Let `lst = MoveToFrontLinkedList([0, 1, 2, ..., n - 1])` be a list of length n

If Branch

- For each iteration of the for loop, the if condition will execute for the until $m = n$.
- For the first call to `lst.__contains__(n - m)`, the last node (n - 1) is called and moved to the front. The (n - 2) node is now the last node.
- Each subsequent call to `lst.__contains__(n - m)` will once again call the last node, shift it to the beginning of the list, and make the (n - 2) node go to (n - 1).
- Therefore, the each time the method is called, n nodes are traversed.
- This iterates until $m = n$, or until the entire list has been looped through. This results in n iterations, which traverse n nodes each time, for a total of n^2 steps

Else Branch

- Once $m > n$, the else branch will execute. This occurs after n iterations, thus the else branch is iterated through $(m - n)$ times.
- For each iteration, `lst.__contains__(0)` is called. After the first if branch iterations, the entire list is shifted over until it is ordered in the same way it started.
- Calling `lst.__contains__(0)` will call the first node in the list. This executes the elif branch in the `__contains__` method, which takes constant time. This gves a total of $(m - n)$ steps.

Total Running Time

- Let T_1 represent the total running time of Heuristic 1
- Ignoring constant factors, the total running time is $T_1 = n^2 + (m - n)$.

b) **Running Time Analysis of Heuristic 2:** `SwapLinkedList`

- Let `lst = SwapLinkedList([0, 1, 2, ..., n - 1])` be a list of length n. Assume n is even.

If Branch

- For each iteration of the for loop, the if condition will execute for the until $m = n$.
- For the first call to `lst.__contains__(n - m)`, the last node (n - 1) switches with the second last node (n - 2).
- The next call to `lst.__contains__(n - m)` will now call the second last node and swap it back to its original spot at the end of the list.
- Following this pattern, the actual number of nodes traversed decreases with every other iteration. After two consecutive nodes swap with each other, the `lst.__contains__(n - m)` call moves further up the list.
- Thus, the number of nodes traversed over all iterations of the if branch can be represented as:

$$\sum_{i=0}^{\frac{n}{2}} 2(n - 2i) \tag{6}$$

$$= 2 \cdot \sum_{i=0}^{\frac{n}{2}} (n - 2i) \tag{7}$$

$$= 2n - \frac{n(n+2)}{2} + n^2 \tag{8}$$

- Thus, the if branch takes $2n - \frac{n(n+2)}{2} + n^2$ steps.

Else Branch

- Once $m > n$, the else branch will execute. This occurs after n iterations, thus the else branch is iterated through $(m - n)$ times.
- For each iteration, `lst.__contains__(0)` is called. After the first if branch iterations, the entire list is swapped back and forth until it is ordered in the same way it started.
- Calling `lst.__contains__(0)` will call the first node in the list. This executes the elif branch in the `__contains__` method, which takes constant time. This gves a total of $(m - n)$ steps.

Total Running Time

- Let T_2 represent the total running time of Heuristic 2
- Ignoring constant factors, the total running time is $T_2 = 2n - \frac{n(n+2)}{2} + n^2 + (m - n)$.

c) **Comparing T_1 and T_2**

- We will compare the running times T_1 and T_2 to determine whether $T_1 - T_2 \notin \mathcal{O}(1)$.

$$T_1 - T_2 \tag{9}$$

$$= (n^2 + (m - n)) - (2n - \frac{n(n+2)}{2} + n^2 + (m - n)) \tag{10}$$

$$= n^2 - 2n + \frac{n(n+2)}{2} - n^2 \tag{11}$$

$$= \frac{n(n+2)}{2} - 2n \tag{12}$$

$$= \frac{n^2 + 2n - 4n}{2} \tag{13}$$

$$= \frac{n^2 - 2n}{2} \tag{14}$$

- Thus, $T_1 - T_2 \in \mathcal{O}(n^2)$, and $\notin \mathcal{O}(1)$.

Part 2: Linked List Visualization

Complete this part in the provided `a1_part2.py` starter file. Do **not** include your solution in this file.