ASSIGNMENT COVERSHEET

PRAGUECITY UNIVERSITY

Student Name: Daniel Shutov	
Class: Algorithms and Data Structures (FT/BL)	
Assignment: Searching and iterating lists	
Lecturer: Dominik Pantůček	Semester: 2201
Due Date: 2022-04-27	Actual Submission Date: Submission date

Evidence Produced (List separate items)	**Location (Choose one)**	
	X	Uploaded to the Learning Center (Moodle)
		Submitted to reception
Note: Email submissions to the lecturer are not valid.		

Student Declaration:	
I declare that the work contained in this assignment was researched and prepared by me, except where acknowledgement of sources is made. I understand that the college can and will test any work submitted by me for plagiarism. **Note:** The attachment of this statement on any electronically submitted assignments will be deemed to have the same authority as a signed statement	
Date: March 3, 2023	Student Signature: Daniel Shutov

A separate feedback sheet will be returned to you after your work has been graded.
Refer to your Student Manual for the Appeals Procedure if you have concerns about the grading decision.

Student Comment (Optional)
Was the task clear? If not, how could it be improved?
Was there sufficient time to complete the task? If not, how much time should be allowed?
Did you need additional assistance with the assignment?
Was the lecturer able to help you?
Were there sufficient resources available?
How could the assignment be improved?

Single linked lists

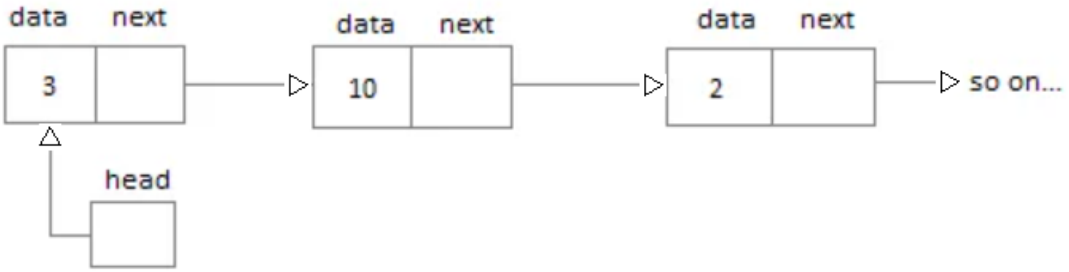
Daniel Shutov

March 3, 2023

Abstract

A computer program can be used in a variety of ways to solve an issue. A Linked List, for example, is a chain-like data structure in which nodes are connected to each other, creating a linked effect. Because linked list is an abstract data type, when the nodes are connected with only the next reference, the list is known as a singly linked list, and those nodes can be accessed sequentially. A process or formula for solving a certain problem is referred to as an algorithm. Which algorithm should be used to tackle a certain problem is the question.

Figure 1: Introduction-to-linked-list



Contents

List of Figures

Listings

1 List of symbols and list of acronyms

Θ : A theoretical measure of the execution of an algorithm, usually the time or memory needed, given the problem size n, which is usually the number of items. Informally, saying some equation $f(n) = (g(n))$ means it is within a constant multiple of g(n) (Black (2016)).
O : Upper Bound: The maximum time a program can take to produce outputs, expressed in terms of the size of the inputs (worst-case scenario). (Singh (2021))
Ω : Lower Bound: The minimum time a program will take to produce outputs, expressed in terms of the size of the inputs (best-case scenario).(Singh (2021))
Node : a place where things such as lines or systems join (Cambridge (2021))
Nil: is a representation of nothing. Its underlying representation is JavaScript's null, and is equal to JavaScript's undefined when compared. (Hickey (2020))

2 Introduction

This technical report will discuss single linked lists and their time as well as complexity as well as, algorithms that were chosen, basic programming in Clojure and what steps should be done before solving the problem that were provided in this technical report, no exposition is required, just logic and understanding basic concepts. The assignment was completed only by using the knowledge that was gained during the classes.

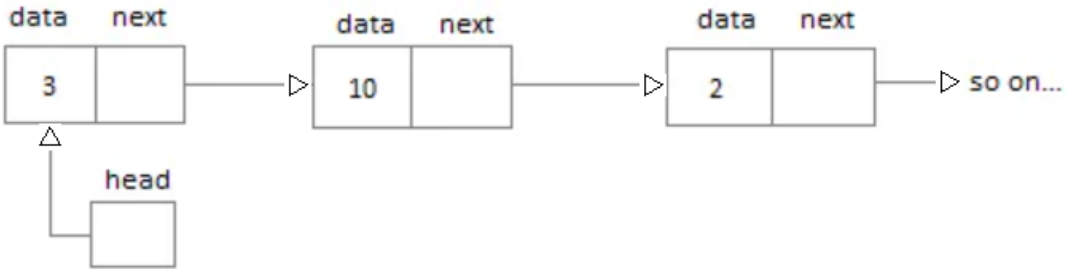
3 Understanding of single linked list

Before beginning, it is necessary to realize that a Linked List is a relatively common linear data structure that consists of a succession of nodes. Each node has its own data and a reference to the preceding node; together, they create a chain-like structure or group of nodes. As seen in Figure 2, a node is composed of two components: data and a reference to the next node. The data component includes the value for the node, while the reference component holds the reference to the next node if there is none (figure 2), and it will be assigned the nil value. Additionally, linked lists are used to form trees and graphs; additionally, they are utilized to implement files and storage, which are considered essential requirements for the daily usage of computers.

Implementation of Linked List in real life according to (Rana (2021)) :

- The time-sharing problem used by the scheduler during the scheduling of the processes in the operating system.
- A list of objects in a 3D game needs to be rendered to the screen.
- Previous and next page in a web browser – one can access the previous and next URL searched in a web browser by pressing the back and next buttons, since they are linked as a linked list.

Figure 2: Full diagram single linked list



3.1 Types of Linked Lists:

Linked lists can be presented as the following four items (Lucknow (2021)):

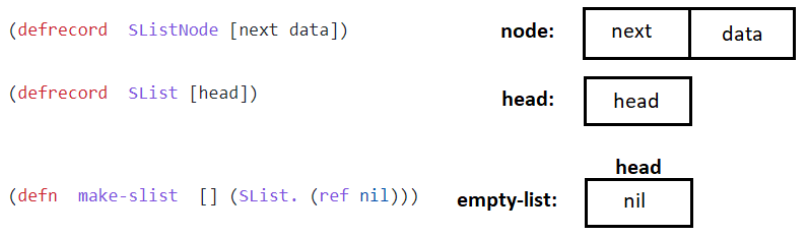
- Single Linked List.
- Double Linked List.
- Circular Linked List.
- Circular Double Linked List.

3.2 Creation of a single linked list

Before the problem could be solved, first the head and node records will be defined, which will enable us to access the linked list's additional nodes. The next lines of code create the structure; more precisely, one generates the empty list that is required in both issues.

- Creating a record for a node, the next reference, which will refer to the list and data item's next node. This type of structure is referred to as a self-referential structure.
- Defining the head.
- Defining a new list that contains only the head node that is referring to nil

Figure 3: Creation of a single linked list



3.3 Inserting new nodes to list

There are three methods to insert new nodes into a single linked list: at the beginning (the preferable and the easiest method), after the first node at any place but the references should be updated but if it's at the end the next should be referring to nil; this step is required in both cases.
Time and memory complexity: O(1) (constant)

Listing 1: Prepending a node inside a Single Linked list

```
1 (defn slist-prepend! [lst val]
2   (dosync
3   (ref-set (:head lst) (SListNode. (ref (deref (:head lst))) val))) val)
```

Figure 4: Prepending to Single linked list

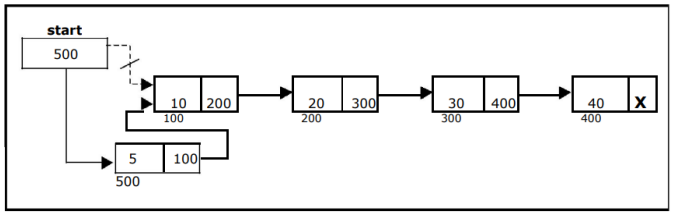


Figure 5: Appending to single linked list

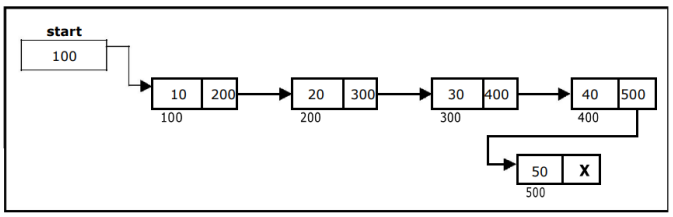
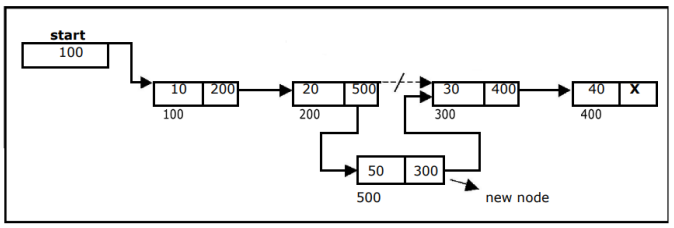


Figure 6: Inserting in the middle to a single linked list



4 Problem 1: Finding an element with a given numeric value

4.1 Recommended algorithm

Because the one is using a single linked list structure, the algorithm for this problem is linear complexity; this means that every time one wants to check if the value provided by the user is in the single linked list, one must iterate through the entire list; in the best case, the one will find the value immediately; in the worst case, the one will not find it at all. To find the value, we want to loop through the list from the first to the last node. The one will start with the head node and verify whether the data in each node is identical to the user value (data = value); otherwise, the value does not exist inside the list.

4.2 Justification

Because the task expressly requires the usage of single linked lists, it cannot be accomplished any better using this data structure.

4.3 Notations

Figure 7: Linear table

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)

4.3.1 Big O:

The big O represents the upper bound of the running time of an algorithm's, that means that it represents the worst case scenario for the time and space complexity of the algorithm.
In our case, the time complexity for our upper bound is: O(n) (linear)
Notation: $\exists n0 \in N : \forall n > n0 : t > k1 \times f(n)$
memory complexity for our upper bound is: O(1) (constant)

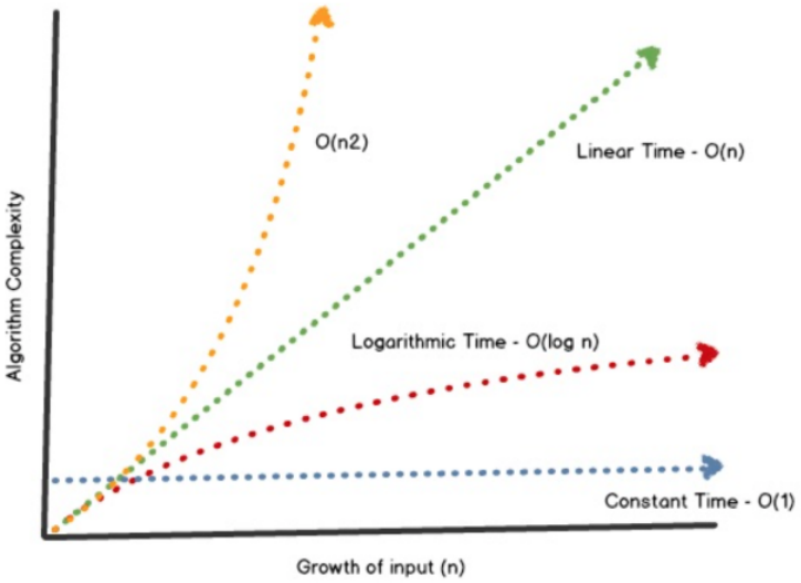
4.3.2 Big Ω:

The big Ω represents the lower bound of the running time of an algorithm's, that means that it represents the best case scenario for the time and space complexity of the algorithm.
In our case, the time complexity for the lower bound is: O(1) (constant)
Notation: $\exists n0 \in N : \forall n > n0 : t < k2 \times f(n)$
memory complexity for our lower bound is: O(1) (constant)

4.3.3 Big theta:

The big theta represents both lower and upper bound of the running time of an algorithm's, that means that it represents the average of the time and space complexity of the algorithm. If the lower and upper bound are the same, that means that the given resource consumption is tightly bound to the given function, else it does not apply. The running time is not tightly bound
In our case, the time complexity for our Big theta is: Ω != O $\Longrightarrow$ Big Theta does not apply.
Notation: $\exists n0 \in N : \forall n > n0 : t > k1 \times f(n) t < k2 \times f(n)$
memory complexity for Big theta is: Ω = O therefor Big Theta = 1, is tightly bound.

Figure 8: Graph representation



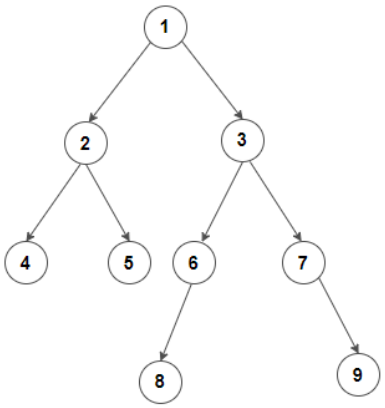
4.4 Evaluation of efficiency

Single linked lists can be good in this case for small values, but as the value grows, then the algorithm time complexity grows as well. The algorithm could be approved, it would be better to use balanced Binary search tree. A binary tree has a better time complexity for searching O(log N) but in the worst case can be the same as a linked list O(n). This means searching a binary tree will (in most cases) be faster than searching a linked list as one see it would be more efficient. Provided that one is only given a single linked list, this is the optimal option.

4.4.1 Binary search tree

A binary search tree is a non-linear data structure made of nodes, each of which can have numerous child nodes and each of which can contain data. The root node of a tree is the initial node; nodes on the same level that share the same parent are referred to as children; in a binary tree, each node might have zero, one, or two children. A tree's rule is that the value of the left side node is less than the value of the right side node. As long as the tree is balanced, the search path to each item in the tree is significantly shorter than in a linked list. In terms of time complexity, the complexity will be logarithmic (O($logn$)).(Krijthe (2008))

Figure 9: Binary search tree



Searchpaths by (Krijthe (2008)):

value	List	Tree
1	1	3
2	2	2
3	3	3
4	4	1
5	5	3
6	6	2
7	7	3
avg	4	2.43

By larger structures, the average search path becomes significant smaller, easily counted by using "(. System (nanoTime))" from java.

Items	List	Tree
1	1	1
3	2	1.67
7	4	2.43
15	8	3.29
31	16	4.16
63	32	5.09

4.5 Solution for problem 1

Listing 2: Program finds numerik value

```
1 (defn find_node [lst val]
2   (loop [node (:head lst)
3          v val]
4     (if (nil? node)
5       false
6       (if (= (:data @node) v)
7         true
8         (recur (:next @node) v)))))
```

5 Problem 2: Listing the Cartesian product of the list with itself

5.1 Recommended algorithm

Because a single linked list structure is used, the algorithm for this issue has quadratic complexity. The Cartesian product of the list with itself will be printed using two methods. The first function will loop recursively over the single linked list, passing the "base" number (base, other number) and the list itself to a helper function until the list head attribute refers to nil. Within the helper function, the "base" number (base, other number) will be printed alongside each number that the node holds throughout the list. The function will then loop recursively and print the Cartesian product of the list with itself until the node's (head) attribute refers to nil.

5.2 Justification

As the task asks all the pairs should be printed, in other words: One must iterate the single linked list, and one must set the base number for each loop that is supplied to the "helper" function, which will loop recursively and print the "base" number with each piece of data contained in the node until the list is locally empty. The operation will be repeated until the head attribute of the main function is set to nil, which indicates that all possible sequences have been written in pairs. It cannot be improved, even with a different data structure.

5.3 Notations

Figure 10: Time complexity

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)

5.3.1 Big O:

The big O represents the upper bound of the running time of an algorithm's, that means that it represents the worst case scenario for the time and space complexity of the algorithm.
In our case, the time complexity for our upper bound is: $O(n^2)$ (quadratic)
Notation: $\exists n0 \in N : \forall n > n0 : t > k1 \times f(n)$
memory complexity for our upper bound is: O(1) (constant)

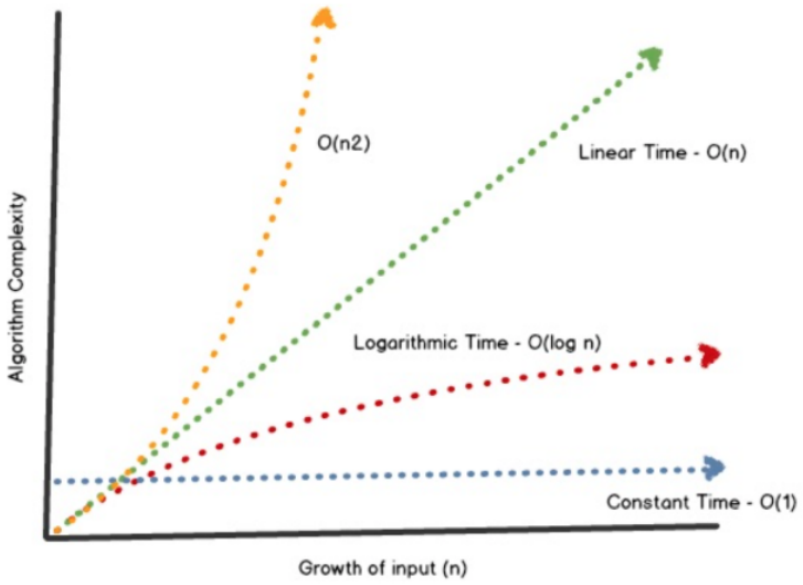
5.3.2 Big Ω:

The big Ω represents the lower bound of the running time of an algorithm's, that means that it represents the best case scenario for the time and space complexity of the algorithm.
In our case, the time complexity for our lower bound is: $O(n^2)$ (quadratic)
Notation: $\exists n0 \in N : \forall n > n0 : t < k2 \times f(n)$
memory complexity for our lower bound is: O(1) (constant)

5.3.3 Big theta:

The big theta represents both lower and upper bound of the running time of an algorithm's, that means that it represents the average of the time and space complexity of the algorithm. If the lower and upper bound are the same, that means that the given resource consumption is tightly bound to the given function, else it does not apply.
In our case, the time complexity for our Big is: (n^2) $(\Omega = \mathrm{O} \implies Theta(\mathrm{n}^2))$
Notation: $\exists n0 \in N : \forall n > n0 : t > k1 \times f(n) t < k2 \times f(n)$
memory complexity for Big theta is: $\Omega = \mathrm{O} \implies$ Big Theta = 1, is tightly bound.

Figure 11: Graph representation (Azar and Alebicto 2016).



5.4 Evaluation of efficiency

Single linked lists are the appropriate data structure for this task, since they allow one to loop over the list and print all the pairs easily, in other words an iteration inside an iteration that prints all the pairs.

5.5 Solution for problem 2

Listing 3: Listing the cartesian product with itself

```
1
2  (defn  right-num-helper [base node]
3      (when (not(nil? node))
4        (println "(" (:data base) "," (:data node) ")")
5        (right-num-helper base @(:next node))))
6
7  (defn  left-num [lst]
8    (loop [node @(:head lst)
9           node2 @(:head lst)]
10     (if (not (nil? node))
11       (do
12         (right-num-helper node node2)
13         (recur @(:next node) node2)))))
14
15 line 3: ;;if the node is not referring to nil keep going
```

```
16 line 4: ;;print the base number with nodes data from the list
17 line 5: ;; recurcive loop for the all the nodes numbers (right side)
18 line 8: ;; setting the base number (left side)
19 line 9: ;;setting the first node to be used for the helper func
20 line 10: ;; if the list is not empty, keep running
21 line 12: ;;sending to the helper function the first node of the list
22 line 13: ;; recurcive loop for the base number (left side)
```

6 List of figures

References

Azar, Erik and Mario Eguiluz Alebicto (2016). *Swift Data Structure and Algorithms*. Packt Publishing Ltd.

Black, Paul E. (June (2016)). "Dictionary of Algorithms and Data Structures". en. In: URL: https://xlinux.nist.gov/dads/HTML/theta.html#:~:text=Definition%3A%20A%20theoretical%20measure%20of,multiple%20of%20g(n)..

Cambridge, university (June (2021)). "Meaning of node in English". en. In: URL: https://dictionary.cambridge.org/dictionary/english/node.

Hickey, Rich (May (2020)). "nil". en. In: URL: https://cljs.github.io/api/syntax/nil.

Krijthe, Toon ((2008)). "Difference between a LinkedList and a Binary Search Tree". en. In: URL: https://stackoverflow.com/questions/270080/difference-between-a-linkedlist-and-a-binary-search-tree/.

Lucknow, University of (June (2021)). "LINKED LISTS". en. In: URL: https://www.lkouniv.ac.in/site/writereaddata/siteContent/202003251324427324himanshu_Linked_List.pdf.

Rana, Rohit (June (2021)). "Where are linked lists used in real life?" en. In: URL: https://www.quora.com/Where-are-linked-lists-used-in-real-life.

Singh, Ashutosh ((2021)). "Upper bound vs lower bound". en. In: URL: https://www.quora.com/In-algorithms-what-is-the-upper-and-lower-bound.

Studytonight.com (June (2021)). "Introduction to Linked List". en. In: URL: https://www.studytonight.com/data-structures/introduction-to-linked-list.

Annexe

Full clojure code

Listing 4: Full solution

```
1
2  (defn find_node [lst val]
3    (loop [node (:head lst)
4           v val]
5      (if (nil? node)
6        false
7        (if (= (:data @node) v)
8          true
9          (recur (:next @node) v)))))
10
11
12
13 (defrecord  SListNode [next data])
14 (defrecord  SList [head])
15 (defn  make-slist  [] (SList. (ref nil)))
16
17 (defn slist-empty? [lst] (nil? (deref (:head lst))))
18
19 (defn slist? [lst] (= (class lst) SList))
20
21 (defn  slist-prepend! [lst val]
22   (dosync (ref-set  (:head  lst) (SListNode. (ref (deref  (:head  lst))) val))) val)
23
24
25 (defn  right-num-helper [base node]
26     (when (not(nil? node)) ;;if the node is not refering to nil keep going
27       (println "(" (:data base) "," (:data node) ")") ;;print the base number with nodes dat
28       (right-num-helper base @(:next node)))) ;; recurcive loop for the all the nodes numbe
29
30 (defn  left-num [lst]
31   (loop [node @(:head lst) ;; setting the base number (left side)
32          node2 @(:head lst)] ;;setting the first node to be used for the helper func
33     (if (not (nil? node)) ;; if the list is not empty, keep running
34       (do
35         (right-num-helper node node2) ;;sending to the helper function the first node of the
36         (recur @(:next node) node2)))) ;; recurcive loop for the base number (left side)
37
```

```
38 (let [a1 (make-slist)]
39   (slist-empty? a1)
40   (slist? a1)
41   (slist-prepend! a1 10)
42   (slist-prepend! a1 1)
43   (slist-prepend! a1 3)
44   (slist-prepend! a1 4)
45   (slist-prepend! a1 6)
46   (slist-prepend! a1 8)
47   (left-num a1)
48   (find_node a1 88))
```