Kristianstad University
SE-291 88 Kristianstad
+46 44-250 30 00
www.hkr.se

Seminar 2: Lists, Stacks, Queues

Course : Algorithms and Data Structures
(DA256F)
Meenu Gupta

Content

1 Introduction

1.1 Background/idea

The Seminar is related to Lists, Stack, Queues. It consists of three tasks:

Task 1 : Writing Code to implement a queue using stacks & stack using queue.

Task 2 : Creating a LinkedList manually without using build in implementations.

Task 3 : Solving Josephus problem using various methods.

1.2 Purpose

Purpose of the report:

- To figure out which implementation is the most efficient for Task1.
- To measure the running time depending on different data structures for Task 3.
- Present the results in a table and graph for Task 3.
- Analyze the results for Task 1 & Task 3.

2 Method

The code developed for all tasks is written in Java language.

3 Results

3.1 Task 1 Results:

Results for Task 1 provides the reflection on efficiency of different routines by measuring time complexity of each method/function in code. Code has been developed for applying four routines:

a) Code to implement a queue using two stacks and one stack with methods enqueue, dequeue & peek. Time complexity for different methods:

Method	**Time Complexity (using two stack)**	**Time Complexity (using one stack)**
enqueue()	O(1)	O(1)
dequeue()	O(n)	O(n)
peek()	O(n)	O(n)

b) Code to implement a stack using two queues and one queue with method push, pop & peek. Time complexity for different methods:

Method	**Time Complexity (using two queues)**	**Time Complexity (using two queues)**
push()	O(1)	O(n)
pop()	O(n)	O(1)
peek()	O(n)	O(1)

3.2 Task 1 Analysis:

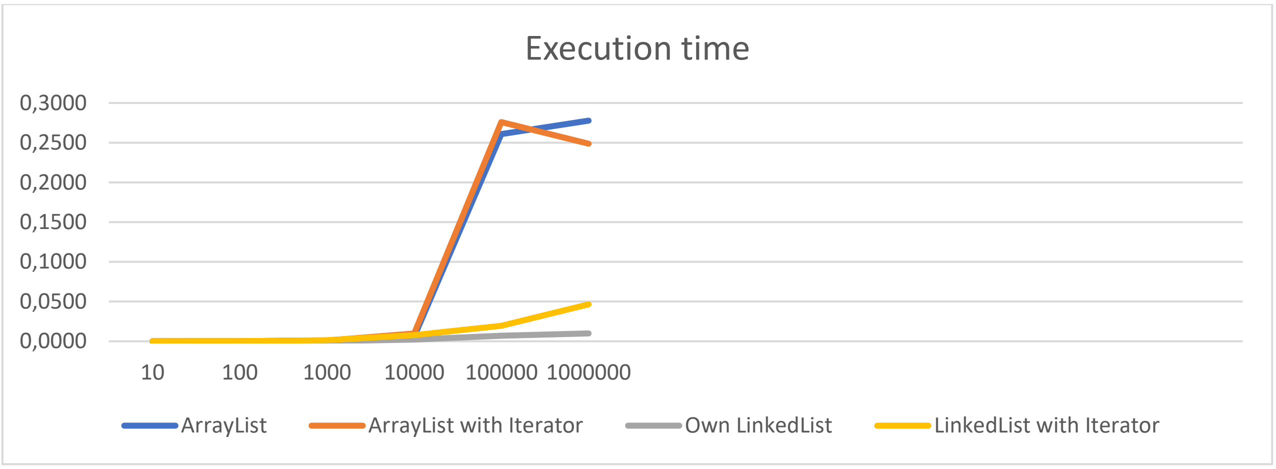
a) Queue using Stacks: Operations like dequeue() & peek() are not as efficient as the built-in queue classes like LinkedList provided by java which provides time complexity of O(1) for all operations. In fact, queue using one stack provide recursion overhead too making it less efficient.

b) Stacks using queues: Operations push(), pop() & peek() has shown different time complexity using two queues & one queue which are not as efficient as the built-in Stack class provided by java which provides time complexity of O(1) for all operations. Stack using two queues also increases the space complexity as two queues are used. The built-in stack implementations are efficient because they use vector/dynamic array as the underlying structure.

3.3 Task 3 Results:

Results for Task 3 provides the running time of the program depending on different data structures:

M=3, Time in Seconds (Rounded up to 4 digits after decimal)				
Input (N)	**ArrayList**	**ArrayList with Iterator**	**Own LinkedList**	**LinkedList with Iterator**
10	0,0000	0,0000	0,0000	0,0000
100	0,0003	0,0002	0,0000	0,0002
1000	0,0009	0,0011	0,0002	0,0014
10000	0,0057	0,0100	0,0023	0,0078
100000	0,2609	0,2760	0,0070	0,0196
1000000	0,2778	0,2489	0,0099	0,0465



3.4 Task 3 Analysis:

After analysing the table and graph for Task 3, we can see that for small values like 10 to 1000 it does not make a huge difference in the performance. However, when input size exceeds 1000, then own LinkedList provides the most efficient operation.

“ArrayList with Iterator” and ”LinkedList” with Iterator take longer than direct implementation of ArrayList & Own LinkedList. While using the iterator, every time the value of n has reached the end, then iterator has to start traversing the list from beginning. This could be inefficient for larger input size.

Reflection on When to Use LinkedList vs ArrayList:

ArrayList are more efficient when random access and where elements need to be removed based on index. It is efficient when elements need to be added at the end of the list. However, if you want to remove the insert and delete the elements frequently at arbitrary positions then LinkedList is better. As in that scenario, ArrayList will lead to the time complexity of O(N) as compared to LinkedList which has time complexity of O(1).

LinkedList stores the additional reference to next and previous element resulting in more memory overhead as compared to LinkedList.

6 Conclusion:

Task 1 – in real world scenarios there are huge amount of data where all the operations are used frequently Stack & Queues are better than our implementation as they provide time complexity O(1) for all operations.

Task 3 – LinkedList performs better than ArrayList when frequent insertions and deletions is needed at arbitrary position.

This seminar has taught me how different data structures can impact the performance of programs in terms of time complexity and space complexity. So, it’s important to choose the right data structure based on requirements of application.

7 References :

[1] Weiss, Mark. A. (2012), Data structures and algorithm analysis in Java. 3rd edition Harlow, Essex : Pearson. (632 p).