OBSERVACIONES DE LA PRÁCTICA

Samuel Villamil Álvarez - 202421118

# Ambientes de pruebas

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Máquina 1** | **Máquina 2** | **Máquina 3** |
| **Procesadores** | Intel(R) Core(TM) i7-10610U CPU @ 1.80GHz 2.30 GHz |  | AMD Ryzen 5 3600  3,6 GHz – 4,2GHz |
| **Memoria RAM (GB)** | 16,0 GB (15,7 GB usable) |  | 16 GB |
| **Sistema Operativo** | Windows 11 |  | Windows 11 |

*Tabla 1. Especificaciones de las máquinas para ejecutar las pruebas de rendimiento.*

# Máquina 1

Resultados para Queue con Array List

|  |  |  |  |
| --- | --- | --- | --- |
| **Porcentaje de la muestra [pct]** | **enqueue (Array List)** | **dequeue (Array List)** | **peek (Array List)** |
| 0,50% | 0.050 | 0,053 | 0,005 |
| 5,00% | 0.388 | 0,447 | 0,004 |
| 10,00% | 0.828 | 0,919 | 0,002 |
| 20,00% | 1.272 | 1.627 | 0.003 |
| 30,00% | 2.283 | 3.441 | 0.002 |
| 50,00% | 3.857 | 8.537 | 0,005 |
| 80,00% | 5.199 | 10.922 | 0.004 |
| 100,00% | 7.38 | 20.026 | 0,003 |

Resultados para Stack con Array List

|  |  |  |  |
| --- | --- | --- | --- |
| **Porcentaje de la**  **muestra** | **push (Array List)** | **pop (Array List)** | **top(Array List)** |
| **0.50%** | 0.015 |  | 0.002 |
| **5.00%** | 0.006 |  | 0.002 |
| **10.00%** | 0.007 |  | 0.001 |
| **20.00%** | 0.008 |  | 0.002 |
| **30.00%** | 0.007 |  | 0.001 |
| **50.00%** | 0.009 |  | 0.002 |

|  |  |  |  |
| --- | --- | --- | --- |
| **80.00%** | **0.015** |  | **0,001** |
| **100.00%** | **0.036** |  | **0,003** |

Resultados para Queue con Linked List

|  |  |  |  |
| --- | --- | --- | --- |
| **Porcentaje de la muestra [pct]** | **enqueue (Linked List)** | **dequeue (Linked List)** | **peek (Linked List)** |
| 0,50% | **0,009** | **0,002** | **0,004** |
| 5,00% | **0,039** | **0,002** | **0,006** |
| 10,00% | **0,007** | **0,001** | **0,002** |
| 20,00% | **0,009** | **0,002** | **0,003** |
| 30,00% | **0,007** | **0,001** | **0,002** |
| 50,00% | **0,006** | **0,001** | **0,002** |
| 80,00% | **0,007** | **0,001** | **0,003** |
| 100,00% | **0,007** | **0,003** | **0,003** |

Resultados para Stack con Linked List

|  |  |  |  |
| --- | --- | --- | --- |
| **Porcentaje de la muestra [pct]** | **push (Linked List)** | **pop (Linked List)** | **top(Linked List)** |
| 0,50% | **0,445** |  | **0,002** |
| 5,00% | **0,508** |  | **0,003** |
| 10,00% | **0,48** |  | **0,002** |
| 20,00% | **0,444** |  | **0,002** |
| 30,00% | **0,478** |  | **0,001** |
| 50,00% | **0,433** |  | **0,001** |
| 80,00% | **0,44** |  | **0,002** |
| 100,00% | **0,381** |  | **0,002** |

# Máquina 2

Resultados para Queue con Array List

|  |  |  |  |
| --- | --- | --- | --- |
| **Porcentaje de la**  **muestra** | **enqueue (Array List)** | **dequeue (Array List)** | **peek (Array List)** |
| **0.50%** |  |  |  |
| **5.00%** |  |  |  |
| **10.00%** |  |  |  |
| **20.00%** |  |  |  |
| **30.00%** |  |  |  |
| **50.00%** |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **80.00%** |  |  |  |
| **100.00%** |  |  |  |

Resultados para Stack con Array List

|  |  |  |  |
| --- | --- | --- | --- |
| **Porcentaje de la**  **muestra** | **push (Array List)** | **pop (Array List)** | **top(Array List)** |
| **0.50%** |  |  |  |
| **5.00%** |  |  |  |
| **10.00%** |  |  |  |
| **20.00%** |  |  |  |
| **30.00%** |  |  |  |
| **50.00%** |  |  |  |
| **80.00%** |  |  |  |
| **100.00%** |  |  |  |

Resultados para Queue con Linked List

|  |  |  |  |
| --- | --- | --- | --- |
| **Porcentaje de la**  **muestra** | **enqueue (Linked List)** | **dequeue (Linked List)** | **peek Linked List)** |
| **0.50%** |  |  |  |
| **5.00%** |  |  |  |
| **10.00%** |  |  |  |
| **20.00%** |  |  |  |
| **30.00%** |  |  |  |
| **50.00%** |  |  |  |
| **80.00%** |  |  |  |
| **100.00%** |  |  |  |

Resultados para Stack con Linked List

|  |  |  |  |
| --- | --- | --- | --- |
| **Porcentaje de la**  **muestra** | **push (Linked List)** | **pop (Linked List)** | **top(Linked List)** |
| **0.50%** |  |  |  |
| **5.00%** |  |  |  |
| **10.00%** |  |  |  |
| **20.00%** |  |  |  |
| **30.00%** |  |  |  |
| **50.00%** |  |  |  |
| **80.00%** |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **100.00%** |  |  |  |

# Máquina 3

Resultados para Queue con Array List

|  |  |  |  |
| --- | --- | --- | --- |
| **Porcentaje de la**  **muestra** | **enqueue (Array List)** | **dequeue (Array List)** | **peek (Array List)** |
| **0.50%** | **0,151** | **0,132** | **0,008** |
| **5.00%** | **0,856** | **1,109** | **0,013** |
| **10.00%** | **1,498** | **1,411** | **0,007** |
| **20.00%** | **2,703** | **2,981** | **0,005** |
| **30.00%** | **4,18** | **4,506** | **0,006** |
| **50.00%** | **5,564** | **7,201** | **0,005** |
| **80.00%** | **14,989** | **20,134** | **0,007** |
| **100.00%** | **15,347** | **34,687** | **0,011** |

Resultados para Stack con Array List

|  |  |  |  |
| --- | --- | --- | --- |
| **Porcentaje de la**  **muestra** | **push (Array List)** | **pop (Array List)** | **top(Array List)** |
| **0.50%** | **0,089** | **0,079** | **0,005** |
| **5.00%** | **0,813** | **0,751** | **0,006** |
| **10.00%** | **1,952** | **2,276** | **0,004** |
| **20.00%** | **3,098** | **5,009** | **0,051** |
| **30.00%** | **3,151** | **4,203** | **0,007** |
| **50.00%** | **4,503** | **5,803** | **0,009** |
| **80.00%** | **11,213** | **8,203** | **0,004** |
| **100.00%** | **16,899** | **10,432** | **0,013** |

Resultados para Queue con Linked List

|  |  |  |  |
| --- | --- | --- | --- |
| **Porcentaje de la**  **muestra** | **enqueue (Linked List)** | **dequeue (Linked List)** | **peek Linked List)** |
| **0.50%** | **0,131** | **0,084** | **0,005** |
| **5.00%** | **1,688** | **1,674** | **0,005** |
| **10.00%** | **5,911** | **2,529** | **0,009** |
| **20.00%** | **5,551** | **3,188** | **0,005** |
| **30.00%** | **5,603** | **3,801** | **0,003** |
| **50.00%** | **26,123** | **11,433** | **0,008** |
| **80.00%** | **30,477** | **13,012** | **0,005** |

|  |  |  |  |
| --- | --- | --- | --- |
| **100.00%** | **30,567** | **15,398** | **0,011** |

Resultados para Stack con Linked List

|  |  |  |  |
| --- | --- | --- | --- |
| **Porcentaje de la**  **muestra** | **push (Linked List)** | **pop (Linked List)** | **top(Linked List)** |
| **0.50%** | **0,214** | **0,346** | **0,007** |
| **5.00%** | **1,051** | **8,456** | **0,038** |
| **10.00%** | **2,005** | **42,096** | **0,005** |
| **20.00%** | **3,329** | **105,644** | **0,006** |
| **30.00%** | **9,276** | **23,054** | **0,035** |
| **50.00%** | **11,256** | **685,232** | **0,005** |
| **80.00%** | **21,54** | **1609,851** | **0,005** |
| **100.00%** | **29,289** | **2846,908** | **0,007** |

# Preguntas de análisis

1. ¿Se observan diferencias significativas entre las implementaciones con ArrayList y LinkedList para las funciones de Queue y Stack? ¿Cuál es más eficiente en cada operación? ¿Por qué una implementación es más rápida en ciertos casos?

Stack y queue tienen implementaciones basadas en array y en single linked list, y en teoría, o lo que nos driría la lógica en primera instancia sería concluir que para queue, array list sería la más eficiente, o que para stack sería single linked list.

Había funciones como peek o top que eran rápidas de realizar en ambos casos, porque sacan datos directos y fáciles de acceder (peek saca el primer valor de la lista y top el último), cosa que los hace más o menos constantes siempre. Pero para añadir cosas al pricipio, como es el caso de push, es mucho mas eficiente hacerlo saliendo de SSL, ya que ahí la complejidad temporal para añadir al principio es O(1). Cosa que es lo opuesto cuando se quiere añadir o eliminar algo del final de cualquier lista, ahí va a ser O(n).

1. ¿Cuándo es preferible usar ArrayList o LinkedList? Si insertamos y eliminamos con frecuencia,

¿qué estructura conviene más? Si accedemos aleatoriamente a elementos, ¿cuál es más eficiente?

Si estamos accediendo aleatoriamente a elementos dentro de la lista es mejor tener un Array, su complejidad es de O(1), tiene accesos directos y pueden extraerse más fácil, sin necesidad de loops, porque tienen un índice.

Si vamos a estar añadiendo cosas y eliminando en los extremos de la estructura (mientras no sea por dentro de la estructura, entonces da igual cuál sea), es mejor usar una SSL.

1. Durante la ejecución de las pruebas ¿Se presentan anomalías en los tiempos de ejecución que no se explican con la teoría?

Hay varias anomalías en las pruebas, en algunos de los casos, el Array era más eficiente que la SSL para insertar objetos en el principio, cosa que no debería ser posible porque en Array List esto es lineal, podría haber problemas de implementación en SLL que generen este problema y multipliquen innecesariamente la complejidad temporal.

De igual manera, no pudimos con la función POP, por lo que nos faltan datos para concluir correctamente sobre la eliminación al final de las listas.

1. Complete la siguiente tabla de acuerdo con qué operación es más eficiente en cada implementación (marque con una x la que es más eficiente). Adicionalmente, explique si este comportamiento es acorde con lo enunciado en la teoría. Justifique las respuestas.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Array List | Linked List | Justificación |
| **QUEUE** | **Enqueue()** |  | X | No concuerda, en la máquina 1 está presentando tiempos más eficientes en SLL, que se supone, debería ser más lenta. |
| **Dequeue()** | | X | Concuerda, la eliminación o inserción de elementos al principio es más eficiente con una SSL. |
| **Peek()** | X |  | Concuerda, pero es variable, en cualquiera de las estructuras es O(1), y no cambia mucho entre implementaciones. |
| **STACK** | **Push()** | X |  | No concuerda, no debería ser más fácil añadir cosas al principio en un Array. La diferencia no es mucha, pero sí está siendo más rápido en los Array cundo debería ser SLL. |
| **Pop()** |  |  |  |
| **Top()** | | x | Si concuerda, igual que en peek, acceder al primer elemento es O(1) en cualquier implementación. |