Universidad Austral de Chile
Facultad de Ciencias de la Ingeniería
Instituto de Informática

Universidad Austral de Chile
Fac. de Cs. De la Ingeniería

Tarea 1

Implementación de Estructuras y Comparación Empırica.

Integrantes: Nicol Huaiquil
Sebastián Lara
Andrés Torres
Rodrigo Vergara
Profesor(es): Erick Araya
Hector Ferrada
Fecha de entrega: 27 de julio de 2020
Valdivia, Chile

Índice de Contenidos

Índice de Figuras

Índice de Tablas

Índice de Códigos

1. Introducción

Este trabajo consiste en la implementación de estructuras de datos vistas en la asignatura, pero experimentaremos con estructuras propias para esta tarea.

Las estructuras vistas en clases son las siguientes:

1.1. Cola (Queue)

Esta estructura consiste en que el primer elemento añadido es el primero en salir, por ejemplo, una fila de un supermercado, en donde la primera persona en la fila es la primera que sale.

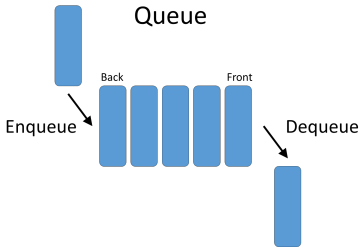

Figura 1: Cola(Queue).

1.2. Lista enlazada

La lista enlazada nos permite agrupar datos de una manera ordenada, en donde se puede recorrer desde el primer elemento de la lista hasta el último gracias a que posee 2 elementos en su interior, un puntero y el contenido. El puntero sólo apunta hacia el nodo siguiente y no al anterior, por lo tanto, no se puede devolver al momento de avanzar en la lista. Otra importante característica es que la lista enlazada va ocupando celdas de una manera dinámica, es decir, a medida que avanza el programa se le va pidiendo memoria al computador, lo que convierte a una lista enlazada en una opción mucho más rápida y económica en tiempo de ejecución .

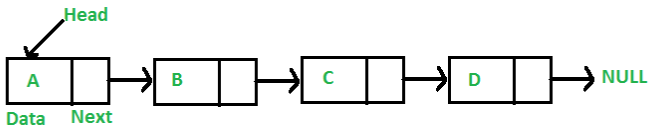

Figura 2: Lista enlazada.

1.3. Lista doblemente enlazada

Una lista doblemente enlazada, al igual que la enlazada simple, sirve para agrupar datos de una manera ordenada pero con la diferencia que ésta además de recorrerla solo hacia adelante, nos

podemos devolver, ya que ésta contiene 3 elementos en cada nodo: Puntero hacia la izquierda, contenido, puntero hacia la derecha. En donde cada puntero tiene la dirección de memoria de la celda asignada tanto la anterior como la siguiente.

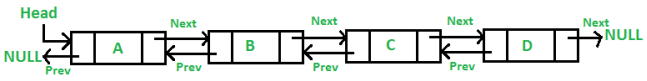

Figura 3: Lista doblemente enlazada.

1.4. Arreglo

Grupo o colección finita, homogénea y ordenada de elementos. Un arreglo es un conjunto de datos o una estructura de datos homogéneos que se encuentran ubicados en forma consecutiva en la memoria RAM (sirve para almacenar datos en forma temporal).

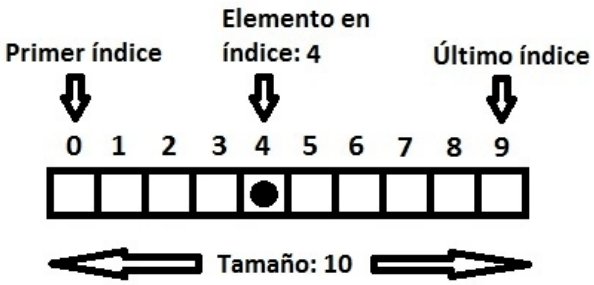

Figura 4: Arreglo.

2. Metodología

2.1. Implementación de las estructuras de datos vistas en clases

Código 1: Cola.

```
class Queue {
   private:
   struct nodeQueue {
      int val;
      nodeQueue* next;
   };
   typedef struct nodeQueue nodoQ;

   nodoQ *Q;  // puntero AL FRENTE de la cola, por donde entran los elementos (enqueue)

   public:
   Queue();
   Queue(int num);
   ~Queue();

   void enqueue(int);
   int dequeue();
   int front();
   int last();
   int size();
   bool isEmpty();
   void display();
};

```

Código 2: Lista enlazada.

```
struct nodeList {
   int val;
   nodeList* next;
};
typedef struct nodeList nodo;

```

Código 3: Lista doblemente enlazada.

```
struct nodeDoubleList {
   int val;
   nodeDoubleList *prev;
   nodeDoubleList *next;
};
typedef struct nodeDoubleList nodoD;

```

Código 4: Arreglo.

```
1       string* A = new string[cap];  //Arreglo de tipo  string  de tamaño cap
2       A[0] = words[x];
3
```

2.2. Estructuras de datos propias para el trabajo

- **Simple Queue:** Es una estructura construida en base al Punto 1.2, y también utilizándola como el Punto 1.1
- **Double Queue:** Es construida en base al Punto 1.3, pero en esta oportunidad agregaremos más variables para que sea mucho más rápido en tiempo de ejecución, para ser explícitos, 2 variables más, las cuales estarán definidas siendo una al principio de la lista doblemente enlazada (la llamaremos L) y uno al final (la llamaremos R). La idea de estas 2 variables es que "L" como se encuentra al principio tendremos más rápido al primer elemento para eliminarlo y "R" al estar al final, podremos agregar más rápido los elementos porque ya sabemos en dónde está el final de la lista doblemente enlazada. Con estos 2 elementos ya definidos, no será necesario recorrer todo el arreglo para ir agregando elementos.
- **Array Queue:** Estructura que implementa el Punto 1.2 en base al Punto 1.4 redimensionable. Inicialmente este arreglo tiene una capacidad *cap*. Además deberá en cada proceso de encolar un nuevo elemento, verificar que hay un espacio disponible en en arreglo, en caso de que no se encuentre espacio disponible, deberá crear un nuevo arreglo con el doble de capacidad (2cap), donde copiará todos los datos del arreglo antiguo, al nuevo y eliminará el arreglo antiguo.

3. Experimentación

3.1. Explicación experimento

Para realizar este experimento usaremos una dataset (archivo .txt) que contiene palabras en inglés, las cuales deberán ser insertadas aleatoriamente en las colas (queues). Además, se nos pide tomar como largos mínimos y máximos definidos antes de la ejecución del programa, para que así las palabras tengan un rango de largo de caracteres. Esta palabra aleatoria deberá ser ingresada en las tres estructuras colas (queues) definidas anteriormente.

Dicho esto, explicaremos en que consiste el experimento:

Para cada estructura creada se ingresará (enqueue) y eliminará (dequeue) los elementos de forma alternada, siguiente estas instrucciones:

1. Crear la cola con un solo datos al azar.
2. Ingresar y eliminar alternadamente en base a la siguiente restricción:
   - Ingresar $2K$ palabras
   - Eliminar $K/2$ palabras Donde K es la cantidad de elementos existentes en la estructura. Inicialmente $K = 1$ (primera palabra aleatoria). Seguido de esto, se debe iterar hasta que K sea igual a n elementos.
3. Luego de haber completado los n elementos, procederemos a eliminarlos de la estructura.

3.2. Ejecución algoritmos

Para ejecutar estos programas usaremos valores (n) significativos, p.e: $10^3, 10^4, 10^5, 10^6$
Luego de ejecutar los programas con esos parametros de entrada, obtenemos la siguiente tabla

Tabla 1: Tabla tiempos de ejecución en segundos

Estructura	n = 10^3	n= 10^4	n = 10^5	n = 10^6
Simple Queue	0.023511	0.597302	77.1085	12499.8
Double Queue	0.016278	0.123674	1.389	15.7352
Array Queue	0.020505	0.212823	2.4435	17.8763

3.3. Gráficos y ecuaciones en base al tiempo de ejecución y número de elementos

A partir de esta información, creamos gráficos por cada estructura para ver su tendencia según la cantidad de elementos (n) y el tiempo de ejecución de dicha estructura. Los gráficos nos muestran una linea de tendencia de forma punteada, la cual nos entrega una ecuación donde n es el número de elementos, en base a esta ecuación podemos experimentar con diversos valores de n para así analizar con mayor profundidad el funcionamiento de estos algoritmos.

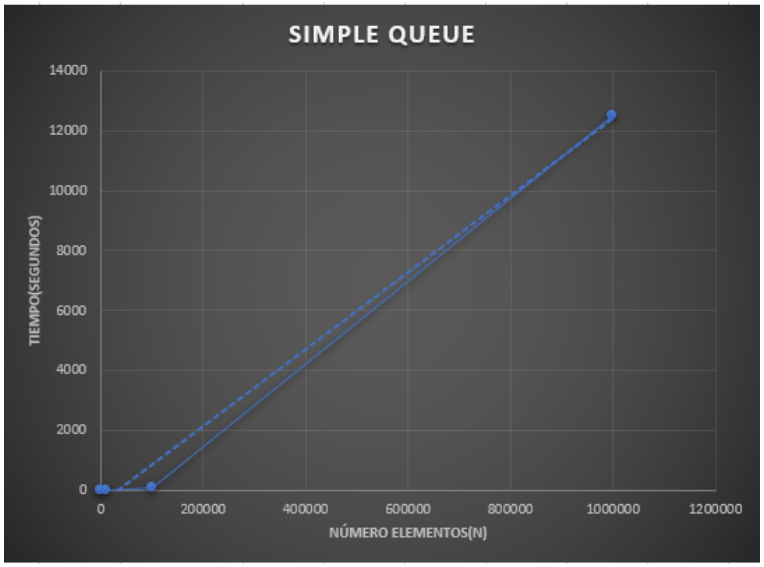

Figura 5: Tiempo ejecución Simple Queue.

$$f(n) = 0.0128 * n - 424.53 \quad (1)$$

Ecuación para Simple Queue

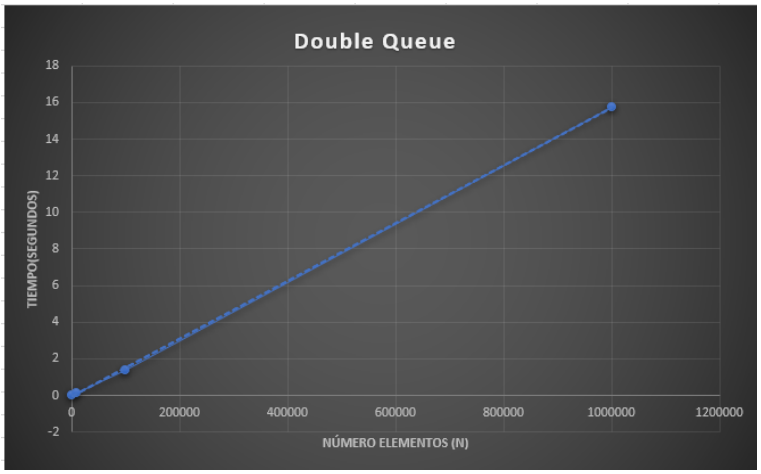

Figura 6: Tiempo ejecución Double Queue.

$$f(n) = 2 * 10^{-5} * n - 0.0709 \tag{2}$$

Ecuación para Double Queue

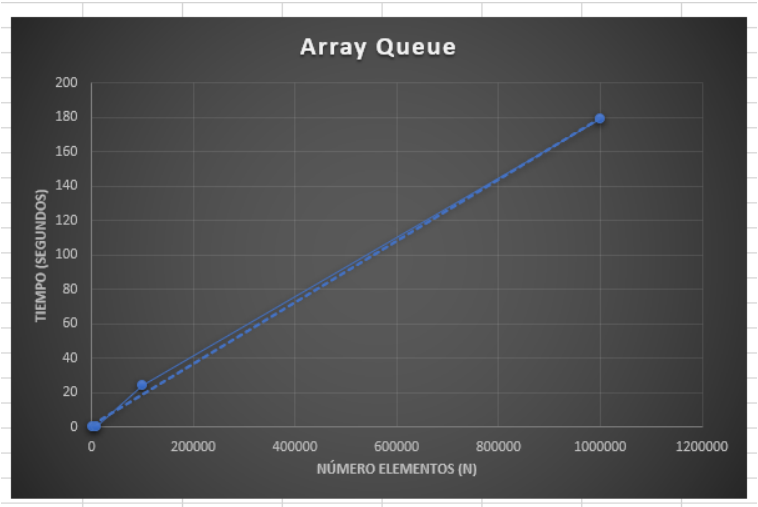

Figura 7: Tiempo ejecución Array Queue.

$$f(n) = 0.002 * n + 1.4837 \tag{3}$$

Ecuación para Array Queue.

3.4. Estimaciones de tiempo de ejecución

En base a estas ecuaciones realizaremos cálculos matemáticos **estimados** sobre el tiempo de ejecución con $10^7, 10^8 y 10^9$

Tabla 2: Tabla tiempos de ejecución estimados en segundos

Estructura	n = 10^7	n= 10^8	n = 10^9
Simple Queue	127575.47	1279575.47	12799575.47
Double Queue	199.929291	1999.9291	19999.9291
Array Queue	20001.4837	200001.4837	2000001.484

4. Conclusiones

Luego de experimentar, medir los tiempos de ejecución y analizar cada algoritmo con sus funcionalidades, podemos concluir que el más rápido y a la vez eficiente, resulta ser el usado con la estructura Double Queue, esto debido a que al usar dos punteros (uno al inicio y otro al final de la cola), facilita y acelera el proceso de agregar y evita el recorrer el queue para hacer este proceso. El algoritmo que le sigue a Double Queue, es Array Queue, con tiempos de ejecución cortos considerando la cantidad de elementos en la cola, esto porque el arreglo va siendo reemplazado (un array redimensionable) en cada iteración y no tiene que recorrer los elementos ya que la cantidad de estos está dado por cap .Por su parte el algoritmo de Simple Queue es el más lento, ya que al usar solo un puntero tiene que recorrer toda la lista para poder agregar un elemento, y esto con listas grandes realiza un gran trabajo que es directamente proporcional al tiempo de ejecución.

Finalmente este trabajo es una gran ayuda para poder aprender a utilizar de mejor manera los punteros y su vital relevancia para optimizar los algoritmos. Además de poder implementar de distinta forma estructuras trabajadas en clases.

Referencias

[1] Arreglos dinámicos y estáticos. *Definición,declaración. Ejemplos en programación* https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=162:arrays-arreglos-dinamicos-y-arrays-estaticos-definicion-declaracion-ejemplos-en-programacion-cu00211a&catid=36&Itemid=60

[2] Arreglos y Listas *Unidad de Apoyo para el Aprendizaje* http://132.248.48.64/repositorio/moodle/pluginfile.php/1469/mod_resource/content/1/contenido/index.html