TAREA EXTRACLASE I

Arreglos Paginados

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computadores
Algoritmos y Estructuras de Datos II (CE 1103)
II Semestre 2024

OBJETIVOS

GENERAL

- Desarrollar una aplicación sencilla utilizando el paradigma de Programación Orientado a Objetos en C++ aplicando el concepto de paginación

ESPECÍFICOS

- Familiarizarse con C++, Programación Orientada a Objetos y conceptos de administración de memoria

REQUERIMIENTOS

GENERATOR

001	Desarrollar un programa en C++ para generar archivos **binarios** grandes que contienen una cantidad predeterminada de números enteros **aleatorios**. Este programa se llamará *generator* y se ejecuta en la terminal de la siguiente forma: `$> generator –size <SIZE> –output <OUTPUT FILE PATH>`
002	El argumento <SIZE> puede ser uno de los siguientes valores: 1. SMALL: genera un archivo de 512 MB 2. MEDIUM: genera un archivo de 1 GB 3. LARGE: genera un archivo 2 GB <OUTPUT FILE PATH> corresponde a la ruta del sistema de archivos donde se guarda el archivo. El contenido del archivo generado por la herramienta **no es legible, es totalmente binario** (ni siquiera se puede ver los caracteres 0 o 1, no tiene codificación). Cada aleatorio número generado, se guarda en el archivo en formato binario, es decir, cada 4

	bytes escritos corresponden a un entero. No se debe separar por comas u otro separador. Es una simple serialización de enteros depositados en un archivo.

SORTER

001	Desarrollar un programa en C++ para ordenar archivos **binarios** grandes que contienen una cantidad predeterminada de números enteros **aleatorios**. Este programa se llamará *sorter* y se ejecuta en la terminal de la siguiente forma: `$> sorter –input <INPUT FILE PATH> -output <OUTPUT FILE PATH> -alg <ALGORITMO>`
002	El argumento <INPUT FILE PATH> es la ruta del sistema de archivos donde está el archivo por ordenar que contiene los números enteros aleatorios en binario. El argumento <OUTPUT FILE PATH> es la ruta del sistema de archivos donde se guarda el archivo **ordenado** resultante. El argumento <ALGORITMO> corresponde a uno de los siguientes valores: 1. QS: Quick sort 2. IS: Insertion sort 3. BS: bubble sort

003	Internamente, *sorter* utiliza una clase llamada *PagedArray* que deberá ser implementada por el/la estudiante. La clase *PagedArray* funciona de la siguiente manera:
	1. Utiliza máximo 4 arreglos de enteros (los usuales provistos por C++) de tamaño fijo. Estos corresponden a las únicas 4 páginas (o espacios para cargar páginas) de enteros que se pueden tener en memoria
	2. Sobrecarga el operador [] para proveer una abstracción familiar a un arreglo normal. Por ejemplo:
	
	`PagedArray* arr = new PagedArray();`
	`arr[1000] = 1;`
	`cout << arr[5000];`
	
	3. Cuando se procesa la operación arr[*indice*] se sigue la siguiente lógica:
	a. Calcular la página correspondiente donde el índice deberia estar.
	b. Si la página está cargada en alguno de los cuatro espacios de páginas, se retorna el valor de arr[indice] directamente desde la memoria.
	c. Si la página no está cargada, se carga desde el disco, desde el archivo que se está ordenando. Para esto, se puede usar la operación seek, junto con algunos cálculos relevantes. Si todos los espacios para páginas

	están ocupados, se escoge alguna al azar y se descarga al disco en su posición correspondiente. d. *sorter* no modifica el archivo especificado en el argumento <INPUT FILE PATH>, sino que lo copia a <OUTPUT FILE PATH> y trabaja sobre <OUTPUT FILE PATH>
004	El código de los algoritmos de ordenamiento no debe conocer de la existencia de la lógica de paginación, es decir, recibirá un PagedArray como argumento, y funcionará de la forma usual como si usara un array normal. PagedArray debe asegurar encapsula correctamente toda la lógica de paginación
005	Al finalizar la ejecución de sorter, se imprime un resumen de: 1. Tiempo durado 2. Algoritmo utilizado 3. Cantidad de page faults vs page hits Un *page fault* corresponde a cada vez que se buscó un elemento y no estaba en las páginas cargadas y **se tuvo que cargar dicha página desde el disco**. Un *page hit*, a cada vez que se buscó un elemento y estaba en las páginas cargadas y **no se tuvo que cargar dicha página desde el disco** El archivo <OUTPUT FILE PATH> deberá estar correctamente ordenado ascendentemente y **debe ser legible para poder verificar la correctitud. Deben usar comas para separar los enteros**

ASPECTOS OPERATIVOS

- El trabajo se realizará de forma **individual**
- El uso de Git y Github es obligatorio
- La fecha de entrega será según lo especificado en el TEC Digital. Se entrega en el TEC digital, un archivo .txt dentro del cual hay un URL del repositorio donde se encuentra el código.
- El repositorio en Github debe tener un README que indique cómo se ejecuta la aplicación y cómo se usa.