ESTRUCTURA DE DATOS Y ALGORITMOS 2
PRIMERA ENTREGA DE EJERCICIOS

Se aceptan entregas por Aulas hasta el 11/09/2016 a las 23:55

Para la realización de este trabajo la cátedra provee un proyecto de Visual Studio con las funciones a implementar, algunos datos de prueba y el framework visto en clase.

IMPORTANTE:

1. La definición de cada función incluye siempre indicar sus **pre** y **postcondiciones**.
2. Las pruebas provistas por la cátedra constituyen meros ejemplos y no son para nada exhaustivas. Se espera que los alumnos desarrollen pruebas propias para su código.
3. Debe entregarse solamente una carpeta comprimida conteniendo el proyecto "Ejercicios", sin el Framework.

?1. ORDENAMIENTO

1. Implemente la función `void Ordenar(Array<T> &a, const Comparador<T> &comp)` que recibe un array 'a' y un comparador 'comp' y ordena a utilizando comp. La función debe tener un orden de ejecución de $n * \log_2(n)$ caso promedio, donde 'n' es la cantidad de elementos del array. Comente brevemente qué algoritmo utilizó para resolver el problema.
2. ¿Sería posible optimizar el tiempo de ejecución de Ordenar si asumimos que el array está compuesto exclusivamente por naturales pares menores a 6000? En caso afirmativo implemente la función `void Ordenar(Array<T> &a, const Comparador<T> &comp)` que resuelva este problema.

   Nota: *el array puede contener repetidos.*

?2. BÚSQUEDA

1. Implemente la función `int Busqueda(const Array<T> a, const T elem, const Comparador<T> comp)` que dado un array ordenado y un elemento devuelve la posición del array donde se encuentra el elemento. Si el elemento no se encuentra en el array entonces devuelve -1. El orden de ejecución de la función deberá ser mejor que si se realizara una búsqueda secuencial. Comente brevemente el algoritmo utilizado y su tiempo de ejecución.

   Nota: *Recuerde que una búsqueda secuencial significa recorrer el array de principio a final comparando secuencialmente cada elemento con el buscado.*

2. Diseñe e implemente un algoritmo que dado un array de enteros y un entero ‘n’, devuelva `true` si y solo si existen dos enteros en dicho array que sumen ‘n’. El orden de ejecución de la función deberá ser $n * \log_2(n)$ en el peor caso.
   `bool ExisteSuma(const Array<int> a, int suma).`

?3. PILAS

1. Implemente la especificación de una pila dada por el archivo `Pila.h` de dos maneras diferentes comentando para cada operación su tiempo de ejecución:
   a. Utilizando una lista simplemente encadenada.
   b. Utilizando un array.

   Nota: *No olvidarse de implementar las funciones* `Puntero<Pila<T>> CrearPilaSimplementeEncadenada()` *y* `Puntero<Pila<T>> CrearPilaArray()`.
2. Implemente una función `bool Iguales(const Pila<T> &p1, const Pila<T> &p2, const Comparador<T> &comp)` que recibe dos pilas y verifica si son iguales. Dos pilas son iguales si y sólo si contienen los mismos elementos en el mismo orden.

?4. CADENAS

1. Implemente una función `Array<Cadena> Split(Cadena c, char s)` que recibe una cadena ‘c’ y un caracter ‘s’ y devuelve el array de cadenas resultante de separar a la cadena original por las ocurrencias del caracter ‘s’.
   Por ejemplo: `Split("Todos los hombres son mortales", "o") = ["T", "d", "s l", "s h", "mbres s", "n m", "rtales"]`. Comente el tiempo de ejecución de su algoritmo.
2. Utilice la pila definida en la parte 3 para implementar una función que revierta el orden de las palabras en una oración. Por ejemplo, la función debería reformar la cadena “Quien ambiciona una corona, ignora su peso.” en “peso. su ignora corona, una ambiciona Quien”. Asuma que las palabras están separadas por espacios y trate a la puntuación igual que a las letras.

?5. LISTAS ORDENADAS

1. Implemente la especificación `ListaOrd.cpp` de una lista dada por el archivo `Lista.h` de dos maneras diferentes comentando para cada operación su tiempo de ejecución:
   a. Utilizando una lista encadenada.
   b. Utilizando un array.

   Notas: *No olvidarse de implementar las funciones* `Puntero<Lista<T>> CrearListaOrdenadaEncadenada()` y `Puntero<Lista<T>> CrearListaOrdenadaConArray()`.

   La lista no tiene tamaño máximo, se deberá tener esto en cuenta a la hora de implementar la lista usando array para que la lista no esté limitada por el tamaño del array.

2. Implemente una función `bool ListasSonIguales(const Puntero<Lista<T>>& lista1, const Puntero<Lista<T>>& lista2, const Comparador<T>& comp);` que recibe dos listas y verifica si son iguales. Dos listas son iguales si y sólo si contienen los mismos elementos en el mismo orden.

?6. COLA DE PRIORIDAD - IMPLEMENTACIÓN

1. Implemente la especificación de una cola de prioridad dada por el archivo `ColaPrioridad.h`

?6. COLA DE PRIORIDAD - EJERCICIO

1. Implemente una función `Array<T> MayoresN(Array<Array<T>> datos, const Comparador<T>& comp, Natural n)` que recibe un grupo de Array con elementos comparables T y debe devolver los n elementos mayores de todos los array en forma ordenada de mayor a menor usando la cola de prioridad implementada en la parte anterior.

   Nota: *Tenga en cuenta que todos los arrays están ordenados de menor a mayor.*