TP Ordenamiento Lista Circular Doblemente Enlazada

Cassinerio Marcos - Cerruti Lautaro

Abril 29, 2020*

*Updated May 2, 2020

1 Introducción

El objetivo de este trabajo fue implementar los algoritmos de ordenamiento **Selection Sort**, **Insertion Sort** y **Merge Sort** sobre un tipo de lista a elección, en este caso utilizamos una **Lista Circular Doblemente Enlazada**.

Para la prueba de la implementación utilizamos una estructura de personas, lo que conllevó a realizar un generador de las mismas.

2 Compilado y ejecucion

Para compilar el proyecto abrimos una terminal, y una vez ubicados en el directorio del proyecto, ejecutamos el comando `make`. Esto nos generará los ejecutables del generador de datos de prueba y del programa que ejecuta los algoritmos de ordenamiento.

Para generar los datos de prueba ejecutamos el comando:

```
./generador cantidadPersonas archivoNombres archivoLugaresDeNacimiento archivoPruebas
```

Para ejecutar el segundo programa corremos el comando:

```
./main archivoPruebas
```

Este programa generará 6 archivos que tendrán de nombre el algoritmo utilizado y el dato por el que se lo ordenó de la forma `algoritmo_dato`. Estos archivos contienen una linea con el tiempo que se tardó en hacer el ordenado, y luego la anterior lista de personas generada, la cual fue ordenada por el campo que se le pasó, con el algoritmo utilizado.

3 Organizacion de los archivos

Para la primera parte del trabajo, el generador de casos de prueba, no hicimos ninguna división. Para la segunda parte, implementamos por un lado toda la lista (llamada GList) en los archivos glist.c y glist.h. Luego todo lo relacionado a persona, en los archivos persona.c y persona.h. Y por último el main del segundo programa es el archivo main.c el cual utiliza las dependecias glist y persona.

4 Implementaciones y estructuras

4.1 Generador Datos de Prueba

En el generador de personas, lo que se realiza es leer de los archivos que se les pasa como parámetros, los nombres de las personas y las localidades. Luego se imprimen la cantidad de personas pasada como parametro, generadas aleatoriamente, con el formato:
`nombre, edad, lugarDeNacimiento`.

4.2 Programa Ordenamiento

4.2.1 GList

En este programa utilizamos una lista circular doblemente enlazada, con la siguiente definición:

```
typedef struct _GNode{
  void* data;
  struct _GNode *prev;
  struct _GNode *next;
} GNode;

typedef GNode *GList;
```

Luego de generar esta estructura en la cabecera, declaramos las funciones de la GList y los 3 métodos de ordenamiento sobre esta estructura. Y una función que se encarga de aplicarlos y guarda los resultados. En definitiva, esta función prueba los distintos ordenamientos, y es la que se utiliza en el main. Las funciones mencionadas son:

`glist_create`
`glist_destroy`
`glist_pop`
`glist_concat`
`glist_insert_last_position`
`glist_copy`
`glist_destroy_copy`
`glist_print_file`
`glist_swap`
`glist_selection_sort`
`glist_insertion_sort`
`glist_merge`
`glist_split`
`glist_merge_sort`
`glist_test_sort_algorithm`

Todas estas funciones declaradas se implementan en glist.c.

4.2.2 Persona

Este programa trabaja con personas, dadas por la siguiente estructura:

```
typedef struct {
  char *nombre;
  int edad;
  char *lugarDeNacimiento; // pais o capital
} Persona;
```

Esta estructura la creamos en la cabecera junto a sus funciones básicas. Las funciones mencionadas son:

```
persona_crear
persona_destruir
persona_compara_edad
persona_compara_largo_nombre
persona_imprimir_archivo
```

Luego en persona.c implementamos todas sus funciones.

4.2.3 Main

El main se encarga de leer las lineas del archivo obtenido despues de correr el generador de los datos de prueba, crea personas a partir de las mismas y las inserta en una GList. Luego ejecuta las pruebas de los ordenamientos en esta lista. Las pruebas realizadas son los tres algoritmos de ordenamiento por edad y los mismos tres por lugar de nacimiento. Finalmente destruye la lista generada.

5 Desarrollo

Para el desarrollo, en una primera instancia hicimos una división del trabajo. Mientras que uno hacia el generador de personas, el otro implementaba las funciones básicas de GList.

Luego de que cada uno terinara su parte, juntos procedimos haciendo los algoritmos de ordenamiento. Para el selection sort y el insertion sort nos basamos principalmente en un gráfico que mostraba como funcionaban estos algoritmos.

Para el merge sort también nos basamos en un gráfico, pero adicionalmente utilizamos una implementacion hecha sobre una lista doblemente enlazada no circular. Habiendo terminando toda la parte de GList, continuamos con la parte de personas, la cual no trajo complicaciones.

Luego realizamos la función que probaba los algoritmos, y nos dimos cuenta que no podiamos ordenar sobre la lista ya existente ya que seria distinta al probar los otros algoritmos. Esto hizo que tuvieramos que desarrollar una función que realizara una copia de la lista y otra para destruir esta copia.

Reviendo los algoritmos nos dimos cuenta que el insertion sort no aprovecha las ventajas que proporciona nuestro tipo de lista. Por lo que decidimos refactorizar la implentación del algoritmo.

Una vez dado por finalizado esto hicimos el `Makefile`.

6 Resultados Pruebas

6.1 Primer Caso de Prueba

Esta primer prueba la realizamos con 5000 personas.
Resultados:

6.1.1 Edad

```
Selection Sort: 0.147 segundos
Insertion Sort: 0.096 segundos
Merge Sort: 0.005 segundos
```

6.1.2 Largo Nombre

```
Selection Sort: 0.859 segundos
Insertion Sort: 0.329 segundos
Merge Sort: 0.007 segundos
```

6.2 Segundo Caso de Prueba

Esta segunda prueba la realizamos con 15000 personas.
Resultados:

6.2.1 Edad

```
Selection Sort: 1.522 segundos
Insertion Sort: 1.103 segundos
Merge Sort: 0.013 segundos
```

6.2.2 Largo Nombre

```
Selection Sort: 9.611 segundos
Insertion Sort: 3.647 segundos
Merge Sort: 0.021 segundos
```

6.3 Tercer Caso de Prueba

Esta última prueba la realizamos con 30000 personas.
Resultados:

6.3.1 Edad

```
Selection Sort: 10.479 segundos
Insertion Sort: 9.065 segundos
Merge Sort: 0.042 segundos
```

6.3.2 Largo Nombre

```
Selection Sort: 68.993 segundos
Insertion Sort: 33.761 segundos
Merge Sort: 0.051 segundos
```

7 Bibliografia

https://en.wikipedia.org/wiki/Selection_sort
https://en.wikipedia.org/wiki/Insertion_sort
https://en.wikipedia.org/wiki/Merge_sort
https://www.geeksforgeeks.org/merge-sort-for-doubly-linked-list/
https://www.geeksforgeeks.org/how-to-measure-time-taken-by-a-program-in-c/