WILDCATS

95.15
Algoritmos y programación 2
Cátedra: Andrés Juárez - Curso 03

La batalla contra el reloj

Trabajo práctico N° 3

Integrantes:

Tiago Sandoval
Liu Gutiérrez
Abril Giordano Hoyo

Manual de usuario:

- Compile el programa con: **g++ *.cpp -o a**
- Para Valgrind: **valgrind --track-origins=yes --leak-check=full --show-leak-kinds=all ./a**
- Dentro de la misma carpeta debe estar Autores.txt y Lecturas.txt
- Corra el programa
    - Linux/macOS: **./a**

Este trabajo práctico se encarga de organizar las lecturas de la Tertulia de los mil y un martes jueves, brinda información de las lecturas y sus respectivos escritores.

Las clases principales son Escritor, Lectura con sus hijos Poema, Cuento, y Novela que a su vez tiene como hijo Novela histórica.

Como estructura de datos utilizamos los templates Lista, Nodo, Cola y Hash. Además, se hace uso de una clase Grafo para obtener el tiempo mínimo que se tarda en leer todas las lecturas, teniendo en cuenta el tiempo que Syd debe dormir entre cada una de ellas (el árbol de expansión mínimo del grafo).

Por cuestiones prácticas y de diseño, para el manejo del proyecto, implementamos las clases Archivo, Parser, Menú, Impresor y Opciones.

A su vez, se crea una Lista_lecturas y un Grafo_lecturas, las cuales heredan de el template Lista y la clase Grafo respectivamente, con el objetivo de heredar los métodos de dichas clases madres.

Para el tema de la novela histórica se utiliza un vector dinámico de tipo char, para así poder almacenar la memoria justa y necesaria dependiendo del tema dado.

Para los géneros de las novelas se utiliza enumerado, y así mediante enteros poder manejar con menos espacio (en comparación de un string) los datos tipo género.

En el menú tenemos un método selector, que se encarga de manejar los métodos de la clase Opciones, la cual se ocupa de conectar las clases y llevar a cabo la ejecución correcta del programa.

Diseño:

- Está desarrollado usando el paradigma de Programación Orientada a Objetos (POO).
- Herencia y polimorfismo.
- Uso de la memoria dinámica.
- Manejo de archivos .TXT (Autores.txt, Lecturas.txt)
- El manejo y almacenamiento temporal y dinámico de datos se hace mediante Listas, Grafos y Colas, esta última para la creación del método próximas lecturas.
- Crea una lista de lecturas, en la cual se basan tanto la cola de lecturas (para la opción de próximas lecturas) como el grafo de lecturas (para la opción del árbol de expansión mínimo), junto con la clase Prim que es la que se encarga principalmente de ejecutar el algoritmo de Prim.
- Crea una tabla de Hash para almacenar escritores.
- El trabajo práctico fue diseñado e implementado en Linux.
- Uso del repositorio: github.com/liuvaneshka/tp3

Archivo:

Se encarga de abrir, cerrar y obtener las líneas del archivo.

Atributos de archivo:

- Un archivo de tipo ifstream.
- Abierto de tipo booleana.
- Línea de tipo cadena

Parser:

Se encarga de traducir los archivos (Autores.txt y Lecturas.txt) a su forma objeto para luego insertar las direcciones de los objetos dentro de los nodos de las lista o tabla correspondiente (para las lecturas y los escritores, respectivamente).

Atributo del parser:

- Una entrada de tipo Archivo, conteniendo una ruta de tipo cadena.

Hash:

Para parte de su implementación se hizo uso de la librería STL. Se encarga de almacenar los objetos de tipo Escritor, su clave es el resultado del cálculo de la suma de cada carácter (representación de cada carácter en ASCII) dentro de la cadena sin incluir los paréntesis, luego a este número se le aplica el módulo t, en este caso 29.

- ¿Cómo sabemos que t es 29?

  En teoría el tamaño de la tabla debe ser un número primo, y al menos 1.3 veces el número máximo de claves a ingresar, para el TP3 asumimos serían 20 (cantidad de escritores del archivo).

  La fórmula dada en clases para determinar el tamaño adecuado de la tabla sería:

  t = claves/lambda, sabiendo que lambda es 0.8.

  Esta operación da por resultado 25, pero utilizamos el t = 29, dado que es el número mayor y más cercano que cumple la condición de ser primo.

- Manejo de colisiones (Direccionamiento cerrado):

  Se creó una Lista con doble parámetros, una para la llave (isni), y otro para el valor (puntero al objeto escritor), en el caso de que ya el bucket esté ocupado, se crea una nueva Lista que se enlaza a la primera.

- Complejidad (Big O):

  - ★ Mejor de los casos

    O(1) para el alta, baja y modificación.

  - ★ Peor de los casos:

    Cuando todas las claves nos conducen a un mismo índice o bucket sería de O(n) para el alta, baja y modificación, siendo n la cantidad de claves.

  - ★ Caso promedio:

    O(1 + n/t), el costo de calcular la función hash es O(1), y luego n/t que representa n = número de claves / t = número índices o Buckets de la tabla.

Complejidad de la baja

```
template <class T1, class T2>
void Hash<T1, T2> :: eliminar(T1 llave){
    if(indices.empty())
        cout << "Tabla vacia" << endl;

    int indice = funcion_hash(llave);
    Lista_doble_parametro<T1, T2> *puntero = indices[indice];

    if(puntero->obtener_llave() == llave){
        indices[indice] = puntero->obtener_siguiente();
        Escritor* escritor_actual = puntero->obtener_valor();
        delete escritor_actual;
        delete puntero;
    }

    else{
        while((puntero != nullptr) && (puntero->obtener_siguiente()->obtener_llave() != llave))
            puntero = puntero->siguiente;

        if(puntero != nullptr){
            Lista_doble_parametro<T1, T2> *aux = puntero->obtener_siguiente();
            puntero->siguiente = puntero->siguiente->obtener_siguiente();
            Escritor* escritor_actual = aux->valor;
            delete escritor_actual;
            delete(aux);
        }
    }
}
```

Mejor caso

Caso promedio y peor caso

Mejor Caso: Cuando el escritor que se desea eliminar se encuentra en la primera posición de la lista, dentro del índice correspondiente de la tabla, ya que solo se accede al índice y se hace una única comparación.

Peor Caso: Cuando el escritor que se desea eliminar se encuentra en la última posición de la lista, dentro del índice correspondiente de la tabla, dado que se la debe recorrer hasta el final y hacer "n" comparaciones (n = tamaño de la lista que se encuentra en dicho índice).

Caso Promedio: Depende principalmente de las probabilidades de que se generen colisiones, es decir, que para distintos datos se obtenga el mismo índice.
Sin embargo, un posible caso promedio podría ser cuando el escritor que se desea eliminar se encuentra en alguna posición intermedia de la lista. Luego, no se recorre hasta el final pero se itera m veces, siendo $m > 1$ y $m < n$, con n el tamaño de la lista de dicho índice.

Complejidad del alta

```
template <class T1, class T2>
void Hash<T1, T2> :: insertar_en_indice(int indice, T1 llave, T2 valor){
    if(indices.at(indice) == nullptr){
        Lista_doble_parametro<T1, T2> *nodo =  new Lista_doble_parametro<T1, T2>(llave, valor);
        indices[indice] = nodo;

    }                                                                    Mejor caso
    else{
        Lista_doble_parametro<T1, T2> *puntero = indices[indice];// Apunta al primero

        while(puntero->siguiente != nullptr){
            puntero = puntero->siguiente;
        }
        Lista_doble_parametro<T1, T2> *nodo = new Lista_doble_parametro<T1, T2>(llave, valor);
        puntero->siguiente = nodo;                                       Caso promedio
    }                                                                    y peor caso
}
```

Mejor Caso: Cuando el escritor que se quiere insertar se almacena en un índice donde todavía no hay ningún escritor, ya que se calcula cuál es el índice que le corresponde y se guarda el dato en esa posición.

Peor Caso: Cuando el escritor que se desea insertar se almacena en un índice donde ya existen escritores, dado que se debe almacenar al final de la lista en dicha posición y por ende se debe recorrer las n posiciones de la lista.

Caso Promedio: Depende en este caso de las posibilidades de que se generen colisiones, teniendo en cuenta los datos de entrada de Escritores.txt como también la función de hashing utilizada, la cual podría ser “mejor” o “peor” dependiendo justamente de los isni de los escritores.

Complejidad de la búsqueda

```
template <class T1, class T2>
T2 Hash<T1, T2> :: encontrar_dato(T1 llave) {
    T2 valor = nullptr;
    int indice = funcion_hash(llave);
    if (indices.at(indice)) {
        Lista_doble_parametro<T1, T2> *puntero = indices[indice];

        while (puntero != nullptr && puntero->obtener_llave() != llave)
            puntero = puntero->obtener_siguiente();

        if(puntero != nullptr)
            valor = puntero->obtener_valor();
    }
    return valor;
}
```

Mejor Caso: Cuando el escritor que se desea buscar se encuentra en la primera posición de la lista, dentro del índice correspondiente de la tabla, ya que solo se accede al índice y se hace una única comparación. En estas condiciones, nunca se ingresa al ciclo y por lo tanto solo se devuelve el valor correspondiente.

Peor Caso: Cuando el escritor que se desea buscar se encuentra en la última posición de la lista, dentro del índice correspondiente de la tabla, dado que se la debe recorrer hasta el final y hacer "n" comparaciones (n = tamaño de la lista que se encuentra en dicho índice).

Caso Promedio: Depende de las posibilidades de que se generen colisiones, teniendo en cuenta los datos de entrada de Escritores.txt como también la función de hashing utilizada, la cual podría ser "mejor" o "peor" dependiendo justamente de los isni de los escritores.
Sin embargo, un posible caso promedio podría ser cuando el escritor que se está buscando se encuentra en alguna posición intermedia de la lista. Luego, no se recorre hasta el final pero se itera m veces, siendo $m > 1$ y $m < n$, con n el tamaño de la lista de dicho índice.

Menú:

Se encarga de conectar al usuario con el programa, muestra las opciones que acceden a los métodos que unen estas clases.

Atributos del menú:

- Opción de tipo entero.
- Opciones de tipo Opción
- Lecturas de tipo Lista lecturas.
- Lecturas de tipo Cola
- Lecturas de tipo Grafo lecturas
- Escritores de tipo Hash

Estas se inicializan en un método del menú que se encarga de cargar dichos atributos. Luego, las listas de lecturas y escritores se inicializan invocando a los métodos de procesar lecturas y escritores dentro del parser.

Impresor:

Se encarga únicamente de pedirle al usuario que ingrese valores determinados para ser procesados en su respectivo entorno. Es una clase diseñada para modularizar correctamente el menú.

Opciones:

Nueva clase creada con el objetivo de que el menú únicamente se centre en lo relacionado a inicializar el programa y en la elección de las opciones. La clase Opciones posee todos los métodos que corresponden con las opciones disponibles del programa.

Las opciones a elegir por el usuario son las siguientes:

- 1. Agregar una nueva lectura a la lista.
- 2. Quitar una lectura de la lista.
- 3. Agregar un escritor.
- 4. Modificar año de fallecimiento.
- 5. Listar los escritores.
- 6. Sortear una lectura random para leer en una de las tertulias
- 7. Listar todas las lecturas.
- 8. Listar las lecturas entre determinados años.
- 9. Listar las lecturas de un determinado escritor.
- 10. Listar las novelas de determinado género.

- 11. Armar una cola ordenada por tiempo de lectura
- 12. Obtener el tiempo mínimo y el orden de lectura
- 13. Eliminar escritor
- 14. Salir

División de Tareas

Tiago Sandoval:
Clases:

- Grafo

Implementaciones :

- Mejoras en la interfaz, incluyendo mensajes de error para el usuario cuando ingresa datos erróneos
- Actualización en métodos de listar por escritor y modificar fallecimiento
- Solución de pérdidas de memoria con Leaks (macOS)

Liu Gutiérrez:
Clases:

- Tabla Hash
- Lista con doble parámetros.

Implementaciones:

- Nuevas implementaciones: alta, baja y modificación de la tabla de escritores.
- Actualización del escritor dentro de las Lecturas, a la hora de dar una baja a un escritor se deben actualizar las lecturas del escritor a eliminar.
- Perfeccionamiento de la clase menu, al crear la clase opciones.

Abril Giordano Hoyo:
Clases:

- Grafo de lecturas
- Prim

Implementaciones:

- Nueva implementación de Lista_lecturas, utilizando herencia para no repetir métodos.
- Nueva implementación de cola lecturas, para que se incluyan las lecturas ingresadas por el usuario.
- Actualización de los atributos de Lectura, incluyendo el atributo de "leída" para la opción de marcar como leída.