Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE 0217 – Estructuras de Datos Abstractas y Algoritmos
III ciclo 2024

Laboratorio II

Danna Guevara C23562
Grupo 01

Profesor: Karen Dayana Tovar Parra

2 febrero

Índice

1. Resumen

Inicialmente, me gustaría recalcar que este es el primer curso en el cual manejo el programa C++, por lo que busque en gran cantidad de sitios para poder tener un buen entendimiento sobre que se debía de realizar y como se aplicaban los diferentes comandos.

2. Ejercicio 2

Al volver a probar el ejercicio, me di cuenta de que estaba eliminando los números duplicados, pero no todos, solamente estaba comparando entre el actual y el que seguía, por lo que al preguntar a chat gpt se me dieron varias opciones; primero opté por realizar una lógica parecida al primer laboratorio donde se acomodan de menor a mayor y se eliminan todos los siguientes, sin embargo implicaba realizar toda otra función, por lo que terminé optando por utilizar el conjunto unordered_set, donde en la página geekforgeeks, muestra cómo se crea y se maneja este conjunto; sin embargo, lo hace con un array, por lo que tuve que cambiar la lógica un poco para que así me pudiese guiar de las mismas ideas que me proporcionó la página tal cual como insert; que agrega un nuevo elemento al unordered_set o count que funciona como un true o false.

`https://chatgpt.com/share/679fa420-505c-8001-bb47-7594fb29a9d9`
`https://www.geeksforgeeks.org/check-given-array-contains-duplicate-elements-within-k-di`

3. Ejercicio 5

Este ejercicio tenía bastantes conceptos por lo que preferí ayudarme buscando en internet para las funciones más pesadas, así mismo encontré una gran ayuda de guía para realizar los métodos más grandes, así fue como creé un nuevo nodo con el valor y la prioridad, de esta manera en el método enqueue, se pudo crear el nuevo nodo, aunque el código encontrado estuviese en lenguaje Java, traducirlo fue muy sencillo para guiarme al traducirlo a C++, unos detalles como el orden de prioridad, este lo imprimía al revés por lo que tan solo cambiando el `>=` al revés se colocó en orden ascendente.
Como tal, lo que se hace en cada una de las funciones especificamente; empezando con enqueue es crear un nuevo nodo con el valor y la prioridad proporcionados, luego si la cola está vacía el nuevo nodo es el primero y último nodo de la cola. Si el nuevo nodo tiene menor prioridad que el primero, se inserta al inicio de la cola. Si el nuevo nodo tiene una prioridad intermedia, se recorre la cola hasta encontrar la posición correcta, es decir, el primer nodo con prioridad menor o igual al nodo actual se inserta ahí y, si es necesario, se actualiza el puntero del último nodo. Finalmente, se incrementa el tamaño de la cola. Con el siguiente método dequeue; si la cola está vacía no hay elementos para eliminar y si la cola no está vacía, extrae el valor del primer nodo. Elimina el primer nodo de la cola, actualizando el puntero first al siguiente nodo de la lista. Disminuye el tamaño de la cola y devuelve el valor extraído del primer nodo. Finalmente, en el updatePriority; si la cola está vacía, no hay nada que actualizar. Se busca el nodo con el valor especificado, si no se encuentra, la función termina. Cuando se encuentra el nodo: si es el primer nodo, se actualiza el puntero first para que apunte al siguiente nodo, si no es el primer nodo, se ajusta el puntero del nodo anterior para que apunte al siguiente nodo del nodo que se va a eliminar y si el nodo encontrado era el último en la cola, se actualiza el puntero last para que apunte al nodo anterior. Disminuye el tamaño de la cola. Finalmente, se vuelve a insertar el nodo con el mismo valor pero con la nueva prioridad utilizando el método enqueue.

`https://www.quora.com/How-would-you-make-a-priority-queue-using-arrays-or-linked-lis`

4. Ejercicio 6

Mi idea principal para realizar este ejercicio era recorrer el datamap con un doble for, sin embargo, no tomé en cuenta que el datamap[i] era un puntero a Node, no una estructura iterable por lo que se necesitaba de un while para poder recorrer las listas enlazadas en el caso de haber colisiones, además de que aunque se estuviera almacenando las claves aún así se tenía que seguir analizando los siguientes nodos con temp $\rightarrow$ *next*.

`https://chatgpt.com/share/679f0a4b-3cec-8001-b01b-aa17843d22af`

5. Instrucciones Makefile

El laboratorio cuenta con un archivo tipo `Makefile`, el cual está diseñado para gestionar los diferentes ejercicios del laboratorio, estableciendo variables de entorno y rutas a directorios específicos.

Para ejecutar este `Makefile`, es necesario abrir una terminal en el directorio raíz del proyecto (donde está ubicado el archivo `Makefile`) y utilizar el comando:

```
make <objetivo>
```

Reemplazando `<objetivo>` con el nombre de la tarea que se desea ejecutar, los objetivos definidos en el `Makefile` se presentan a continuación. Cada objetivo especifica un conjunto de comandos para compilar los ejercicios según las rutas definidas en las variables.

- `ej1`
- `ej2`
- `ej3`
- `ej4`
- `ej5`
- `ej6`

Después de ejecutar make, los ejecutables se crean en la carpeta build/; por lo que se debe de utilizar el siguiente comando seguidamente del make;

```
./build/<objetivo>
```

- `ej1`: ejecuta el ejercicio 1 que invierte una lista enlazada.
- `ej2`: ejecuta el ejercicio 2 que eliminar todos los valores duplicados de una lista enlazada simple.
- `ej3`: ejecuta el ejercicio 3 que verifica si una lista doblemente enlazada es un palíndromo.
- `ej4`: ejecuta el ejercicio 4 que utiliza el stack para invertir una cadena de entrada.
- `ej5`: ejecuta el ejercicio 5 que utiliza el sistema de cola para implementar un sistema de tickets que permita manejar diferentes niveles de prioridad dinámica
- `ej6`: ejecuta el ejercicio 6 que obtiene todas las claves utilizando hashtables.