CS2023 - Aula de Ejercicios Nº 5

Heider Sanchez
ACL: Juan Diego Castro
Semestre 2024-1

- Para este laboratorio ser formarán grupos de dos integrantes, pero se sugiere que cada estudiante intente primero resolver los ejercicios de forma **individual**.
- Queda totalmente prohibido el uso de foros de internet, ChatGTP, repositorios de GitHub, etc.
- Cualquier pequeño indicio de plagio será calificado con 0 (ya no habrá clemencia).
- Entregar los archivos **ejercicio1.cpp**, **ejercicio2.cpp** y **ejercicio3.cpp** sin comprimir. Cualquier otro archivo **no será revisado**.

Ejercicios

1. (6 pts) Dado un string **s**, encuentra el primer caracter no repetido en **s** y retorna su índice (posición). Si dicho caracter no existe, retornar -1.

   Nota: Solo puede recorrer el string 1 vez.

   - **Ejemplo 1:**
     Input: s = "leetcode"
     Output: 0
   - **Ejemplo 2:**
     Input: s = "loveleetcode"
     Output: 2
   - **Ejemplo 3:**
     Input: s = 'aabb'
     Output: -1

   Restricciones:

   - $1 \leq s.length \leq 10^5$
   - s solo contiene letras del alfabeto inglés en minúsculas

   Nota: Resolver utilizando el std::unordered_map.

2. (6 pts) Dado un arreglo no ordenado de números **nums**, retornar el entero positivo más pequeño que no está presente en **nums**. Es obligatorio que el algoritmo corra en $O(n)$. Puede utilizar una estructura auxiliar.

   Reto: (+2 pts) ¿Puedes resolverlo en $O(n)$ y a la vez usar $O(1)$ memoria adicional? (in-place)

   - **Ejemplo 1:**
     Input: nums = [1,2,0]
     Output: 3
     Explicación: Los números en el rango [1,2] están presentes en el arreglo, pero 3 no.
   - **Ejemplo 2:**
     Input: nums = [3,4,-1,1]
     Output: 2
     Explicación: 1 está en el arreglo, pero 2 no.

- **Ejemplo 3:**
  Input: nums = [7,8,9,11,12]
  Output: 1
  Explicación: El entero positivo más pequeño no está!

Restricciones:

- $1 \leq nums.length \leq 10^5$
- $-2^{31} \leq nums[i] \leq 2^{31} - 1$

Nota: Resolver utilizando el std::unordered_map.

3. (8 pts) Diseñe una estructura de datos que almacene números y responda algunas queries sobre las frecuencias de los elementos que contiene.

   Implemente la clase FrequencyTracker:

```
class FrequencyTracker {
public:
    FrequencyTracker();
    void add(int number);
    void deleteOne(int number);
    bool hasFrequency(int frequency);
};
```

   - FrequencyTracker(): Inicializa la instancia de FrequencyTracker.
   - void add(int number): Agrega un número a la estructura.
   - void deleteOne(int number): Disminuye en uno la ocurrencia del número dentro de la estructura. La estructura podría no tener dicho número, en cuyo caso no debería ocurrir ningún cambio.
   - bool hasFrequency(int frequency): Retorna verdadero si es que existe al menos un número en la estructura con frecuencia **frecuency**. Caso contrario, retorna false.

   Nota: **Todas las operaciones tienen que realizarse en $O(1)$ y puede usar hasta $O(n)$ memoria adicional.**

   Hint: Recordar que $2n \in O(n)$

   - Ejemplo 1:
     Input

```
["FrequencyTracker", "add", "add", "hasFrequency"]
[[], [3], [3], [2]]
```

     Output

```
[null, null, null, true]
```

     Contexto

```
FrequencyTracker frequencyTracker = new FrequencyTracker();
frequencyTracker.add(3); // La estructura contiene: [3]
frequencyTracker.add(3); // La estructura contiene: [3, 3]
frequencyTracker.hasFrequency(2); // Retorna true, porque 3 ocurre 2 veces
```

   - Ejemplo 2:
     Input

```
["FrequencyTracker", "add", "deleteOne", "hasFrequency"]
[[], [1], [1], [1]]
```

Output

```
[null, null, null, false]
```

Contexto

```
FrequencyTracker frequencyTracker = new FrequencyTracker();
frequencyTracker.add(1); // La estructura contiene: [1]
frequencyTracker.deleteOne(1); // La estructura se convierte en: []
frequencyTracker.hasFrequency(1); // Retorna false, porque está vacía
```

Restricciones

- $1 \leq number \leq 10^5$
- $1 \leq frequency \leq 10^5$
- Se harán como máximo $2 \cdot 10^5$ llamadas a add, deleteOne, y hasFrequency en total.