Análisis y diseño de algoritmos

Tarea 1 - Problema 2

Rodriguez Prieto Diego Guadalupe

Profesor/Dr. Eduardo Arturo Rodríguez Tello

1. Introducción

El presente reporte ofrece una solución al problema planteado, el cual nos presenta una situación en la que ahora el coordinador del CINVESTAV necesita saber entre cual es la primer edad repetida que hay dentro de su vector U de tamaño n.

2. Metodología

Para la resolución de esta problemática, inicialmente hicimos uso de dos opciones que nos proponen maneras efectiva de realizarlo, pero se hará comparacion del mismo para poder logar definir cual es mejor, estos son:

- vector
- unordered_set

2.1. Pseudocódigo

El problema puede descomponer en 4 partes importantes:

1. **Leer datos**: n, y $U[]$.
2. **Tomar la primera edad**: $U[i]$
3. **Verificar si la edad se repite**: $U[i] = U[i+1]$ (caso contrario, cambiar de posición para comparar)
4. **Resultado**: Retronar la edad repetida.

La problemática principal dentro de este cuestionamiento, radica en que el peor de los casos para nosotros seria en agregar una complejidad de n^2 cuando se busque y se almacene el primer componente del vector, para ello se deberia de hacer un for anidado (haciendo uso del componente *vector*), pero si se hace uso de *unordered_set* se lograría una complejidad de $O(1)$, debido a que este esta diseñado para la inserción o búsqueda de datos en volúmenes grandes, siempre y cuando no se priorice el orden de los componentes del vector (tal como se habla en este problema).[2]

Algorithm 1 Encontrar la primera edad repetida.

```
1:  procedure ENCONTRARPRIMERAREPETIDA(U, n)
2:      edades_vistas ← ∅
3:      for i ← 0 to n − 1 do
4:          if U[i] in edades_vistas then
5:              return U[i]
6:          end if
7:          edades_vistas.insert(U[i])
8:      end for
9:      return -1
10: end procedure
```

Este código mejora mucho la eficiencia debido a que esta estructura trabaja por contenedores, de modo que cuando recorre, lo almacenaría en dos posibles, en el bucket, siendo el espacio donde se guardarían los datos que contengan el mismo valor del hash y edades_vistas, que es un contenedor donde se van las edades ya sea que se vieron y no pasaron al *bucket* o la que se verá. [3]

3. Resultados

Los resultados haciendo uso de unordered_set, fueron muy p0co notorios, teniendo ambientes de prueba en donde solo se trabajaba con archivos txt de solo 2000 elementos para el vector, lo que si se denota, es el aumento, ya que cuando se hicieron pruebas para archivos con 200 elementos el contenedor vector tardaba 0.00015 y el otro 0.0002, al momento de que se aumentara a 2000, vector multiplico este tiempo por 5 veces y unordered_set solo por 2.

4. Discusión

Con respecto a lo dicho en el apartado de resultados, se denota como unordered_set puede mantenerse en tiempos favorables, mientras que si aumentamos la cantidad de elementos por ejemplo a 1×10^6, el tiempo de ejecución se dipararia haciéndose mayor.

5. Conclusión

Mas allá de que el código contenga herramientas importantes para la gestión de la complejidad algorítmica como lo es el uso de *vector*[1] (mismo que en trasfondo hace uso de memoria dinámica), no llega a ser lo mas eficiente, ya que debido a la necesidad de almacenar, recorrer y comparar se le agregaba una mayor complejidad que la que nos proporciona la estructura de *unordered_set* por su naturaleza de funcionamiento.

Referencias

[1] «Duda sobre la clase vector». Accessed: 2024-09-09. (mayo de 2018), dirección: https://es.stackoverflow.com/questions/163092/duda-sobre-la-clase-vector.

[2] G. Sharma. «Searching in Vector, Set, and Unordered_Set». Accessed: 2024-09-09. (), dirección: https://medium.com/@gx578007/searching-vector-set-and-unordered-set-6649d1aa7752.

[3] *unordered_set (Clase)*, https://learn.microsoft.com/es-es/cpp/standard-library/unordered-set-class?view=msvc-170, Accedido: 2024-09-09.