Universidad del Rosario
Matemáticas aplicadas y ciencias de la computación
Complejidad Conjuntos y Mapas
Tatiana Cabrera
María Valeriano
2024.1

UNIVERSIDAD DEL ROSARIO

COMPLEJIDAD CONJUNTOS Y MAPAS

1. Entiende el Problema:

El problema consiste en gestionar las vacaciones de una familia, lo que incluye:
Agregar destinos a una lista general.
Permitir que los miembros de la familia planifiquen viajes a destinos específicos.
Consultar los destinos planificados por un miembro de la familia.
Encontrar los miembros que planean visitar un destino específico.

2. Identifica las Partes Críticas:

Las partes críticas del código son las funciones que realizan las operaciones principales:
agregarDest: Agrega un nuevo destino a la lista general.
planificarViaje: Permite a un miembro de la familia planificar un viaje a un destino.
consultarDestinosPlanificados: Muestra los destinos planificados por un miembro de la familia.
encontrarMiembrosPorDestino: Encuentra los miembros que planean visitar un destino específico.

3. Conteo de Operaciones Básicas:

La cantidad de operaciones básicas que se ejecutan en cada función depende de la entrada:
agregarDest: Realiza O(1) operaciones para insertar el destino en la lista.
planificarViaje: Realiza O(n) operaciones para buscar el destino en la lista general y O(1) para agregarlo al miembro.
consultarDestinosPlanificados: Realiza O(n) operaciones para recorrer los destinos planificados del miembro.
encontrarMiembrosPorDestino: Realiza O(n * m) operaciones, donde n es el número de miembros y m es el número de destinos planificados por cada uno.

4. Notación O Grande (O):

La complejidad en el peor caso del código se puede expresar como:
agregarDest: O(1).
planificarViaje: O(n).
consultarDestinosPlanificados: O(n).
encontrarMiembrosPorDestino: O(n * m).

5. Analiza los Bucles:

El código contiene bucles en las funciones planificarViaje, consultarDestinosPlanificados y encontrarMiembrosPorDestino.
planificarViaje recorre la lista general de destinos para buscar el destino especificado (O(n)).
consultarDestinosPlanificados recorre los destinos planificados del miembro para mostrarlos (O(n)).
encontrarMiembrosPorDestino recorre los miembros de la familia y sus destinos planificados para encontrar coincidencias (O(n * m)).

6. Considera Funciones y Recursión:

El código no contiene llamadas recursivas.

7. Evalúa Algoritmos Específicos:

El código utiliza estructuras de datos eficientes como unordered_set y map para almacenar la información de destinos y miembros de la familia. La búsqueda del destino en la lista general se realiza con el algoritmo find_if, que tiene una complejidad de O(n).

8. Prueba con Diferentes Tamaños de Entrada:

El rendimiento del código dependerá del tamaño de la entrada, es decir, la cantidad de destinos y miembros de la familia. Para listas pequeñas, el código será rápido y eficiente. Para listas grandes, el rendimiento puede disminuir, especialmente en la función encontrarMiembrosPorDestino.

9. Comparación con Escalabilidad:

El código es escalable hasta cierto punto. Puede manejar listas de tamaño mediano a grande de manera eficiente. Sin embargo, para listas muy grandes, el rendimiento de la función encontrarMiembrosPorDestino puede degradarse significativamente.

10. Documenta tus Conclusiones:

La complejidad del código en el peor caso es O(n * m), donde n es el número de miembros de la familia y m es el número de destinos planificados por cada uno.
El código es escalable hasta cierto punto, pero el rendimiento puede disminuir para listas muy grandes.Se podrían considerar optimizaciones para la función encontrarMiembrosPorDestino, como utilizar un índice hash para almacenar los destinos planificados de cada miembro.