In flight entretainment system

Miguel Pérez García - A01630428

Problema

You've built an inflight entertainment system with on-demand movie streaming.
Users on longer flights like to start a second movie right when their first one ends, but they complain that the plane usually lands before they can see the ending. So you're building a feature for choosing two movies whose total runtimes will equal the exact flight length.
Write a function that takes an integer flightLength (in minutes) and an array of integers movieLengths (in minutes) and returns a boolean indicating whether there are two numbers in movieLengths whose sum equals flightLength.
When building your function:

- Assume your users will watch exactly two movies
- Don't make your users watch the same movie twice
- Optimize for runtime over memory

GOTCHAS

We can do this in O(n) time, where n is the length of movieLengths.
Remember: your users shouldn't watch the same movie twice. Are you sure your method won’t give a false positive if the array has one element that is half flightLength?

Solución

Primero que nada tenemos que pensar en el problema en sí, ya que es similar al primer problema que hicimos en el semestre (two sum), podemos decir que una solución podría ser aplicar el mismo algoritmo que usamos para two sum, sin embargo, ese algoritmo es de fuerza bruta y daría un resultado en O(n^2) y el problema nos dice que se puede solucionar en O(n).

También podemos ordenar el arreglo, en cuyo caso podría facilitar la búsqueda de dos números que sumados nos den un tercero, pero si lo ordenamos, utilizando un Quick Sort por ejemplo, tendremos O(n^2) al menos, por el ordenamiento, incluso utilizando un ordenamiento como merge sort, obtenemos una complejidad de O(n log(n)) al menos. Además, aunque el arreglo esté ordenado, no significa que los números que nos den estarán uno junto al otro, así que también queda descartado por no cumplir con el O(n).

Yo propongo la siguiente solución:

```
bool SumExists(int fligthLength, int[] moviesLength){

    HashSet<int> complements = new HashSet<int>(moviesLength.Length + 1);
    for (int i = 0; i < moviesLength.Length; i++)
    {
        if(complements.Contains(moviesLength[i]))
        {
            return true;
        }

        complements.Add(fligthLength - moviesLength[i]);
    }

    return false;
}
```

Ahora, para cumplir con la cuota de O(n) debemos hacer un solo loop, no nos podemos salvar del loop porque tenemos que revisar los elementos del arreglo, así que solo podemos hacer una pasada sobre el arreglo.

Por ejemplo, si necesitamos encontrar dos números que sumados nos den 10, y nuestro arreglo es el siguiente: [3, 4, 8, 10, 2, 5, 7], entonces el metodo toma el primer elemento, checa si ya lo tiene dentro del set de complementos, si no lo tiene, obtiene el complemento requerido para ese elemento y lo agrega al set de complementos, en el caso de nuestro arreglo ejemplo, cuando llega al tercer elemento, va a meter un dos al set de complementos, por lo que al llegar al quinto elemento, va a regresar con un verdadero, pues el tercer elemento ingresó el complemento que buscaba, el cual es un dos.

Este metodo es O(n), como nos dice la documentacion del HashSet en C#, los metodos de Add (https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.hashset-1.add?view=netframework-4.7.1#Remarks) y Contains (https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.hashset-1.contains?view=netframework-4.7.1#Remarks) son O(1), hace la aclaracion que Add es O(1) siempre y cuando no tenga que aumentar el tamaño del arreglo, por lo que desde un inicio le damos el tamaño que va a tener, sabemos que no sobrepasará jamas el tamaño del arreglo de peliculas, por lo que es seguro inicializarlo al tamaño del arreglo de peliculas mas uno, para que el numero de elementos siempre sea menor a la capacidad.

Además, podemos asegurar que no recomendaremos la misma pelicula dos veces, pues nunca volvemos a regresar en el arreglo, siempre se busca hacía adelante.