Two Pointers problems

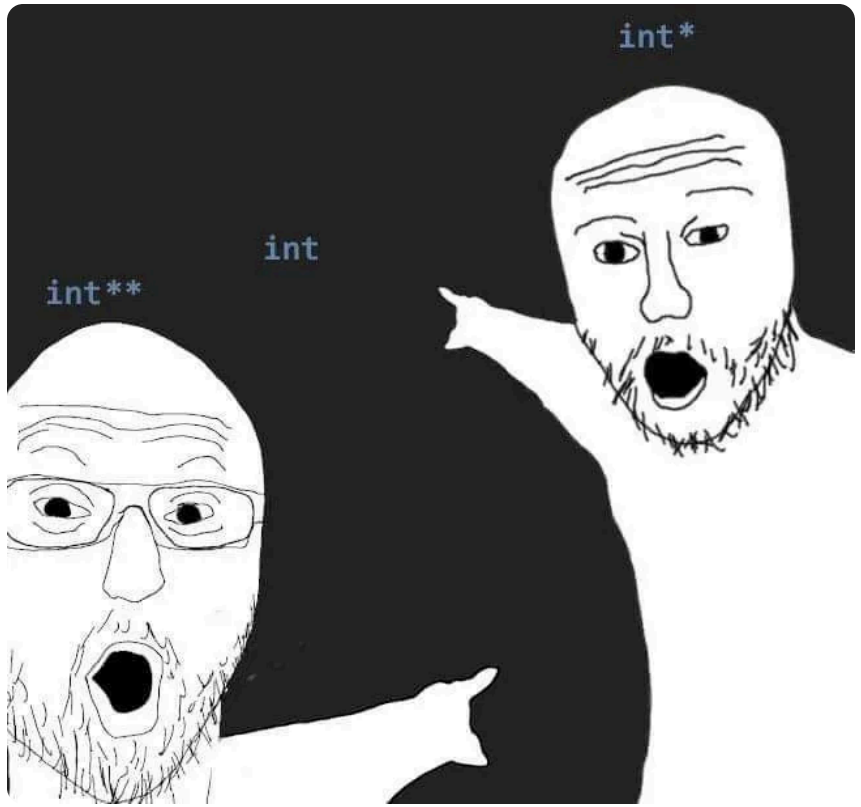
Por Alan Martinez

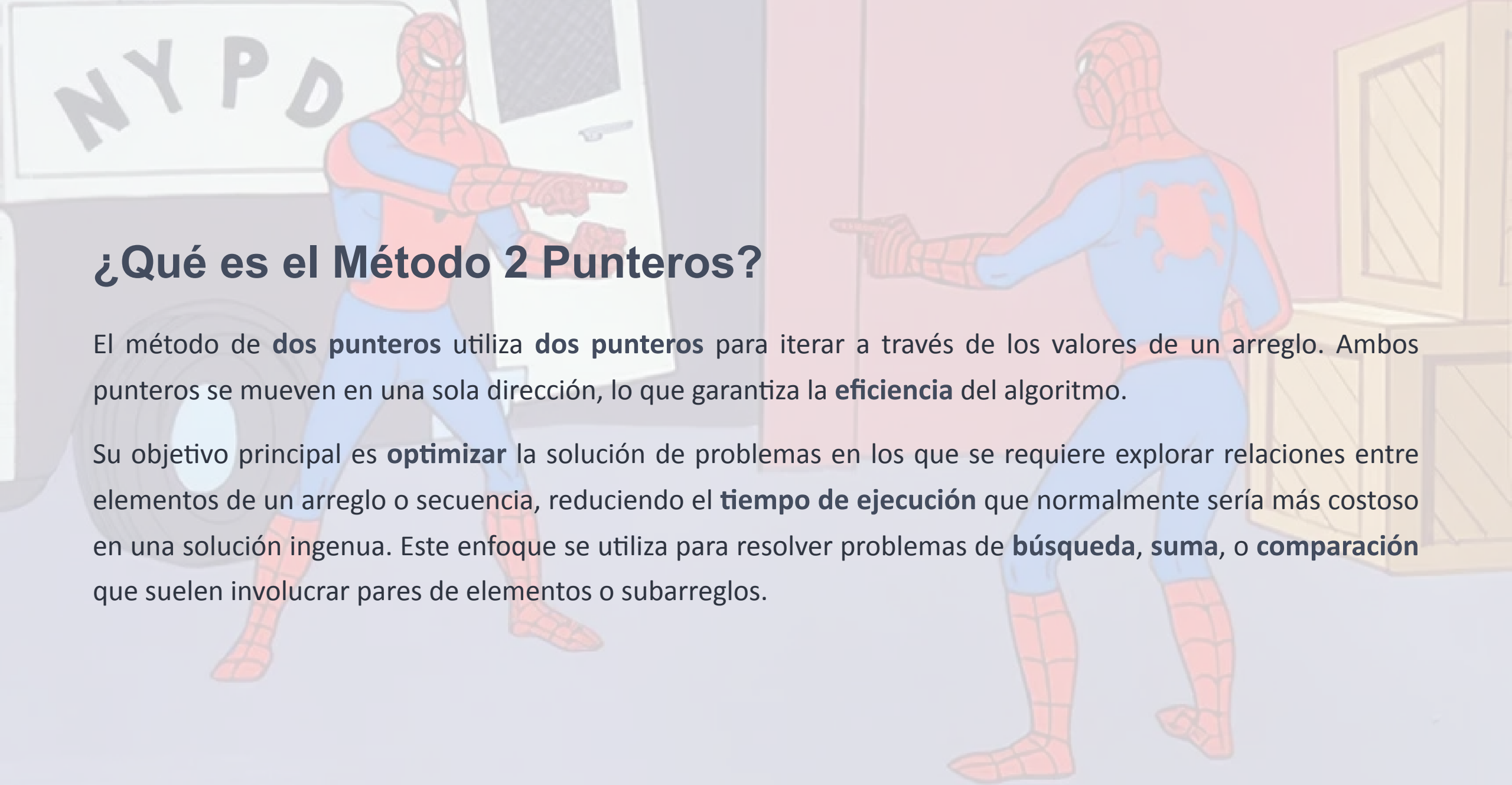
[ΓΩ5]

Repaso

Un apuntador también conocido como punteros, son la **representación de una dirección**, estos son sirven para **ahorrar memoria**, **crear vectores y matrices con memoria dinamica** y como **paso de parametros**.

Este se declara con el operador `*`, y se le asignan direcciones de memoria con el operador `&`.





¿Qué es el Método 2 Punteros?

El método de **dos punteros** utiliza **dos punteros** para iterar a través de los valores de un arreglo. Ambos punteros se mueven en una sola dirección, lo que garantiza la **eficiencia** del algoritmo.

Su objetivo principal es **optimizar** la solución de problemas en los que se requiere explorar relaciones entre elementos de un arreglo o secuencia, reduciendo el **tiempo de ejecución** que normalmente sería más costoso en una solución ingenua. Este enfoque se utiliza para resolver problemas de **búsqueda**, **suma**, o **comparación** que suelen involucrar pares de elementos o subarreglos.

Casos comunes donde se aplica:

1. **Búsqueda de pares con ciertas propiedades**:
    - Un ejemplo típico es encontrar dos números en un arreglo que sumen un valor dado. Al usar dos punteros, uno desde el inicio y otro desde el final, es posible reducir una solución (O(n^2)) a (O(n)).
2. **Manipulación de subarreglos o subsecuencias**:
    - En problemas como encontrar el subarreglo más largo con una propiedad específica, o sumar un rango de elementos, el uso de dos punteros que se mueven en una dirección controlada permite optimizar el proceso.
3. **Problemas de fusión o comparación de listas ordenadas**:
    - Comparar elementos en dos listas ordenadas para encontrar intersecciones o unirlas eficientemente es otra aplicación del método.

Problema: Encontrar un subarreglo cuya suma sea igual a un valor objetivo (x).

Dado un arreglo de enteros positivos y un objetivo **x**, queremos encontrar un subarreglo cuya suma sea exactamente **x**.

- **Ejemplo**:
  - Array: $[1, 3, 2, 5, 1, 1, 2, 3]$
  - Objetivo $x = 8$

¿Cómo resolvemos esto?

Idea principal del algoritmo

- Mantenemos dos punteros que apuntan al inicio y al final del subarreglo.
- **Paso a paso**:
  - Avanzamos el puntero izquierdo para reducir la suma.
  - Avanzamos el puntero derecho para aumentar la suma hasta encontrar la suma objetivo.

Código en C++

```
int main() {
    vector<int> arr = {1, 3, 2, 5, 1, 1, 2, 3};
    int x = 8;

    if (subarraySum(arr, n, x)) {
        cout << "Subarreglo encontrado.\n";
    } else {
        cout << "No se encontró subarreglo.\n";
    }

    return 0;
}
```

```cpp
bool subarraySum(vector<int>& arr, int n, int x) {
    int left = 0, right = 0;
    int current_sum = 0;
    while (right < arr.size()) {
        current_sum += arr[right]; // sumamos el valor de arr[right]

        while (current_sum > x && left <= right) {
            current_sum -= arr[left]; // reducimos el valor de arr[left]
            left++; // movemos el puntero izquierdo
        }

        if (current_sum == x) {
            return true; // Se encontró el subarreglo con suma x
        }

        right++; // movemos el puntero derecho
    }
    return false; // No se encontró subarreglo
}
```

Explicación del Código

- **Punteros**: Los punteros serian `left` y `right`, estos se utilizan para definir el inicio y el final del subarreglo.
- **Movimiento de punteros**:
  - El puntero derecho avanza cuando la suma es menor o igual que `x`.
  - El puntero izquierdo avanza cuando la suma supera `x`.
- **Complejidad**: Ambos punteros se mueven **O(n)** veces, lo que hace que el algoritmo sea eficiente.

Vamos a hacer una prueba de escritorio paso a paso usando el código en C++ que implementa el método de dos punteros para encontrar un subarreglo cuya suma sea igual a `x = 8`. Seguiremos el código y explicaremos cómo funcionan los punteros `left` y `right` junto con la variable `current_sum`.

Array: `[1, 3, 2, 5, 1, 1, 2, 3]`, objetivo `x = 8`

Paso 1:

- **Posición inicial**:
  - `left = 0` (apunta al elemento `1`)
  - `right = 0` (apunta al elemento `1`)
  - `current_sum = 0`
- **Acción**: Sumamos `*right` al `current_sum`.
  - `current_sum = 1`
- **Evaluación**: `current_sum < x`, así que movemos el puntero `right` una posición a la derecha.

Paso 2:

- **Posición**:
  - `left = 0` (apunta al elemento `1`)
  - `right = 1` (apunta al elemento `3`)
  - `current_sum = 1`
- **Acción**: Sumamos `*right` al `current_sum`.
  - `current_sum = 1 + 3 = 4`
- **Evaluación**: `current_sum < x`, así que movemos el puntero `right` una posición a la derecha.

Paso 3:

- **Posición**:
  - `left = 0` (apunta al elemento `1`)
  - `right = 2` (apunta al elemento `2`)
  - `current_sum = 4`
- **Acción**: Sumamos `*right` al `current_sum`.
  - `current_sum = 4 + 2 = 6`
- **Evaluación**: `current_sum < x`, así que movemos el puntero `right` una posición a la derecha.

Paso 4:

- **Posición**:
  - `left = 0` (apunta al elemento `1`)
  - `right = 3` (apunta al elemento `5`)
  - `current_sum = 6`
- **Acción**: Sumamos `*right` al `current_sum`.
  - `current_sum = 6 + 5 = 11`
- **Evaluación**: `current_sum > x`, así que movemos el puntero `left` una posición a la derecha para reducir la suma.

Paso 5:

- **Posición**:
  - `left = 1` (apunta al elemento `3`)
  - `right = 3` (apunta al elemento `5`)
  - `current_sum = 11`
- **Acción**: Restamos `*left` del `current_sum` y movemos `left`.
  - `current_sum = 11 - 1 = 10`
- **Evaluación**: `current_sum > x`, así que movemos el puntero `left` una posición más a la derecha.

Paso 6:

- **Posición**:
  - `left = 2` (apunta al elemento `2`)
  - `right = 3` (apunta al elemento `5`)
  - `current_sum = 10`
- **Acción**: Restamos `*left` del `current_sum` y movemos `left`.
  - `current_sum = 10 - 3 = 7`
- **Evaluación**: `current_sum < x`, así que movemos el puntero `right` una posición a la derecha.

Paso 7:

- **Posición**:
  - `left = 2` (apunta al elemento `2`)
  - `right = 4` (apunta al elemento `1`)
  - `current_sum = 7`
- **Acción**: Sumamos `*right` al `current_sum`.
  - `current_sum = 7 + 1 = 8`
- **Evaluación**: `current_sum == x`. ¡Subarreglo encontrado!

Subarreglo final:

El subarreglo encontrado es `[2, 5, 1]`, que suma exactamente `8`.

Resumen paso a paso:

1. Inicialmente, la suma es $1+3+2=6$. Avanzamos el puntero `right`.
2. La suma se convierte en $1+3+2+5=11$. Avanzamos el puntero `left` para reducir la suma.
3. Con la nueva suma $3+2+5=10$, avanzamos `left` nuevamente.
4. Finalmente, obtenemos $2+5+1=8$. Subarreglo encontrado.

Conclusión

- El método de dos punteros es una técnica poderosa para resolver problemas de subarreglos.
- Usando punteros en C++, podemos lograr un código eficiente y claro.
- La solución tiene una complejidad temporal de O(n), ideal para grandes conjuntos de datos.

Problemas

- **580A** Kefa and First Steps ⮭
- **6A** Triangle ⮭

Referencias

- GeeksforGeeks. (n.d.). *Two pointers technique*. GeeksforGeeks. https://www.geeksforgeeks.org/two-pointers-technique/
- Halldórsson, M. M., & Laaksonen, A. (n.d.). *Competitive programmer's handbook*. CSES. https://cses.fi/book/book.pdf
- Nazzal, F. (2023, August 24). *Intro to algorithms: Two pointers technique*. Medium. https://medium.com/geekculture/intro-to-algorithms-two-pointers-technique-b37f962eab5