Informe de Laboratorio 02

Tema: Búsqueda en vector ordenado. Búsqueda Binaria

Nota

Estudiante	Escuela	Asignatura
Reyser Julio Zapata Butrón rzapata@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Análisis Y Diseño de Algoritmos Semestre: IV Código: 1702231

Laboratorio	Tema	Duración
02	Búsqueda en vector ordenado. Búsqueda Binaria	02 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - B	24 septiembre 2024	24 septiembre 2024

1. Código Fuente

El siguiente código es el que se usará para analizar los ejercicios propuestos en el laboratorio.

```cpp
#include <iostream>
#include <filesystem>
#include <string>
#include <cstdlib>

int busquedaBinariaOriginal(int x, int n, int v[])
{
  int i = -1, d = n;
  while (i < d - 1)
  {
    int m = (i + d) / 2;
    if (v[m] < x)
      i = m;
    else
      d = m;
  }
  return d;
}

int main(int argc, char *argv[])
{
  std::string fileName = std::filesystem::path(argv[0]).filename().string();

```

```cpp
  if (argc < 3)
  {
    std::cout << "\n[-] Uso: ./" << fileName << " <x> <numero1> <numero2> ... <numero n>\n"
              << std::endl;
    return 1;
  }

  int x = std::stoi(argv[1]);
  int n = argc - 2;
  int *numbers = new int[n];

  std::cout << "\n[*] v: ";
  for (int i = 0; i < n; i++)
  {
    numbers[i] = std::stoi(argv[i + 2]);
    std::cout << argv[i + 2] << ", ";
  }

  std::cout << "\n[*] x: " << x << "\n[*] n: " << n << std::endl;

  int r1 = busquedaBinariaOriginal(x, n, numbers);
  std::cout << "\n\n[+] Result : " << r1 << std::endl;

  printf("\n\n");
  delete argv;
  return 0;
}
```

Listing 1: main.cpp

Ejecución del código

```
> .\main.exe

[-] Uso: ./main.exe <x> <numero1> <numero2> ... <numero n>

> .\main.exe 3 1 2 3 4 5 6 7 8

[*] v: 1, 2, 3, 4, 5, 6, 7, 8,
[*] x: 3
[*] n: 8


[+] Result : 2
```

2. Ejercicios

2.1. Ejercicio 1

- Para evitar valores de índice fuera del rango $0..n-1$, podríamos cambiar la línea $i=-1; d=n$ de la función `busquedaBinaria` a través de las siguientes tres líneas. Analice esta variante de código.

```
if (v[n-1] < x) return n;
if (x <= v[0]) return 0;
// ahora v[0] < x <= v[n-1]
i = 0; d = n-1;
```

El cambio introduce dos condiciones adicionales antes de iniciar el ciclo `while`, lo cual garantiza que no se intenten acceder a índices fuera del rango permitido. La condición `if (v[n-1] <x)` maneja el caso en que el valor buscado x es mayor que todos los valores del vector, devolviendo n como el índice en el que debería estar x (justo después del último elemento). Por otro lado, `if (x <= v[0])` devuelve 0 si x es menor o igual al primer valor del vector, lo que indica que x debería estar al principio.

En conclusión, las líneas de código añadidas nos aseguran que el valor buscado está dentro del rango ((v[0], v[n-1]), y por eso se ajustan los valores de i a 0 y d a $n-1$.

2.2. Ejercicio 2

- Responda las siguientes preguntas sobre la función `busquedaBinaria`. ¿Qué sucede si `while (i <d-1)` se reemplaza por `while (i <d)`? O por `while (i <= d-1)`? ¿Qué pasa si intercambiamos `(i + d)/2` por `(i + d - 1)/2` o por `(i + d + 1)/2` o por `(d - i)/2`? ¿Qué sucede si `if (v[m] <x)` se reemplaza por `if (v[m] <= x)`? ¿Qué sucede si `i = m` se reemplaza por `i = m+1` o por `i = m-1`? ¿Qué sucede si `d = m` se reemplaza por `d = m+1` o `d = m-1`?

a. Cambios en el ciclo `while (i <d-1)`

- **Reemplazo por `while (i <d)`:** Esto rompe el invariante, ya que el ciclo continuará hasta que $i == d$, lo cual no pasará, ya que i nunca debería igualarse a d en el algoritmo. Esto lleva a una **condición infinita**.
- **Reemplazo por `while (i <= d-1)`:** Al modificar esto, ocurre algo similar con la anterior modificación, generará una **condición infinita**.

b. Cambios en el cálculo de `(i + d)/2`

- **Reemplazo por** $(i + d - 1)/2$: Esto moverá el valor de m hacia la izquierda, haciendo que el algoritmo converja más rápido en algunos pocos casos, pero podría perder el valor correcto si no se ajustan las condiciones para actualizar i y d en base a esta nueva lógica.
- **Reemplazo por** $(i + d + 1)/2$: Similar a la modificación anterior, pero esta vez moverá el valor de m hacia la derecha, lo cual también afectará la convergencia y requeriría ajustes en la lógica de actualización de i y d.
- **Reemplazo por** $(d - i)/2$: Esto romperá la lógica del algoritmo, ya que cambia completamente el cálculo del punto medio.

c. Cambios en la condición `if (v[m] <x)`:

- **Reemplazo por `if (v[m] <= x)`:** Esto altera el resultado del algortimo. En lugar de buscar el punto exacto donde x se encuentra entre $v[i]$ o $v[d]$, se permitiría que $v[m] == x$ pase al siguiente lado de la búsqueda, lo cual lleva a resultados incorrectos al intentar encontrar el **índice correcto**.

d. Cambios en la asignación de $i = m$:

- **Reemplazo por** $i = m + 1$: Esto altera el algoritmo, ya que al poner $m + 1$ afecta el valor de retorno, en muchos casos sumandole al resultado, generando un **indice incorrecto**.
- **Reemplazo por** $i = m - 1$: Esto rompe el algoritmo, ya que estamos retrocediendo innecesariamente, lo cual genera en un **ciclo infinito**.

e. Cambios en la asignación de $d = m$:

- **Reemplazo por** $d = m+1$: Esto rompe el algortimo, al mover **d** a la derecha con $m+1$ genera una **condición infinita**.
- **Reemplazo por** $d = m-1$: Esto rompe el algoritmo, al mover **d** a la izquierda con $m-1$ en muchos casos genera un **retorno incorrecto** o una **condicion infninita**.

2.3. Ejercicio 3

- Ejecute la función `busquedaBinaria` con $n = 16$. ¿Cuáles son los valores posibles de m en la primera iteración? ¿Cuáles son los posibles valores de m en la segunda iteración? ¿En la tercera? ¿En la cuarta?

1. **Primera iteración:** Siempre será 7.
2. **Segunda iteración:** Los posibles valores son 3 y 11.
3. **Tercera iteración:** Los posibles valores son 1, 5, 9, 13.
4. **Cuarta iteración:** Los posibles valores son 0, 2, 4, 6, 8, 10, 12, 14.
5. **Quinta iteración:** El único valor posible es 15.

2.4. Ejercicio 4

- En la función `busquedaBinaria`, ¿es cierto que m está en $0..n-1$ siempre que se ejecuta la sentencia `if (v[m] <x)`? (Tenga en cuenta que i y d pueden no estar en $0..n-1$.)

Sí, porque en cada iteración m se calcula como $\frac{i+d}{2}$, y tanto i como d siempre están en el rango de $[-1, n]$. Como el ciclo while se detiene antes de que $i == d$, m siempre se encuentra dentro de los límites de $0..n-1$.

2.5. Ejercicio 5

- Variantes de algoritmos. Escriba una versión de la función de `busquedaBinaria` que tenga la siguiente invariante: al comienzo de cada iteración, $v[i-1] < x \leq v[d]$. Repita con $v[i-1] < x \leq v[d+1]$. Repita con $v[i] < x \leq v[d+1]$.

Con invariante $v[i-1] < x \leq v[d]$:

```
int busquedaBinaria(int x, int n, int v[]) {
    int i = 0, d = n;
    while (i < d) {
        int m = (i + d) / 2;
        if (v[m] < x) i = m + 1;
        else d = m;
    }
    return d;
}
```

Con invariante $v[i-1] < x \leq v[d+1]$:

```
int busquedaBinaria(int x, int n, int v[]) {
    int i = 0, d = n - 1;
    while (i <= d) {
        int m = (i + d) / 2;
        if (v[m] < x) i = m + 1;
```

```
            else d = m - 1;
        }
        return i;
    }
```

Con invariante $v[i] < x \leq v[d+1]$**:**

```
    int busquedaBinaria(int x, int n, int v[]) {
        int i = 0, d = n - 1;
        while (i <= d) {
            int m = (i + d) / 2;
            if (v[m] <= x) i = m + 1;
            else d = m - 1;
        }
        return i;
    }
```

3. Repositorio de Github

- Repositorio de Github donde se encuentra el actual laboratorio
  `https://github.com/ReyserLynnn/ada-lab-b-24b/tree/main/laboratorio02/src`
- Repositorio de Github donde se encuentran los laboratorios del curso
  `https://github.com/ReyserLynnn/ada-lab-b-24b.git`