Patrones algorítmicos

Objetivo: Implementar patrones algorítmicos de carga, recorrido y búsqueda en estructuras con posiciones contiguas de memoria, homogéneas, no homogéneas y combinadas.

Introducción

- Breve repaso de los conceptos básicos de arrays.
- Arrays como String →Detalles extras

  ¿Cómo asignar texto a una variable?

```
// Está OK
char nuevoTexto[10] = "nuevo";

 // No está OK
char otroTexto[10];
otroTexto = "nuevo";
```

```
#include <cstring> // Necesario para usar la función strcpy

int main() {
    char nuevoTexto[10];
    strcpy(nuevoTexto, "nuevo");

    return 0;
}
```

  - ¿Por qué cuando cargo frases solo se queda con al primer palabra?

    La función `cin` se detiene en el primer espacio en blanco (como el espacio entre "hola" y "mundo").

```
#include <iostream>
using namespace std;

int main() {
    char nombre[30];
    cin.getline(nombre, sizeof(nombre));
```

```
    string nombre2;
    getline(cin,nombre2);

    cout << nombre;
    cout << nombre2;
    return 0;
}
```

Patrones de carga

1. Carga secuencial: El patrón de carga secuencial implica llenar un array secuencialmente, asignando valores a cada posición de forma consecutiva desde el inicio hasta el final. En este ejemplo, el array se carga secuencialmente con los valores del 1 al 5.

```
// Carga secuencial
for (int i = 0; i < SIZE; i++) {
    array[i] = i + 1; // Asignar valor secuencial a cada posición
}
```

2. Carga directa: El patrón de carga directa implica asignar valores específicos a posiciones determinadas del array, sin seguir un orden secuencial. En este ejemplo, se realizó una carga directa asignando valores específicos a posiciones determinadas del array.

```
int array[5]; // Declaración del array

// Carga directa
array[0] = 10; // Asignar valor 10 a la primera posición
array[2] = 25; // Asignar valor 25 a la tercera posición
array[4] = 50; // Asignar valor 50 a la quinta posición
```

3. Carga ordenada: El patrón de carga ordenada implica llenar el array con valores en orden ascendente o descendente, utilizando un criterio específico.

```
const int SIZE = 5; // Tamaño del array
int array[SIZE]; // Declaración del array

// Carga ordenada en orden ascendente
for (int i = 0; i < SIZE; i++) {
    array[i] = i * 10; // Asignar valores en orden ascendente (0, 10, 20, 30, 40)
```

```
}

// Carga ordenada en orden descendente
for (int i = 0; i < SIZE; i++) {
    array[i] = (SIZE - 1 - i) * 10; // Asignar valores en orden descendente (40, 30, 20,
 10, 0)
}
```

Patrones de recorrido

1. Recorrido total/secuencial: El patrón de recorrido total implica visitar todos los elementos del array en secuencia, desde el primer elemento hasta el último.
   En este ejemplo, se realiza un recorrido total del array y se imprime cada uno de sus elementos.

```
const int SIZE = 5; // Tamaño del array
int array[SIZE] = {1, 2, 3, 4, 5}; // Array inicializado

// Recorrido total
for (int i = 0; i < SIZE; i++) {
    // Realizar acción en cada elemento, por ejemplo, imprimir su valor
    cout << array[i] << " ";
}
// Salida: 1 2 3 4 5
```

2. Recorrido parcial: El patrón de recorrido parcial implica visitar solo un subconjunto de elementos del array, generalmente definido por un rango específico. En este ejemplo, se realiza un recorrido parcial del array, omitiendo el primer y el último elemento, y se imprime cada uno de los elementos en el rango definido.

```
const int SIZE = 5; // Tamaño del array
int array[SIZE] = {1, 2, 3, 4, 5}; // Array inicializado

// Recorrido parcial
for (int i = 1; i < SIZE - 1; i++) {
    // Realizar acción en cada elemento del rango, por ejemplo, imprimir su valor
    cout << array[i] << " ";
}
// Salida: 2 3 4
```

3. Recorrido en ambas direcciones: El patrón de recorrido en ambas direcciones implica visitar los elementos del array tanto de forma ascendente como descendente, en

diferentes etapas del recorrido. En este ejemplo, se realiza un recorrido ascendente y descendente del array, imprimiendo los elementos en ambos órdenes.

```
const int SIZE = 5; // Tamaño del array
int array[SIZE] = {1, 2, 3, 4, 5}; // Array inicializado

// Recorrido ascendente
for (int i = 0; i < SIZE; i++) {
    // Realizar acción en cada elemento en orden ascendente, por ejemplo, imprimir su val
or
    cout << array[i] << " ";
}
// Salida: 1 2 3 4 5


// Recorrido descendente
for (int i = SIZE - 1; i >= 0; i--) {
    // Realizar acción en cada elemento en orden descendente, por ejemplo, imprimir su va
lor
    cout << array[i] << " ";
}
// Salida: 5 4 3 2 1
```

4. Recorrido con corte de control: El patrón de recorrido con corte de control implica detener el recorrido cuando se cumple una condición específica, en lugar de visitar todos los elementos del array. En este ejemplo, el bucle recorre el array y se detiene cuando encuentra un valor que es igual al valor anterior. En este caso, el valor 2 se repite consecutivamente, por lo que el recorrido se interrumpe después de imprimir el primer valor 2.

```
const int SIZE = 5; // Tamaño del array
int array[SIZE] = {1, 2, 2, 3, 4}; // Array inicializado

// Recorrido con corte de control
for (int i = 1; i < SIZE; i++) {
    // Realizar acción en cada elemento hasta que se cumpla la condición de corte
    if (array[i] == array[i - 1]) {
        break; // Interrumpir el recorrido cuando el valor sea igual al anterior
    }
    // Realizar acción en el elemento actual, por ejemplo, imprimir su valor
    cout << array[i] << " ";
}
// Salida: 2
```

Patrones de búsqueda

1. Búsqueda lineal/secuencial: El patrón de búsqueda lineal implica buscar un elemento en un array recorriendo secuencialmente cada elemento hasta encontrar una coincidencia.

```
bool busquedaLineal(const int array[], int size, int valorBuscado, int& indiceEncontrado) {
    for (int i = 0; i < size; i++) {
        if (array[i] == valorBuscado) {
            indiceEncontrado = i; // Almacena el índice donde se encontró el valor
            return true; // Valor encontrado
        }
    }
    return false; // Valor no encontrado
}
```

   Otra forma

```
int busquedaLineal(const int array[], int size, int valorBuscado) {
    for (int i = 0; i < size; i++) {
        if (array[i] == valorBuscado) {
            return i + 1; // Posición encontrada (1-indexed)
        }
    }
    return -1; // Valor no encontrado
}
```

2. Búsqueda binaria: El patrón de búsqueda binaria se aplica cuando el **array está ordenado** y se busca un valor específico dividiendo el rango de búsqueda a la mitad en cada iteración.

```
bool busquedaBinaria(const int array[], int size, int valorBuscado) {
    int inicio = 0;
    int fin = size - 1;

    while (inicio <= fin) {
        int medio = (inicio + fin) / 2;

        if (array[medio] == valorBuscado) {
            return true; // Valor encontrado
        } else if (array[medio] < valorBuscado) {
            inicio = medio + 1; // El valor está en la mitad derecha
        } else {
            fin = medio - 1; // El valor está en la mitad izquierda
        }
    }
```

```
    return false; // Valor no encontrado
}
```

Otra forma

```
int busquedaBinaria(const int array[], int size, int valorBuscado) {
    int inicio = 0;
    int fin = size - 1;

    while (inicio <= fin) {
        int medio = (inicio + fin) / 2;

        if (array[medio] == valorBuscado) {
            return medio + 1; // Posición encontrada (1-indexed)
        } else if (array[medio] < valorBuscado) {
            inicio = medio + 1; // El valor está en la mitad derecha
        } else {
            fin = medio - 1; // El valor está en la mitad izquierda
        }
    }
    return -1; // Valor no encontrado
}
```

Patrones de ordenamiento

1. Simple:

```
void ordenar(int v[], int n) {
  int aux;
  int i, j, ord = 0;
  for(i = 0; i < n - 1 && ord == 0; i++) {
    ord = 1;
    for(j = 0; j < n - 1; j++) {
      if(v[j] > v[j + 1]) {
        aux = v[j];
        v[j] = v[j + 1];
        v[j + 1] = aux;
        ord = 0;
      }
    }
  }
}
```

2. De selección: El algoritmo de selección es una técnica sencilla pero efectiva para ordenar un array. El proceso comienza seleccionando repetidamente el elemento más

pequeño y colocándolo en su posición correcta. A medida que avanzamos en el array, encontramos el elemento mínimo restante y lo intercambiamos con el elemento en la posición actual. Este proceso continúa hasta que todos los elementos estén en su lugar.

```
void ordenarPorSelección(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < size; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        // Intercambio de elementos
        int temp = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }
}
```

3. De inserción: El algoritmo de ordenamiento de inserción es una técnica simple y eficiente para ordenar un array. En este enfoque, se divide el array en una parte ordenada y una parte desordenada. En cada iteración, se toma un elemento de la parte desordenada y se inserta en la posición correcta dentro de la parte ordenada, desplazando los elementos mayores según sea necesario.

```
void insertionSort(int arr[], int size) {
    for (int i = 1; i < size; i++) {
        int key = arr[i];
        int j = i - 1;

        // Mover los elementos mayores hacia la derecha
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }

        // Insertar el elemento en su posición correcta
        arr[j + 1] = key;
    }
}
```

4. De burbuja: En este enfoque, los elementos adyacentes se comparan y se intercambian si están en el orden incorrecto. Este proceso se repite hasta que no haya

más intercambios necesarios, lo que indica que el array está completamente ordenado.

```
void ordenamientoBurbuja(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Intercambio de elementos
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
```

5. Plus → Podemos también insertarlos de forma ordenada.

```
void insertarOrdenado(int arr[], int size, int element) {
    int i = size - 1;

    // Desplazar elementos mayores hacia la derecha
    while (i >= 0 && arr[i] > element) {
        arr[i + 1] = arr[i];
        i--;
    }

    // Insertar el elemento en su posición correcta
    arr[i + 1] = element;
}
```

```
int main() {
    int array[10];

    for (int i = 0; i < 10; i++) {
        int num;
        cin >> num;
        insertarOrdenado(array,i,num);
        mostrarArray(array,10);
    }
    return 1;
}
```

Ejercicios

Ordenamiento

1. Crear una función para ordenar un array de enteros de menor a mayor utilizando el algoritmo de selección.
2. Crear una función para ordenar un array de enteros de mayor a menor utilizando el algoritmo de inserción.
3. Siendo `Persona { int documento; string nombre}`. Crear una función para ordenar un array de Persona por su documento en orden ascendente utilizando el algoritmo de burbuja.
4. Siendo `Persona { int documento; string nombre}`. Crear una función para ordenar un array de Persona por su nombre en orden descendente utilizando el algoritmo de selección.
5. Siendo `Equipo { string nombre; int puntos }`. Crear una función que devuelva la tabla de posiciones.
6. Ordenar un array de cadenas de caracteres en orden alfabético utilizando el algoritmo de inserción.

Búsqueda

1. Ejercicio de búsqueda secuencial:
   Dado un array de números enteros, solicita al usuario ingresar un número y utiliza la búsqueda secuencial para determinar si ese número está presente en el array. Muestra un mensaje indicando si se encontró o no.
2. Ejercicio de búsqueda binaria:
   Dado un array de nombres, ordenar el array y luego solicitar al usuario ingresar un nombre y utiliza la búsqueda binaria para determinar si ese nombre está presente en el array. Muestra un mensaje indicando si se encontró o no.
3. Ejercicio de combinación de búsqueda secuencial y búsqueda binaria:
   Dado un array de números enteros ordenados de forma ascendente, solicita al usuario ingresar un número. Si el número ingresado es mayor que el último elemento del array, aplica la búsqueda secuencial para determinar si está presente. Si el número ingresado es menor o igual al último elemento del array, aplica la búsqueda binaria. Muestra un mensaje indicando si se encontró o no.
4. Ejercicio de búsqueda secuencial en un array de struct:
   a. Crea una estructura (struct) "Producto" con los atributos "nombre" y "precio".

b. Crea un array de objetos Producto y solicita al usuario ingresar un nombre de producto. Utiliza la búsqueda secuencial para determinar si ese producto está presente en el array y, en caso afirmativo, muestra su precio.

5. Ejercicio de búsqueda binaria en un array ordenado de fechas:
   Crea un array de fechas ordenadas cronológicamente. Solicita al usuario ingresar una fecha y utiliza la búsqueda binaria para determinar si esa fecha está presente en el array. Si se encuentra, muestra un mensaje indicando la posición de la fecha en el array.
   - Para manejar fechas usar un numero con este formato: AAAAMMDD
6. Crea una estructura llamada "Producto" con los atributos "nombre", "precio" y "stock" e implementa un sistema donde el usuario pueda ingresar los datos de 10 productos. Luego, el sistema solicitará al usuario que ingrese el nombre de un producto a buscar. Utilizando un algoritmo de búsqueda adecuado, se determinará si el producto está presente en el conjunto de productos cargados. Si se encuentra, se mostrará su nombre, precio y stock. Además, se brindará información adicional según el enunciado.
7. Crea una estructura llamada "Instagamer" con los atributos "nombre", "edad" y una estructura anidada llamada "Redes" con los atributos "instagram" y "twitter" e implementa un sistema donde el usuario pueda ingresar los datos de 10 Instagamers. Luego, el sistema solicitará al usuario que ingrese el nombre de un Instagamer a buscar. Utilizando un algoritmo de búsqueda adecuado, se determinará si el Instagamer está presente en el conjunto de Instagamers cargados. Si se encuentra, se mostrará su nombre, edad, instagram y twitter. Además, se brindará información adicional según el enunciado.
8. Crea una estructura llamada "Factura" con los atributos "numero", "fecha" y "monto" e implementa un sistema donde el usuario pueda ingresar los datos de 10 facturas. Luego, el sistema solicitará al usuario que ingrese un número de factura a buscar. Utilizando un algoritmo de búsqueda adecuado, se determinará si la factura está presente en el conjunto de facturas cargadas. Si se encuentra, se mostrará su número, fecha y monto. Además, se brindará información adicional según el enunciado.
9. Dada una estructura llamada Alumno con los atributos de 'nombre', 'edad' y 'promedio', implementa un sistema donde el usuario pueda ingresar los datos de 10 alumnos de una clase. Luego, el sistema solicitará al usuario que ingrese el nombre de un alumno

a buscar. Utilizando la búsqueda que considere correspondiente, se determinará si el alumno está presente en el conjunto de alumnos cargados. Si se encuentra, se mostrará su nombre, edad y promedio. Además, se informará la cantidad de alumnos que tienen un promedio mayor que el del alumno buscado y la cantidad de alumnos que tienen un promedio menor que el del alumno buscado.

10. Crea una estructura llamada "Cliente" con los atributos "nombre", "edad" y "saldo" e implementa un sistema donde el usuario pueda ingresar los datos de 10 clientes. Luego, el sistema solicitará al usuario que ingrese el nombre de un cliente a buscar. Utilizando un algoritmo de búsqueda adecuado, se determinará si el cliente está presente en el conjunto de clientes cargados. Si se encuentra, se mostrará su nombre, edad y saldo y cuantos clientes tienen un saldo más alto que el cliente encontrado.