INTEGRANTES

Yunisse Peña	1-18-2568
Félix Artiles	2-16-1692
Alberto García	2-17-1097
Diógenes Camacho	1-16-1383

Facilitador

Prof. Iván Mendoza

ALGORITMOS PARALELOS
PROYECTO FINAL

5 de diciembre del año 2021

Santiago de los Caballeros, Rep. Dom.

ÍNDICE

8. ALGORITMOS DE BÚSQUEDA Y ORDENAMIENTO

Los algoritmos de búsqueda son las instrucciones que nos permiten ubicar los elementos dentro de una estructura de datos. Mientras que, los algoritmos de ordenamiento se relacionan con los de búsqueda, ordenando los datos ya sea de mayor a menor o viceversa.

A continuación, se va a mostrar el pseudocódigo y la implementación en el lenguaje de programación C# de ciertos algoritmos de búsqueda y ordenamiento. Es necesario aclarar que el uso de la variable watch y su asignación a un TextBox (caja de texto, en C#) es para llevar un control del tiempo de ejecución de dichos algoritmos.

8.1. BÚSQUEDA SECUENCIAL

Este método consiste en comparar el elemento a buscar con cada elemento del arreglo con la finalidad de llegue a encontrar la posición de dicho elemento.

Este método se utiliza cuando el contenido de un vector no se puede encontrar o no se puede ordenar. Implica buscar un elemento comparando secuencialmente el elemento (de ahí el nombre) con cada elemento de la matriz o conjunto de datos hasta que se encuentra, o hasta que se alcanza el final de la matriz. La existencia de un elemento se puede determinar desde el momento en que se coloca, pero no podemos garantizar que no exista hasta que analicemos todos los elementos de la disposición.

8.1.1. PSEUDOCÓDIGO

```
Funcion BusquedaSecuencial(vector, tamano)
    posicion = 0
    encontrado = falso

    Mientras posicion < tamaño(vector) y encontrado = falso:
        Si vector[posicion] == numero:
            encontrado = verdadero
        De lo contrario:
            posicion = posicion +1

    Retornar encontrado
```

8.1.2. CÓDIGO

```
public void BusquedaSecuencial(TextBox txt, int[] vector, int n)
{
    Stopwatch watch = new Stopwatch();
    watch.Start();

    Boolean x = false;

    for (int a = 0; a < vector.Length; a++)
    {
        if (vector[a] == n)
        {
            txt.Text = "El número " + n + " se encuentra en la posición " + a + 1 + ".";
            x = true;
        }
    }

    if (!x)
    {
        txt.Text = "El número " + n + " no se encuentra en el arreglo.";
    }

    watch.Stop();
    txt.Text = watch.Elapsed.TotalMilliseconds.ToString("0.00") + " ms";
}
```

8.2. BÚSQUEDA BINARIA

Es un algoritmo que sirve para encontrar elementos en una lista ordenada, generalmente unidireccional, de una manera muy eficiente. Este funciona dividiendo a la mitad la lista deseada en los fundamentos de bit (que contiene dos estados), hasta acortar y encontrar la ubicación del valor buscado.

Este algoritmo tiene una condición única y es que los valores del vector estén ordenados de menor a mayor.

8.2.1. PSEUDOCÓDIGO

```
Funcion BusquedaBinaria(vector, numero):
    izquierda, derecha = 0, tamaño(vector) - 1

    Mientras izquierda <= derecha:
        indiceDelElementoDelMedio = (izquierda + derecha) / 2
        elementoDelMedio = vector[indiceDelElementoDelMedio]

        Si elementoDelMedio == numero:
            return indiceDelElementoDelMedio

        Si numero < elementoDelMedio:
            derecha = indiceDelElementoDelMedio - 1
        De lo contrario:
            izquierda = indiceDelElementoDelMedio + 1

    Retornar -1
```

8.2.2. CÓDIGO

```
public void BusquedaBinaria(TextBox txt, int[] vector, int n)
{
    Stopwatch watch = new Stopwatch();
    watch.Start();

    Array.Sort(vector);

    int l = 0, h = vector.Length - 1;
    int m = 0;
    bool found = false;

    while (l <= h && found == false)
    {
        m = (l + h) / 2;

        if (vector[m] == n)
            found = true;

        if (vector[m] > n)
            h = m - 1;
        else
            l = m + 1;
    }

    if (found == false)
    {
        txt.Text = String.Format("\nEl elemento {0} no está en el arreglo.", n);
    }
    else
    {
        txt.Text = String.Format("\nEl elemento {0} está en la posición: {1}", n, m + 1);
    }

    watch.Stop();
    txt.Text = watch.Elapsed.TotalMilliseconds.ToString("0.00") + " ms";
}
```

8.3. ALGORITMO DE ORDENAMIENTO DE LA BURBUJA

Este es un algoritmo que nos permite ordenar vectores. Su funcionamiento va en un recorrido por el arreglo, donde comparamos cada elemento de este. Normalmente, cuando se trabaja con el ordenamiento de la burbuja, se busca que el vector quede en orden ascendente.

8.3.1. PSEUDOCÓDIGO

```
Funcion OrdenamientoBurbuja(vector)
    i, j, aux;

    Para i = 0 hasta tamaño(vector) hacer:
        Para j = 0 hasta j < i - 1 hacer:
            Si vector[i] > vector[j]:
                aux = vector[i]
                vector[i] = vector[j]
                vector[j] = aux;
            j = j + 1
        i = i + 1

    Para l = 0 hasta tamaño(vector) hacer:
        Imprimir vector[l];
        l = l + 1
```

8.3.2. CÓDIGO

```
public void OrdenamientoDeLaBurbuja(TextBox txt, int[] vector)
{
    Stopwatch watch = new Stopwatch();
    watch.Start();

    int t;

    for (int a = 1; a < vector.Length; a++)
    {
        for (int b = vector.Length - 1; b >= a; b--)
        {
            if (vector[b - 1] > vector[b])
            {
                t = vector[b - 1];
                vector[b - 1] = vector[b];
                vector[b] = t;
            }
        }
    }

    for (int f = 0; f < vector.Length; f++)
    {
        txt.Text += vector[f] + "  ";
    }

    watch.Stop();
    txt.Text = watch.Elapsed.TotalMilliseconds.ToString("0.00") + " ms";
}
```

8.4. ALGORITMO DE ORDENAMIENTO RÁPIDO (QUICKSORT)

Es un algoritmo ordenado muy eficiente, su codificación es compleja y debemos hacer uso de la recursividad, se usa esta misma porque este trata de subdividir el vector, ya que pueden deseleccionar pivotes al principio, medio y final. Este algoritmo funciona llevando todos los pivotes menores a la izquierda y los mayores a la derecha.

8.4.1. PSEUDOCÓDIGO

```
Funcion OrdenamientoQuicksort(vector, principio, final)
    i, j, k, aux;
    i = principio
    j = final
    k = vector[(principio + final) / 2]

    Mientras i <= j:
        Mientras vector[i] < k:
            i = i + 1
        Mientras vector[j] < k:
            j = j - 1

        Si i <= j:
            aux = vector[i]
            vector[i] = vector[j]
            vector[j] = aux
            i = i + 1
            J = j - 1

    Si principio < j:
        OrdenamientoQuicksort(vector, principio, j)

    Si i < final:
        OrdenamientoQuicksort(vector, i, final)
```

8.4.2. CÓDIGO

```
public void QuickSort(TextBox txt, int[] vector, int principio, int final)
{
    Stopwatch watch = new Stopwatch();
    watch.Start();

    int i, j, k = vector.Length, centro; double pivote;

    centro = (principio + final) / 2;
    pivote = vector[centro];
    i = principio;
    j = final;

    do
    {
        while (vector[i] < pivote) i++;
        while (vector[j] > pivote) j--;

        if (i <= j)
        {
            int temp;
            temp = vector[i];
            vector[i] = vector[j];
            vector[j] = temp;
            i++;
            j--;
        }

        if (j < 0)
        {
            for (int l = 0; l < k; l++)
            {
                txt.Text += vector[l] + " ";
            }
        }
     }
     while (i <= j);

     if (principio < j)
     {
        QuickSort(txt, vector, principio, j);
     }

     if (i < final)
     {
        QuickSort(txt, vector, i, final);
     }

     watch.Stop();
     txt.Text = watch.Elapsed.TotalMilliseconds.ToString("0.00") + " ms";
}
```

8.5. ALGORITMO DE ORDENAMIENTO POR INSERCIÓN

Este tiene un funcionamiento parecido a la forma en que se ordenan las cartas. El vector se divide en una parte ordenada y otra sin ordenar, luego elementos sin ordenar se ponen del lado ordenado.

8.5.1. PSEUDOCÓDIGO

```
Funcion OrdenamientoInsercion(vector)
    Para i = 1 hasta tamaño(vector) hacer:
        aux = vector[i]
        j = i - 1

        Mientras j >= 0 y vector[j] > aux:
            vector[j + 1] = vector[j]
            j = j = 1

        vector[j + 1] = aux
```

8.5.2. CÓDIGO

```
public void OrdenamientoPorInsercion(TextBox txt, int[] vector)
{
    Stopwatch watch = new Stopwatch();
    watch.Start();

    for (int i = 0; i < vector.Length; i++)
    {
        int pos = i;
        int aux = vector[i];

        while (pos > 0 && vector[pos - 1] > aux)
        {
            vector[pos] = vector[pos - 1];
            pos--;
        }

        vector[pos] = aux;
    }

    for (int i = 0; i < vector.Length; i++)
    {
        txt.Text += vector[i] + " ";
    }

    watch.Stop();
    txt.Text = watch.Elapsed.TotalMilliseconds.ToString("0.00") + " ms";
}
```