EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE: AGOSTO – DICIEMBRE 2021

ING. SISTEMAS COMPUTACIONALES

ESTRUCTURA DE DATOS - 2SC3C

INVESTIGACIÓN
UNIDAD 4 – Estructuras no Lineales

GUILLEN MARTINEZ ANTHONY - 20210575

M.C.C. LUZ ELENA CORTEZ GALVAN

FECHA DE ENTREGA 05/11/2021

CONTENIDO DEL TRABAJO:

a) INDICE

b) Material de cada tema investigado.

4.1 – Arboles

Un árbol es una estructura (posiblemente no lineal) de datos compuesta de nodos, vértices y aristas. Un árbol que no tiene ningún nodo se llama árbol vacío o nulo. Un árbol que no está vacío consta de un nodo raíz y potencialmente muchos niveles de nodos adicionales que forman una jerarquía.

4.1.1- Conceptos de árbol general y binario

Árbol general: Los arboles representan las estructuras no lineales y dinámicas de datos más importantes en programación. Dinámicas porque las estructuras de árbol pueden cambiar durante la ejecución de un programa. No lineales, puesto que a cada elemento del árbol pueden seguirle varios elementos.

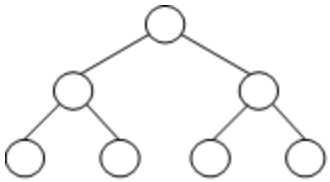
Árbol binario: Los árboles binarios son estructuras de datos muy similares a las listas doblemente enlazadas, en el sentido que tienen dos punteros que apuntan a otros elementos, pero no tienen una estructura lógica de tipo lineal o secuencial como aquellas, sino ramificada.

4.1.2- Clasificación de árboles

Los árboles se clasifican de la siguiente manera:

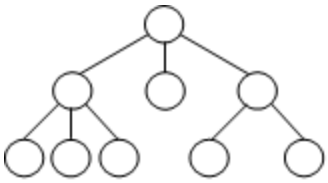
1. Árboles binarios.

Distintos
Similares
Equivalentes
Equilibrado
Completo



2. Árboles Multicaminos.

B
B+
B*
R
2-4



Programa ejemplo en C# que implemente árboles

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;

namespace ArbolBinarioOrdenado1
{
    public class ArbolBinarioOrdenado {
        class Nodo
        {
            public int info;
            public Nodo izq, der;
        }
        Nodo raiz;

        public ArbolBinarioOrdenado()
        {
            raiz=null;
        }

        public void Insertar (int info)
        {
            Nodo nuevo;
            nuevo = new Nodo ();
            nuevo.info = info;
            nuevo.izq = null;
            nuevo.der = null;
            if (raiz == null)
                raiz = nuevo;
            else
            {
                Nodo anterior = null, reco;
                reco = raiz;
                while (reco != null)
                {
                    anterior = reco;
                    if (info < reco.info)
                        reco = reco.izq;
                    else
                        reco = reco.der;
                }
                if (info < anterior.info)
                    anterior.izq = nuevo;
                else
                    anterior.der = nuevo;
            }
        }


        private void ImprimirPre (Nodo reco)
        {
            if (reco != null)
            {
                Console.Write(reco.info + " ");
                ImprimirPre (reco.izq);
                ImprimirPre (reco.der);
            }
```

```
        }

        public void ImprimirPre ()
        {
            ImprimirPre (raiz);
            Console.WriteLine();
        }

        private void ImprimirEntre (Nodo reco)
        {
            if (reco != null)
            {
                ImprimirEntre (reco.izq);
                Console.Write(reco.info + " ");
                ImprimirEntre (reco.der);
            }
        }

        public void ImprimirEntre ()
        {
            ImprimirEntre (raiz);
            Console.WriteLine();
        }


        private void ImprimirPost (Nodo reco)
        {
            if (reco != null)
            {
                ImprimirPost (reco.izq);
                ImprimirPost (reco.der);
                Console.Write(reco.info + " ");
            }
        }


        public void ImprimirPost ()
        {
            ImprimirPost (raiz);
            Console.WriteLine();
        }

        static void Main(string[] args)
        {
            ArbolBinarioOrdenado abo = new ArbolBinarioOrdenado ();
            abo.Insertar (100);
            abo.Insertar (50);
            abo.Insertar (25);
            abo.Insertar (75);
            abo.Insertar (150);
            Console.WriteLine ("Impresion preorden: ");
            abo.ImprimirPre ();
            Console.WriteLine ("Impresion entreorden: ");
            abo.ImprimirEntre ();
            Console.WriteLine ("Impresion postorden: ");
```

```
            abo.ImprimirPost ();
            Console.ReadKey();
        }
    }
}
```

4.1.3- Operaciones Básicas en Arboles Binarios.

1. **Inserción.** Es un procedimiento muy simple, sólo hay que cuidarse de no romper la estructura ni el orden del árbol.

2. **Eliminación.** La eliminación o el borrado en árboles binarios de búsqueda es otra operación bastante sencilla excepto en un caso.
   - Tras realizar la búsqueda del nodo a eliminar, observamos que el nodo no tiene hijos. Éste es el caso más sencillo, únicamente habrá que borrar el elemento y ya habremos concluido la operación.
   - Si tras realizar la búsqueda, nos encontramos con que tiene un sólo hijo. Éste caso también es sencillo, para borrar el nodo deseado, hacemos una especie de puente, el padre del nodo a borrar pasa a apuntar al hijo del nodo borrado.
   - El caso más complicado sucede cuando el nodo a borrar tiene dos hijos. Ahora, se debe sustituir el nodo a borrar por el mayor de los nodos menores del nodo borrado, o por el menor de los nodos mayores de aquél. Una vez realizada esta sustitución, se borra el nodo que sustituyó al nodo eliminado (operación que ya resultaría simple pues éste tendrá un hijo a lo sumo).

3. **Búsqueda.** La operación buscar es muy eficiente. El algoritmo compara el elemento a buscar con la raíz, si es menor, continúa la búsqueda por la rama izquierda; si es mayor, continúa por la derecha.

4. **Recorrido.** Este procedimiento se puede realizar de tres formas diferentes:
   - **Preorden:** primero el nodo raíz, luego el subárbol izquierdo y a continuación el subárbol derecho.
   - **In orden:** primero el subárbol izquierdo, luego la raíz y a continuación el subárbol derecho.
   - **Postorden:** primero el subárbol izquierdo, luego el subárbol derecho y a continuación la raíz.

4.1.4- Aplicaciones de Arboles Binarios.

Árbol de búsqueda binaria - Usado en muchas aplicaciones de búsqueda en las que los datos se introducen y salen constantemente, como la map y set objetos en las bibliotecas de muchos idiomas.
Partición del espacio binario - Se usa en casi todos los videojuegos 3D para determinar qué objetos necesitan ser renderizados.
Intentos binarios - Se utiliza en casi todos los enrutadores de banda ancha para almacenar tablas de enrutadores.
Árboles de hash - utilizado en programas p2p y firmas de imágenes especializadas en las que hay que verificar un hash, pero el archivo completo no está disponible.

4.2 – Grafos.

4.2.1 - Qué es un grafo

Los grafos son una composición de conjuntos de objetos que denominamos nodos. En ellos se almacena diferentes tipos de elementos o datos que podemos utilizar para procesar o conocer con fines específicos.

Adicionalmente estos nodos, suelen estar unidos o conectados a otros nodos a través de elementos que denominamos aristas.

Los nodos pertenecientes a un grafo pueden contener datos estructurada o no estructurada y al interrelacionarse con otros nodos producen relaciones interesantes que podemos analizar con diferentes finalidades.

4.2.1 - Tipos de grafos.

- Grafo simple. o simplemente grafo es aquel que acepta una sola arista uniendo dos vértices cualesquiera. Esto es equivalente a decir que una arista cualquiera es la única que une dos vértices específicos. Es la definición estándar de un grafo.
- Multígrafo. o pseudografo son grafos que aceptan más de una arista entre dos vértices. Estas aristas se llaman múltiples o lazos (loops en inglés). Los grafos simples son una subclase de esta categoría de grafos. También se les llama grafos no-dirigido.
- Grafo dirigido. Son grafos en los cuales se ha añadido una orientación a las aristas, representada gráficamente por una flecha
- Grafo etiquetado. Grafos en los cuales se ha añadido un peso a las aristas (número entero generalmente) o un etiquetado a los vértices.
- Grafo aleatorio. Grafo cuyas aristas están asociadas a una probabilidad.
- Hipergrafo. Grafos en los cuales las aristas tienen más de dos extremos, es decir, las aristas son incidentes a 3 o más vértices.
- Grafo infinito. Grafos con conjunto de vértices y aristas de cardinal infinito.

4.2.2 - Cómo se representan los grafos.

Listas de aristas

Una forma sencilla de representar un grafo es solo una lista, o un arreglo, de |E||E|vertical bar, E, vertical bar aristas, a la que llamamos una lista de aristas. Para representar una arista, solo tenemos un arreglo de dos números de vértices, o un arreglo de objetos que contienen los números de vértices sobre los que inciden las aristas. Si las aristas tienen pesos, agrega ya sea un tercer elemento al arreglo o más información al objeto, que dé el peso de la arista. Como cada arista contiene solo dos o tres números, el espacio total para una lista de aristas es \Theta (E) Θ (E)\Theta, left parenthesis, E, right parenthesis.

```
[ [0,1], [0,6], [0,8], [1,4], [1,6], [1,9], [2,4], [2,6], [3,4], [3,5],

[3,8], [4,5], [4,9], [7,8], [7,9] ]
```

Matrices de adyacencia

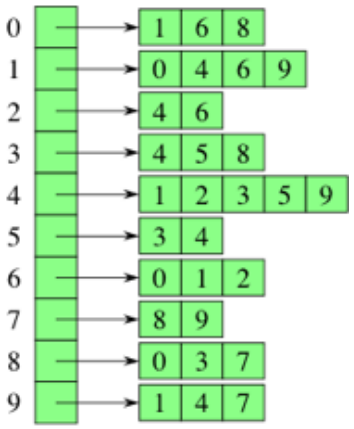
Para un grafo con |V||V|vertical bar, V, vertical bar vértices, una matriz de adyacencia es una matriz de |V| \times |V||V|×|V|vertical bar, V, vertical bar, times, vertical bar, V, vertical bar de ceros y unos, donde la entrada en el renglón iii y la columna jjj es 1 si y solo si la arista (i, j) (i, j) left parenthesis, i, comma, j, right parenthesis está en el grafo. Si quieres

indicar un peso de la arista, ponlo en la entrada del renglón iii, columna jjj y reserva un valor especial (tal vez null) para indicar una arista ausente. Aquí está la matriz de adyacencia para el grafo de la red social:

	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	0	0	1	0	1	0
1	1	0	0	0	1	0	1	0	0	1
2	0	0	0	0	1	0	1	0	0	0
3	0	0	0	0	1	1	0	0	1	0
4	0	1	1	1	0	1	0	0	0	1
5	0	0	0	1	1	0	0	0	0	0
6	1	1	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	1
8	1	0	0	1	0	0	0	1	0	0
9	0	1	0	0	1	0	0	1	0	0

Listas de adyacencia

Representar un grafo con listas de adyacencia combina las matrices de adyacencia con las listas de aristas. Para cada vértice iii, almacena un arreglo de los vértices adyacentes a él. Típicamente tenemos un arreglo de |V||V|vertical bar, V, vertical bar listas de adyacencia, una lista de adyacencia por vértice. Aquí está una representación de una lista de adyacencia del grafo de la red social:



4.2.3 – Operaciones básicas.

Insertar vértice

La operación de inserción de un nuevo vértice es una operación muy sencilla, únicamente consiste en añadir una nueva entrada en la tabla de vértices (estructura de datos que almacena los vértices) para el nuevo nodo. A partir de ese momento el grafo tendrá un vértice más, inicialmente aislado, ya que ninguna arista llegará a él.

Insertar arista

Esta operación es también muy sencilla. Cuando se inserte una nueva arista en el grafo, habrá que añadir un nuevo nodo a la lista de adyacencia (lista que almacena los nodos a los que un vértice puede acceder mediante una arista) del nodo origen, así si se añade la arista (A,C), se deberá incluir en la lista de adyacencia de A el vértice C como nuevo destino.

Eliminar vértice
Esta operación es inversa a la inserción de vértice. En este caso el procedimiento a realizar es la eliminación de la tabla de vértices del vértice en sí. A continuación habrá que eliminar las aristas que tuviesen al vértice borrado como origen o destino.
Eliminar arista
Mediante esta operación se borra un arco del grafo. Para llevar a cabo esta acción es necesario eliminar de la lista de adyacencia del nodo origen el nodo correspondiente al nodo destino.
Otras operaciones
Las operaciones adicionales que puede incluir un grafo son muy variadas. Además de las clásicas de búsqueda de un elemento o recorrido del grafo, también podemos encontramos con ejecución de algoritmos que busquen caminos más cortos entre dos vértices, o recorridos del grafo que ejecuten alguna operación sobre todos los vértices visitados, por citar algunas operaciones de las más usuales.

4.2.4 - Aplicaciones de grafos.
Gracias a la teoría de grafos se pueden resolver diversos problemas como por ejemplo la síntesis de circuitos secuenciales, contadores o sistemas de apertura. Se utiliza para diferentes áreas como pueden ser el Dibujo computacional o en áreas de Ingeniería.
Los grafos se utilizan también para modelar trayectos como el de una línea de autobús a través de las calles de una ciudad, en el que se pueden obtener caminos óptimos para el trayecto aplicando diversos algoritmos como puede ser el algoritmo de Floyd.
Para la administración de proyectos, utilizamos técnicas como técnica de revisión y evaluación de programas (PERT) en las que se modelan los mismos utilizando grafos y optimizando los tiempos para concretar los mismos.
Una importante aplicación de la teoría de grafos es en el campo de la informática, ya que ha servido para la resolución de importantes y complejos algoritmos. Un claro ejemplo es el Algoritmo de Dijkstra, utilizado para la determinación del camino más corto en el recorrido de un grafo con determinados pesos en sus vértices.
Dentro de este campo, un grafo es considerado un tipo de dato abstracto TAD.
La teoría de grafos también ha servido de inspiración para las ciencias sociales, en especial para desarrollar un concepto no metafórico de red social que sustituye los nodos por los actores sociales y verifica la posición, centralidad e importancia de cada actor dentro de la red. Esta medida permite cuantificar y abstraer relaciones complejas, de manera que la estructura social puede representarse gráficamente. Por ejemplo, una red social puede representar la estructura de poder dentro de una sociedad al identificar los vínculos (aristas), su dirección e intensidad y da idea de la manera en que el poder se transmite y a quiénes.
Programa ejemplo en C# que implemente grafos.

```
using
System;
          using System.IO;
          using System.Collections.Generic;
          namespace Graph_Things
          {
                  public class Graph
```

```
	{
		private int vertices,edges;
		private String text;
		private int [,] adjacenceMatrix, incidencyMatrix, mstMatrix;
		private int[][] distanceMatrix;
		public Graph ()
		{
			LoadFile ();
			PrintAM ();
			PrintIM ();
			PrintDM ();
			PrintMSTM ();
		}
		private void LoadFile(){
			Console.Write ("Entre com o nome do arquivo: ");
			string file = Console.ReadLine ();
			try{
				using(StreamReader sr = new StreamReader(file +
".txt")){
					text = sr.ReadToEnd();
				}

			}
			catch(Exception e){
				Console.WriteLine ("Error in load file.");
				Console.WriteLine (e.ToString ());
			}
			if (text != null) {
				string[] verticesStrings = text.Split (';');
				vertices = Convert.ToInt16 (verticesStrings [0]);
				adjacenceMatrix = new int[vertices , vertices];
				int count = 1;
				for (int line = 0 ; line < vertices ; line++) {
					for (int col = 0; col < vertices ; col++) {
						if (verticesStrings [count] != "\n")
							adjacenceMatrix [line,col] =
Convert.ToInt16 (verticesStrings [count]);
						else
							col--;
						count++;
					}
				}
				CalculateEdges ();
```

```
                        CreateIM ();
                        CreateDM ();
                        CreateMSTM ();
                }
        }
        private void CreateIM(){
                incidencyMatrix = new int[vertices, edges];
                int edgeIndex = 0;
                for (int line = 0 ; line < vertices ; line++) {
                        for (int col = 0; col < vertices ; col++) {
                                if (adjacenceMatrix [line, col] == 1) {
                                        incidencyMatrix [line, edgeIndex] = 1;
                                        incidencyMatrix [col, edgeIndex] = 1;
                                        edgeIndex++;
                                }
                        }
                }
        }
        private void CreateDM(){
                distanceMatrix = new int[vertices][];
                for (int i = 0; i < vertices; i++) {
                        distanceMatrix [i] = Dijkstra (i);
                }
        }
        private void CreateMSTM(){ // Cria a matriz da arvore geradora
minima
                mstMatrix = new int[vertices,vertices];
                int [] pred = Prim();
                for (int i = 1; i < pred.Length; i++) {
                        mstMatrix [i, pred [i]] = distanceMatrix[i][pred[i]];
                        mstMatrix [pred [i] ,i] = distanceMatrix[pred[i]][i];
                }
        }
        private int TakeMin(List<int> queue, int[] dist){
                int lesser = 0;
                foreach (int vertex in queue) {
                        lesser = vertex;
                        break;
                }
                foreach (int vertex in queue) {
                        if (dist [lesser] > dist [vertex])
                                lesser = vertex;
                }
```

```
                return lesser;
        }
        private int EdgeWeight(int a, int b){
                if (a == -1 || b == -1) {
                        return 9999;
                }else
                        return adjacenceMatrix [a, b];
        }
        private int ExtractDictionary(Dictionary<int,int> dict){
                int lesser = 0;
                List<int> list = new List<int> (dict.Keys);
                foreach (int vertex in list) {
                        lesser = vertex;
                        break;
                }
                foreach (int vertex in list) {
                        if (dict [lesser] > dict [vertex])
                                lesser = vertex;
                }

                return lesser;
        }
        public void PrintAM(){
                for (int line = 0 ; line < vertices ; line++) {
                        for (int col = 0; col < vertices ; col++) {
                                Console.Write (adjacenceMatrix
[line,col].ToString() + ";");
                        }
                        Console.Write ("\n");
                }
                Console.Write ("\n");
        }
        public void PrintMSTM(){
                for (int line = 0 ; line < vertices ; line++) {
                        for (int col = 0; col < vertices ; col++) {
                                Console.Write (mstMatrix [line,col].ToString()
+ ";");

                        }
                        Console.Write ("\n");
                }
                Console.Write ("\n");
        }
        public void PrintIM(){
```

```
                        for (int line = 0 ; line < vertices ; line++) {
                                for (int col = 0; col < edges ; col++) {
                                        Console.Write (incidencyMatrix
[line,col].ToString() + ";");
                                }
                                Console.Write ("\n");
                        }
                        Console.Write ("\n");
                }
                public void PrintDM(){
                        for (int line = 0 ; line < vertices ; line++) {
                                for (int col = 0; col < vertices ; col++) {
                                        Console.Write (distanceMatrix
[line][col].ToString() + ";");
                                }
                                Console.Write ("\n");
                        }
                        Console.Write ("\n");
                }

                public void CalculateEdges(){
                        edges = 0;
                        for (int line = 0 ; line < vertices ; line++) {
                                for (int col = 0; col < vertices ; col++) {
                                        if (adjacenceMatrix [line, col] == 1)
                                                edges++;
                                }
                        }
                }
                public int[,] GetMatrix(){
                        return adjacenceMatrix;
                }
                public int[] GetGraph(){
                        int[] graph = new int[2];
                        graph [0] = vertices;
                        graph [1] = edges;
                        return graph;
                }
                public int[] Dijkstra(int start){
                        int infinite = 9999;
                        List<int> queue = new List<int> ();
                        int[] pred = new int[vertices];
                        int[] dist = new int[vertices];
```

```
                        for (int vertex = 0; vertex < pred.Length; vertex++) {
                                pred [vertex] = -1;
                                if (vertex != start)
                                        dist [vertex] = infinite;
                                queue.Add (vertex);
                        }
                        while (queue.Count > 0) {
                                int u = TakeMin (queue,dist);
                                queue.Remove (u);
                                int[] neighbors = ReturnNeighbors (u);
                                for (int v = 0; v < neighbors.Length; v++) {
                                        if (neighbors [v] >= 1) {
                                                int aux = neighbors [v] + dist [u];
                                                if (aux < dist [v]) {
                                                        dist [v] = aux;
                                                        pred [v] = u;
                                                }
                                        }
                                }
                        }
                        return dist;
                }
                public int[] ReturnNeighbors(int vertex){
                        int[] neighbors = new int[vertices];
                        for (int i = 0; i < neighbors.Length; i++) {
                                neighbors [i] = adjacenceMatrix [vertex, i];
                        }
                        return neighbors;
                }
                public int[] Prim(){ // Só funciona para grafos conexos / Porque o
VERTICE escolhido eh o 0, entao se o 0 nao estiver conexo.../
                        Dictionary<int,int> queue = new Dictionary<int,int> ();
                        List<int> explored = new List<int>();
                        int[] pred = new int[vertices];
                        for (int vertex = 0; vertex < pred.Length; vertex++) {
                                pred [vertex] = -1;
                        }
                        queue.Add (0,0); // Adicionando qualquer véritice. Peso
inicial 0. Dic(vertex,weight) ~~ (key,value)
                        while (queue.Count > 0) {
                                int v = ExtractDictionary (queue);
                                queue.Remove (v);
                                explored.Add (v);
```

```
                        int[] neighbors = ReturnNeighbors (v);
                        for (int u = 0; u < neighbors.Length; u++) {
                                if (neighbors [u] >= 1) {
                                        if (!explored.Contains (u) &&
EdgeWeight (pred [u], u) > EdgeWeight (v, u)) {
                                                if (queue.ContainsKey (u))
                                                        queue [u] = EdgeWeight
(v, u);
                                                else
                                                        queue.Add (u, EdgeWeight
(v, u));
                                                pred [u] = v;
                                        }
                                }
                        }
                /*      Console.WriteLine ("Predecessores:  ");
                        for (int vertex = 0; vertex < pred.Length; vertex++) {
                                Console.WriteLine ("Predecessor de " + vertex.ToString
() + " eh " + pred [vertex]);
                        }*/
                        return pred;
                }
        }
}
```

c) Fuentes de Información.

https://es.wikipedia.org/wiki/%C3%81rbol_(inform%C3%A1tica)#:~:text=Un%20%C3%A1rbol%20es%20una%20estructura,adicionales%20que%20forman%20una%20jerarqu%C3%ADa
https://estructuradedatos192.wordpress.com/arboles-generales/
https://hhmosquera.wordpress.com/arbolesbinarios/
https://sites.google.com/site/estdatinfjiq/unidad-iv-estructuras-no-lineales

https://repositorio.clavijero.edu.mx/repositorio/cpf/006_md/modulo6/contenidos/tema6.2.3.html?opc=1
https://www.iteramos.com/pregunta/5134/cuales-son-las-aplicaciones-de-los-arboles-binarios

https://www.grapheverywhere.com/grafos-que-son-tipos-orden-y-herramientas-de-visualizacion/
https://rootear.com/desarrollo/grafos#:~:text=a%20nuestro%20grafo.-,Tipos%20de%20grafos,que%20une%20dos%20v%C3%A9rtices%20espec%C3%ADficos.&text=Multigrafo.%20o%20pseudografo%20son%20grafos,una%20arista%20entre%20dos%20v%C3%A9rtices.
https://es.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs
https://es.wikipedia.org/wiki/Teor%C3%ADa_de_grafos#:~:text=teor%C3%ADa%20de%20grafos-

,Aplicaciones,o%20en%20%C3%A1reas%20de%20Ingenier%C3%ADa