1-12-2020

Reporte (Algoritmos de búsqueda).

Ingeniería en Sistemas Computacionales.

Inteligencia Artificial

Docente: I.S.C. Jesús Aranda Gamboa.

Adán Ruiz Villalobos

Núm. Control: 16070137

Correo: adnruiz1@gmail.com

Semestre: 9°

Instituto Tecnológico Superior de Jerez

Jerez de García Salinas Zacatecas, Zac.

Planteamiento del programa:

Desarrollar un programa para el recorrido de grafos, visitando cada vertice y nodo exactamente una vez en un orden definido se deben implementar los algoritmos de búsqueda por profundidad y por anchura en grafos. Se debe comenzar por un nodo raíz especificado por el usuario y terminara la búsqueda hasta que haya recorrido todos los nodos y sus respectivas aristas del grafo. Como resultado final debe mostrar el stack obtenido durante la búsqueda, es decir, los nodos recorridos.

Resultado:

Para el resultado de este programa utilice solo una clase y se crearon métodos para realizar los algoritmos de búsqueda. Dentro de esta clase se crearon dos variables una para almacenar los Nodos y la otra es una Lista, en la cual se almacenan los nodos por recorrer.

```
public class Busqueda {
	//Variable que alamcena la cantidad de nodos
	private int Nodos;
	private LinkedList<Integer> listaNodosRec[];
```

Creamos un contructor de nuestra clase con solo un valor entero de parámetro:

```
//cosntructor de la clase:
	@SuppressWarnings("unchecked")
	public Busqueda(int nodos) {
		Nodos = nodos;
		listaNodosRec = new LinkedList[nodos];

		for (int i = 0; i < nodos; i++) {
			listaNodosRec[i] = new LinkedList<>();
		}
	}
```

Se creo un método para agregar las aristas a cada nodo, sus parámetros son el nodo al cual le vamos a ingresar la arista y hacia que donde se dirige nuestra ariste, todo esto en valores enteros.

```
//metodo para agreagar aristas a los nodos
	public void agregarArista(int nodo, int arista) {
		listaNodosRec[nodo].add(arista);
	}
```

Depth First Search

Comencemos con la búsqueda por profundidad (Depth First Search). Para esta búsqueda se creo un método con parámetros para el nodo y un arreglo de booleans para comprobar si los nodos han sido visitados.

```
//mmetodo para el Algoritmo Depth First Search
public void algoritmoDFS(int nodo, boolean visitado[]) {
	//establecemos el nodo actual como visitado y se imprime
	visitado[nodo] = true;
	System.out.println(nodo + " ");

	//creamos un iterator para recorrer y eliminar los elementos
	Iterator<Integer> listaRecorrerAristas =
	listaNodosRec[nodo].listIterator();

	//Recorremos lsita de aristas mientras que tengamos algo dentro de esta.
	while(listaRecorrerAristas.hasNext()) {
		//pasamos el valor del siguiente elemento en el Iterator y se hace
		una comparacion para verificar si ha sido visitado,
		//de lo contrario establecemos el elemento como visitado.
		int i  = listaRecorrerAristas.next();
		if(!visitado[i]) {
			algoritmoDFS(i, visitado);
		}//if
	}//while
}//metodo DFS

//metodo para hacer el recorrido DFS
public void dfs(int nodo) {
	//se establecen todos los nodos cono no visitados.
	boolean visitados[] = new boolean[Nodos];

	algoritmoDFS(nodo, visitados);
}
```

Breadth First Search:

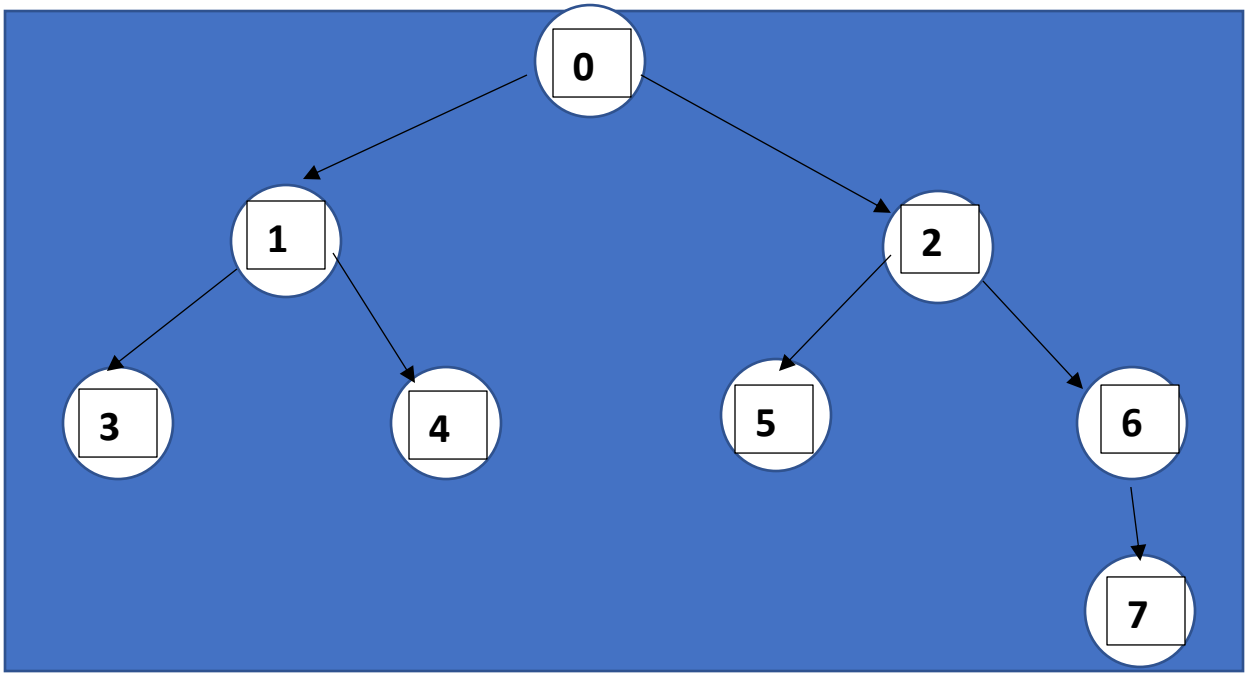
Es un algoritmo de recorrido en que debe empezar a atravesar desde un nodo seleccionado (nodo origen/nodo raíz) y atravesar el grafo por capas.

Para este algoritmo solo recibimos como parámetro el nodo raíz, para verificar que el nodo sea visitado se creo un arreglo de booleans como en el método anterior. En este método se utilizo una lista Queue

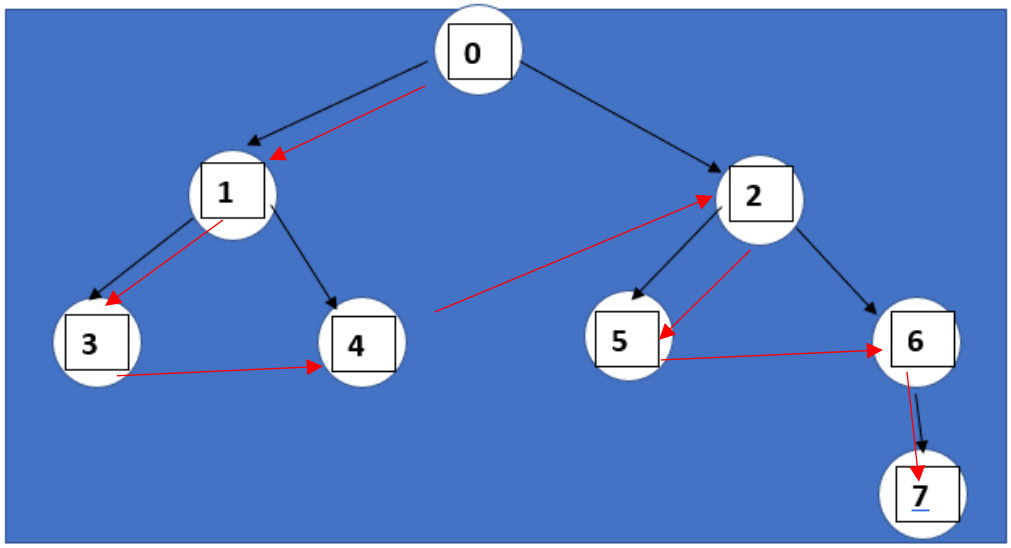
```
public void algoritmoBFS(int nodo) {
		//se establecen todos los nodos cono no visitados.
		boolean visitados[] = new boolean[Nodos];

		//Creamos la lista queue para BFS
		LinkedList<Integer> queue = new LinkedList<>();

		//establecemos el nodo actual como visitado y se agrega a nuestro Queue
		visitados[nodo]= true;
		queue.add(nodo);

		//mientras que el tamaño de nuestra lista queue no se cero
		while(queue.size() != 0) {
			//establecemos el valor del nodo al valor que obtenemos mediante
el metodo poll de nuestra lsita queue, este metodo devuelve y
			//elimina el elemento al frente del contenedor (FIFO)
			nodo = queue.poll();
			System.out.println(nodo + " ");
			//Se obtinen todas las aristas de de los nodos
			Iterator<Integer>  listaRecorrerAristas =
listaNodosRec[nodo].listIterator();

			while(listaRecorrerAristas.hasNext()) {
				int i = listaRecorrerAristas.next();
				if (!visitados[i]) {
					visitados[i]=true;
					queue.add(i);
				}
			}

		}

	}
```

Pruebas:

Para probar mis algoritmos creados utilice un grafo bastante sencillo, con el cual comprobé los algoritmos.



Para la búsqueda a profundidad obtuve los siguientes resultados:



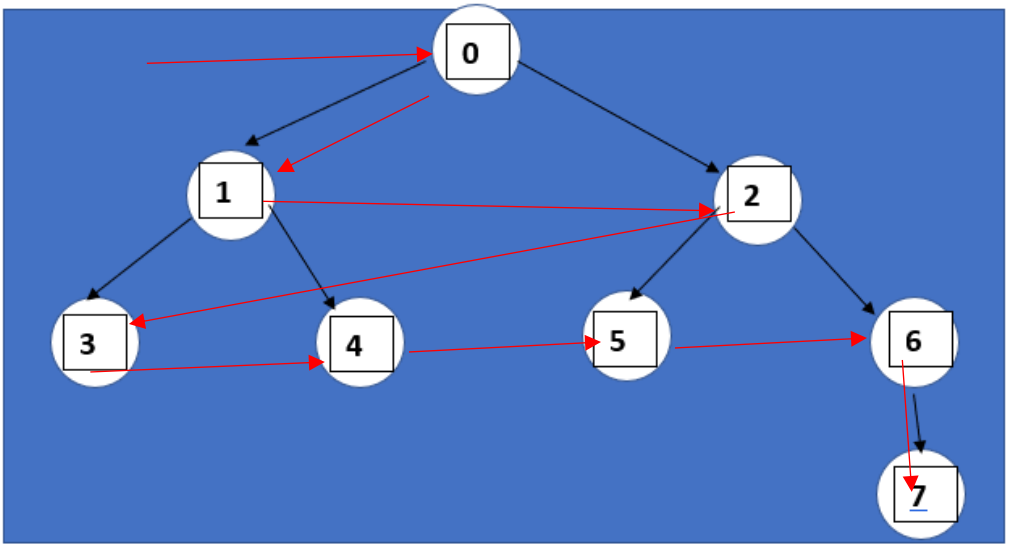
Aquí podemos observar cual es el recorrido que se desea obtener con este algoritmo.

Ingresamos los nodos y sus respectivas aristas, esta inserción de datos será igual para ambos algoritmos.

```
<terminated> Busqueda (1) [Java Application] C:\Program Files\Java\jre1.8.
Ingresa el numero de nodos
8
Ingresa la cantidad de aristas del nodo: 1
2
Ingresa a que nodo se dirige la arista: 1 del nodo 1
1
Ingresa a que nodo se dirige la arista: 2 del nodo 1
2
Ingresa la cantidad de aristas del nodo: 2
2
Ingresa a que nodo se dirige la arista: 1 del nodo 2
3
Ingresa a que nodo se dirige la arista: 2 del nodo 2
4
Ingresa la cantidad de aristas del nodo: 3
2
Ingresa a que nodo se dirige la arista: 1 del nodo 3
5
Ingresa a que nodo se dirige la arista: 2 del nodo 3
6
Ingresa la cantidad de aristas del nodo: 4
0
Ingresa la cantidad de aristas del nodo: 5
0
Ingresa la cantidad de aristas del nodo: 6
0
Ingresa la cantidad de aristas del nodo: 7
1
Ingresa a que nodo se dirige la arista: 1 del nodo 7
7
Ingresa la cantidad de aristas del nodo: 8
0
```

```
Que algoritmo de busqueda utlizara en su grafo?
(Ingrese el numero correspondiente a la opcion)
1.- Depth Firt Search
2.-Breadth First Search
1
Ingrese desde que nodo partira la busqueda:
0
Stack utilizando el algoritmo DFS:
0
1
3
4
2
5
6
7
```

En cambio, para la búsqueda en anchura se busca un recorrido totalmente diferente:



Aquí se puede observar el recorrido que se desea obtener con este algoritmo. Se ingresan exactamente como el algoritmo anterior, la misma cantidad de nodos y aristas. Por lo cual obtenemos de resultado para este algoritmo:

```
Que algoritmo de busqueda utlizara en su grafo?
(Ingrese el numero correspondiente a la opcion)
1.- Depth Firt Search
2.-Breadth First Search
2
Ingrese desde que nodo partira la busqueda:
0
Stack utilizando el algoritmo BFS:
0
1
2
3
4
5
6
7
```

Conclusión:

Como conclusión puedo decir que, si sabemos que la solución o el nodo de búsqueda se encuentra en algún lugar profundo de un grafo o lejos del nodo raíz, podemos utilizar el algoritmo Depth First Search (por profundidad), en cambio si sabemos que la solución no esta retirada del nodo raíz es mas factible utilizar el algoritmo Breadth First Search (por anchura). Si nuestro grafo es muy ancho se recomienda utilizar DFS ya que BFS tomará demasiada memoria en casos extremos. Esto se define ya que la memoria tomada por los algoritmos se define por la estructura de nuestro grafo, la memoria tomada por DFS es igual a la profundidad del grafo y la memoria utilizada por BFS es igual a la anchura del árbol. Como ya se mencionó anteriormente para la implementación de estos algoritmos utilice BFS estructura de datos Queue ya que este algoritmo se considera un LIFO y para el DFS se utilizo la pila de llamadas recursivas ya que esta se considera un FIFO.

Bibliografía

(22 de 05 de 2019). Obtenido de https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/

Guru99. (03 de 11 de 2020). Obtenido de https://www.guru99.com/difference-between-bfs-and-dfs.html

Shaik, M. R. (03 de 07 de 2020). *GeekforGeeks*. Obtenido de https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/