



Facultad de Ingeniería UNAM



Maestro: Tista Garcia Edgar MI.

Asignatura: Estructura de Datos y Algoritmos II

Práctica 6 : Algoritmos de grafos Parte 1

Alumno: Mejía Alba Israel Hipólito

Grupo:8

Mejía Alba Israel Hipólito G8 P6 V1.1

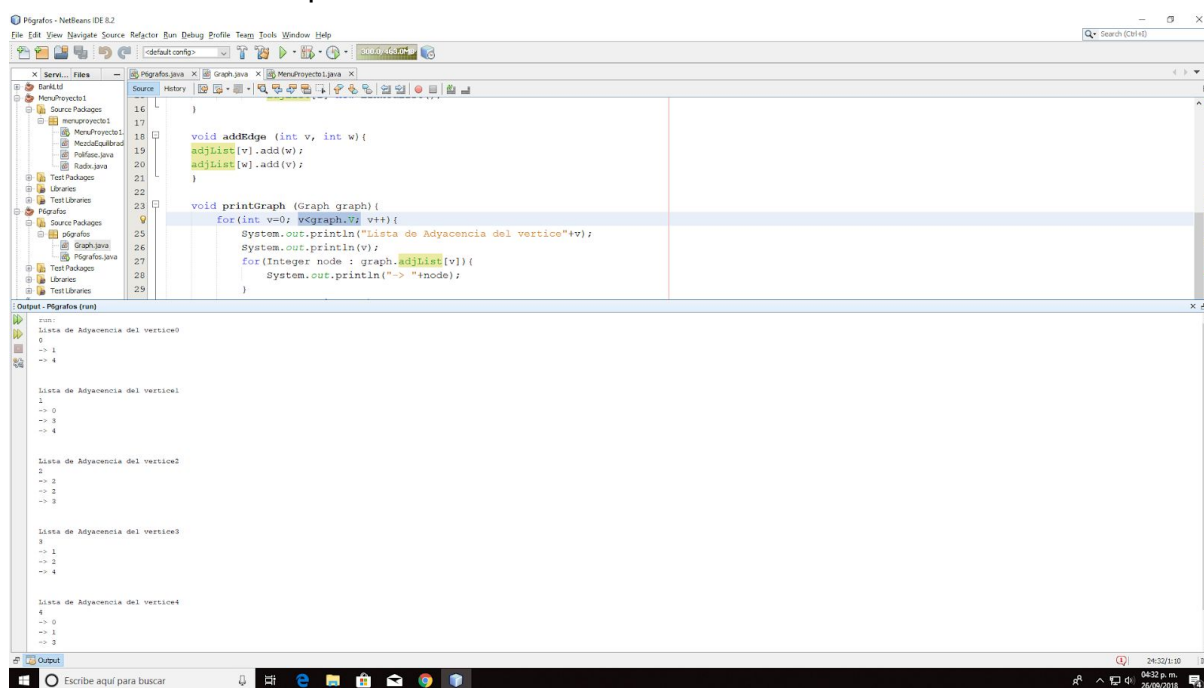
Objetivo:

El estudiante conocerá las formas de representar un grafo e identificará las características necesarias para componer el algoritmo de búsqueda por expansión.

Desarrollo:

a) Compila y ejecuta el código, realiza un análisis de la implementación de grafos y menciona las características que te parezcan relevantes del mismo:

Me llama la atención como en el método principal se inicializó V en 5 y este valor se manda al constructor del objeto Graph, el cual se le asigna a su V privada, la cual se usa su valor dentro de los métodos de Graph, como por ejemplo, en el ciclo del método printGraph se usa como tope del ciclo, definiéndose que se mande a imprimir los resultados 5 veces a pantalla.



The screenshot shows the NetBeans IDE with a project named 'Pígrafos'. The source code for 'Pígrafos.java' is displayed, featuring two methods: 'addEdge' and 'printGraph'. The 'printGraph' method uses a loop from 0 to 4 (inclusive) to print the adjacency list for each vertex. The output window shows the results of this execution, displaying the adjacency lists for vertices 0 through 4.

```
16  
17  
18 void addEdge (int v, int w){  
19     adjList[v].add(w);  
20     adjList[w].add(v);  
21 }  
22  
23 void printGraph (Graph graph){  
24     for(int v=0; v<graph.V; v++){  
25         System.out.println("Lista de Adyacencia del vertice"+v);  
26         System.out.println(v);  
27         for(Integer node : graph.adjList[v]){  
28             System.out.println("-> "+node);  
29         }  
30     }  
31 }
```

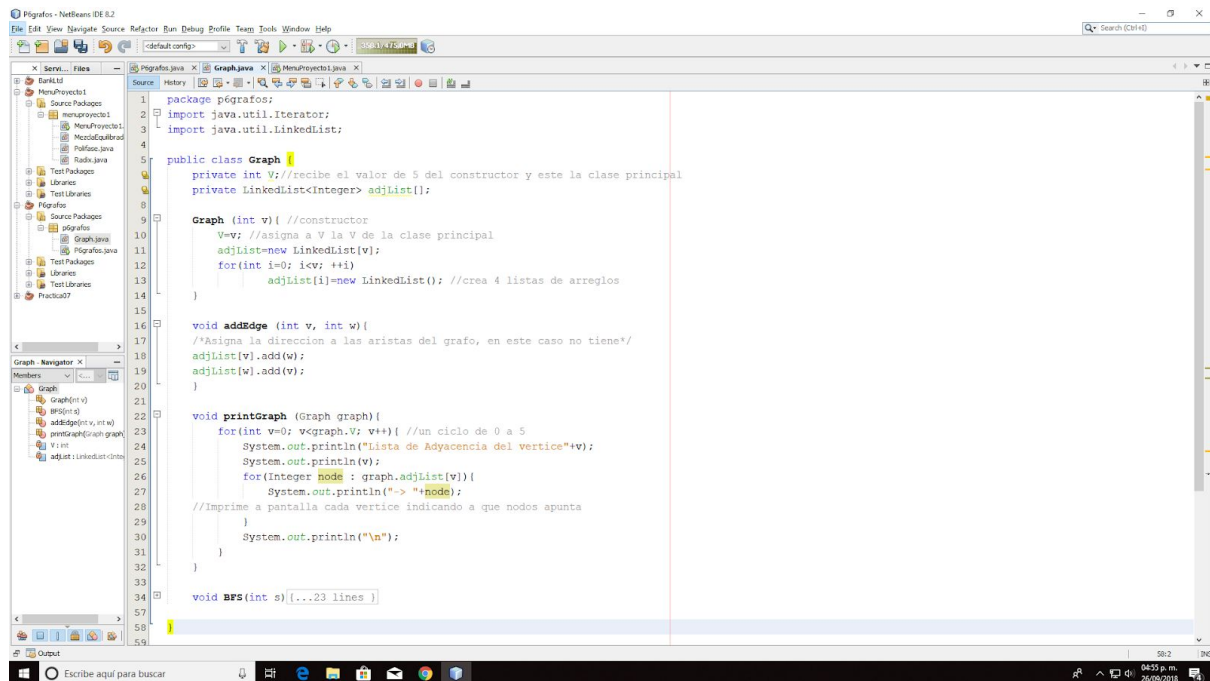
Output - Pígrafos (run)

```
Lista de Adyacencia del vertice0  
0  
-> 1  
-> 4  
  
Lista de Adyacencia del vertice1  
1  
-> 0  
-> 3  
-> 4  
  
Lista de Adyacencia del vertice2  
2  
-> 2  
-> 3  
-> 3  
  
Lista de Adyacencia del vertice3  
3  
-> 1  
-> 2  
-> 4  
  
Lista de Adyacencia del vertice4  
4  
-> 0  
-> 1  
-> 3
```

b) Explica y modifica la clase Graph para que ahora se trabaje con grafos dirigidos:

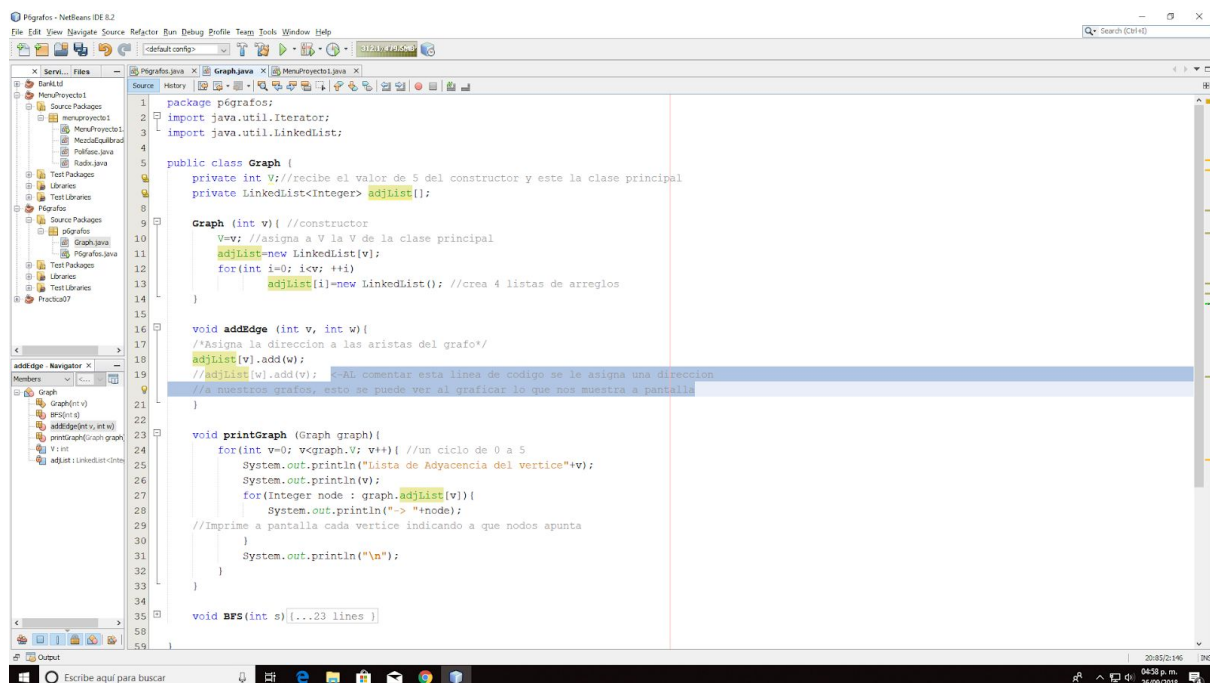
-Explicación:

A continuación agrego capturas de pantalla donde explico de manera breve cada parte de la clase Graph:



```
1 package p6grafos;
2 import java.util.Iterator;
3 import java.util.LinkedList;
4
5 public class Graph {
6     private int V; //recibe el valor de 5 del constructor y este la clase principal
7     private LinkedList<Integer> adjList[];
8
9     Graph (int v) { //constructor
10         V=v; //asigna a V la V de la clase principal
11         adjList=new LinkedList<Integer>[v];
12         for(int i=0; i<v; ++i)
13             adjList[i]=new LinkedList<Integer>(); //crea 4 listas de arreglos
14     }
15
16     void addEdge (int v, int w){
17         //Asigna la direccion a las aristas del grafo, en este caso no tiene*/
18         adjList[v].add(w);
19         adjList[w].add(v);
20     }
21
22     void printGraph (Graph graph){
23         for(int v=0; v<graph.V; v++){ //un ciclo de 0 a 5
24             System.out.println("Lista de Adyacencia del vertice"+v);
25             System.out.println(v);
26             for(Integer node : graph.adjList[v]){
27                 System.out.println("-> "+node);
28             }
29             //Imprime a pantalla cada vertice indicando a que nodos apunta
30             System.out.println("\n");
31         }
32     }
33
34     void BFS(int s){...23 lines }
35
36 }
```

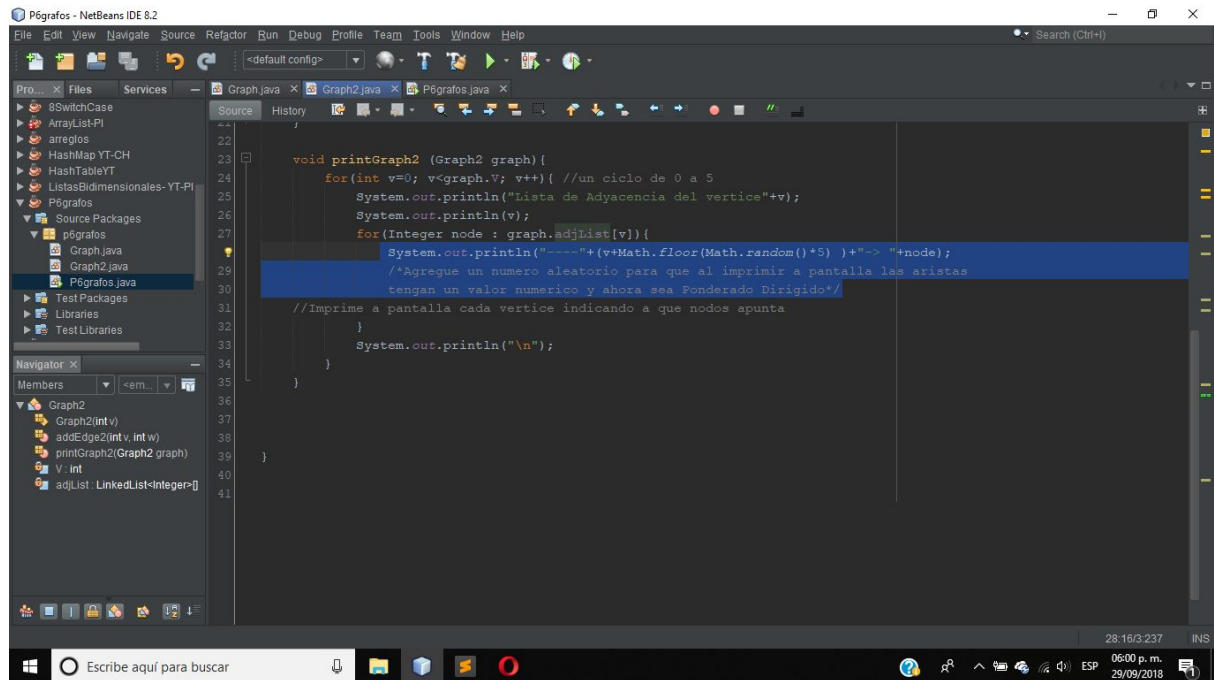
-Modificación:



```
1 package p6grafos;
2 import java.util.Iterator;
3 import java.util.LinkedList;
4
5 public class Graph {
6     private int V; //recibe el valor de 5 del constructor y este la clase principal
7     private LinkedList<Integer> adjList[];
8
9     Graph (int v) { //constructor
10         V=v; //asigna a V la V de la clase principal
11         adjList=new LinkedList<Integer>[v];
12         for(int i=0; i<v; ++i)
13             adjList[i]=new LinkedList<Integer>(); //crea 4 listas de arreglos
14     }
15
16     void addEdge (int v, int w){
17         //Asigna la direccion a las aristas del grafo*/
18         adjList[v].add(w);
19         //Al comentar esta linea de codigo se le asigna una direccion
20         //a las aristas del grafo, esto se puede ver al graficar lo que nos muestra a pantalla
21         adjList[w].add(v);
22     }
23
24     void printGraph (Graph graph){
25         for(int v=0; v<graph.V; v++){ //un ciclo de 0 a 5
26             System.out.println("Lista de Adyacencia del vertice"+v);
27             System.out.println(v);
28             for(Integer node : graph.adjList[v]){
29                 System.out.println("-> "+node);
30             }
31             //Imprime a pantalla cada vertice indicando a que nodos apunta
32             System.out.println("\n");
33         }
34     }
35
36     void BFS(int s){...23 lines }
37
38 }
```

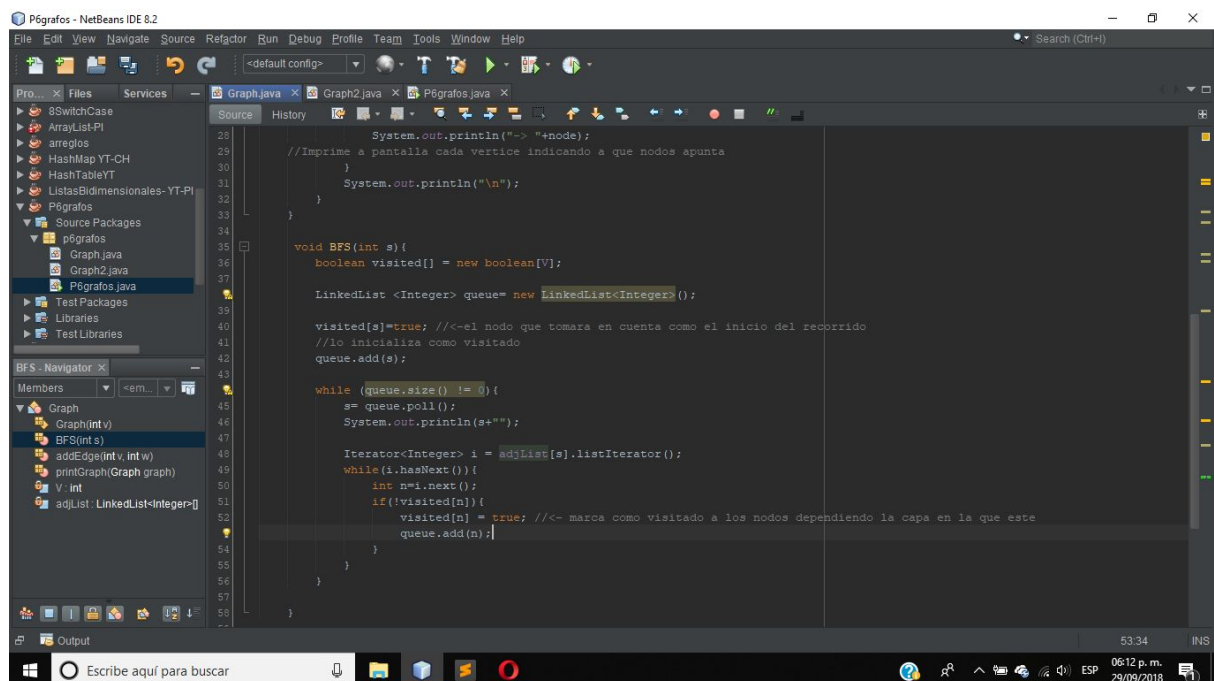
Al comentar la 19 linea de codigo es como si ya no existiera en nuestra clase y ahora al graficar lo que nos muestra a pantalla podemos decir que ahora tiene una dirección para nuestras aristas. Esto a diferencia como estaba antes cuando tenía una “Doble” dirección, por lo que se podría decir que no tenía.

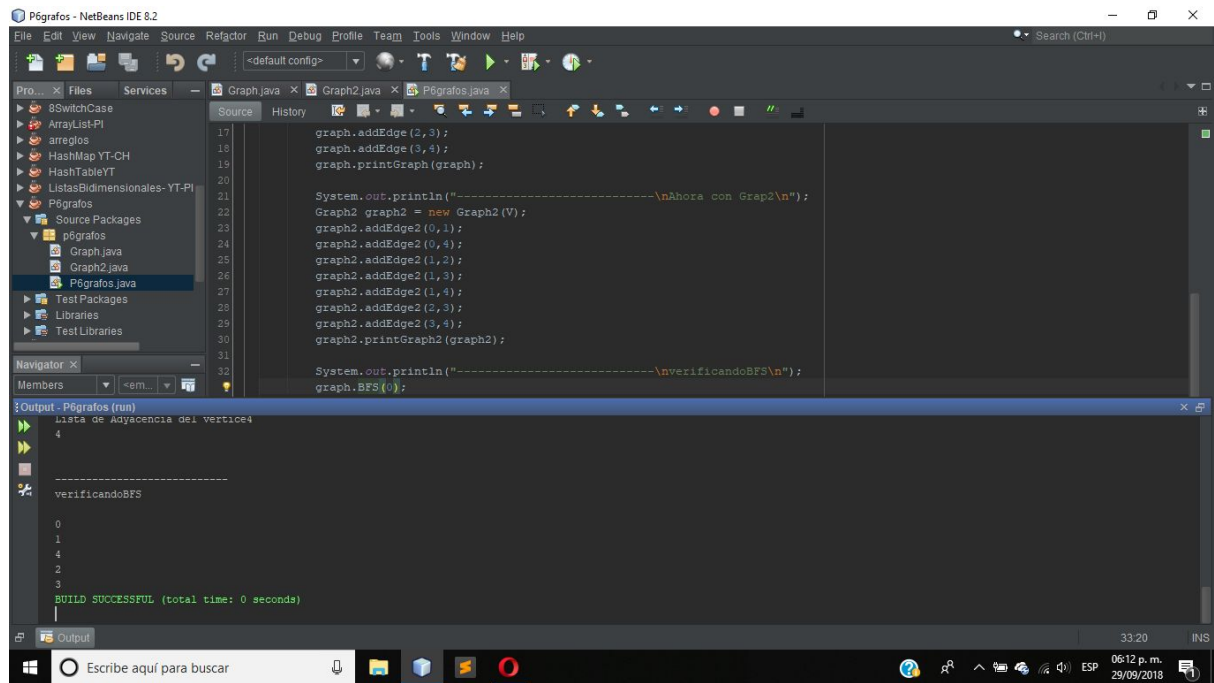
c) Agrega una clase al proyecto (Graph2) que sea una copia de la clase Graph. Modifica Graph2 de tal manera que ahora trabajé con grafos ponderados:



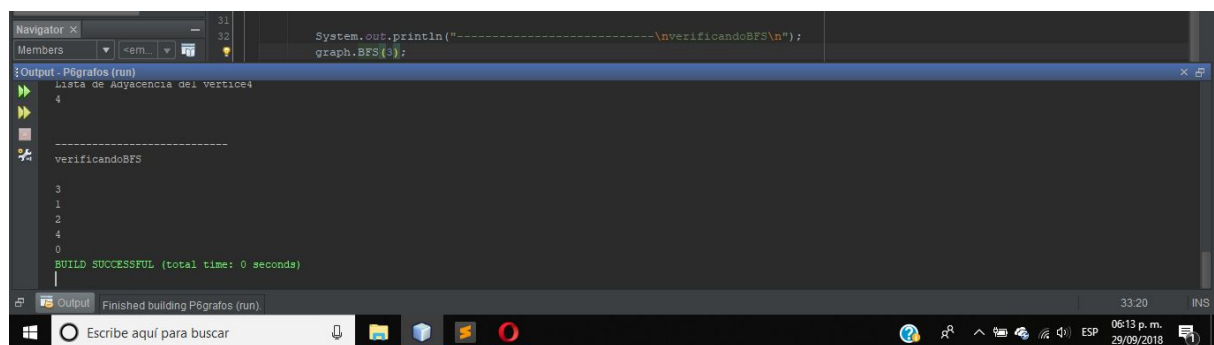
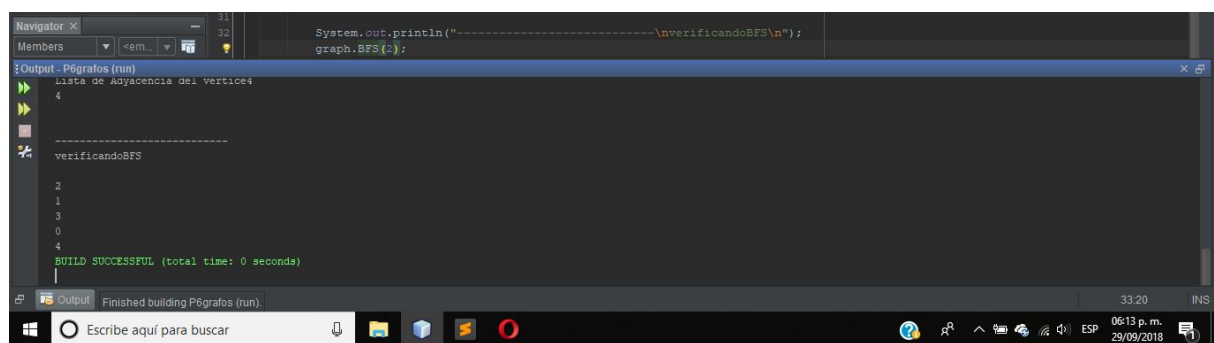
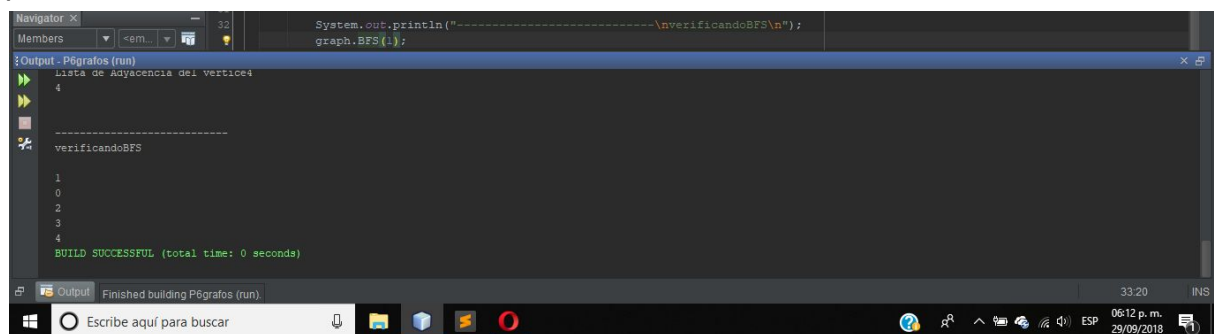
Ejercicio 2 : BFS:

Verificar el funcionamiento del método BFS:





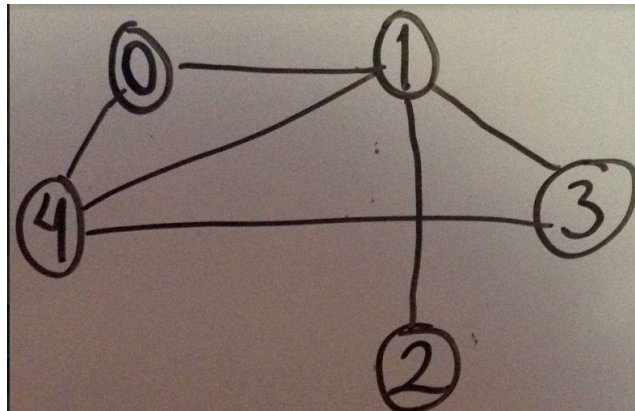
probando con otros vertices:



```
Navigator X
Members
System.out.println("-----\nverificandoBFS\n");
graph.BFS(4);

Output - P6grafos (run)
Lista de adyacencia del vertice 4
4
0
1
3
2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Como se puede ver en pantalla el código de BFS realiza el recorrido por capas, tomando en cuenta el orden numérico de los números, esto se puede ver mejor en el grafo que anexo como ilustración:



Conclusiones:

Esta práctica reforzó mis conocimientos vistos en clase en grafos ya sea del tipo dirigido o ponderado, así como su recorrido de los mismos, en este caso usando BFS para un grafo simple de 5 elementos, me llamó la atención como seguía los términos para este recorrido visto en clase que se resume de ir de capa en capa tomando en cuenta el valor numérico de cada vértice.