



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

M.I Tista Garcia Edgar

Profesor:

Estructura de Datos y Algoritmos II

Asignatura:

5

Grupo

6

:No

Calzada Martinez Jonathan Omar

dePráctica(s):Inte

grante(s):

No. de Equipo de cómputo empleado

36

2019-2

Semestre:

12/03/2019

Fecha de entrega:

Observaciones:

CALIFICACIÓN: _____

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

Actividades

Ejercicio 1. Existen diversas implementaciones de Grafos que son adaptadas al lenguaje de programación correspondiente, así como a la forma de representación que se utilizan.

Ejercicio 2. BFS (Breadth First Search)

Introducción :

Un Grafo no es más que un conjunto de nodos o vértices que se encuentran relacionados con unas aristas. Además los vértices tienen un valor y en ocasiones las aristas también y se le conoce como el costo.

Grafos en Java

Desde un punto de vista intuitivo un grafo es un conjunto de nodos unidos por un conjunto de arcos. Un ejemplo de grafo que podemos encontrar en la vida real es el de un plano de trenes. El plano de trenes está compuesto por varias estaciones (nodos) y los recorridos entre las estaciones (arcos) constituyen las líneas del trazado.

Desarrollo:

Ejercicio 1)

Se realizó la codificación de las clases del proyecto principal agregando la clase Graph1 y el método en la clase Graph1.

practica6calzadajonathan
Graph1.java
Practica6CalzadaJonathan.java

est Packages
braries
est Libraries

ator x

<empty>

ica6CalzadaJonathan
main(String[] args)

```
2  * To change this license header, choose License
3  * To change this template file, choose Tools | 1
4  * and open the template in the editor.
5  */
6  package practica6calzadajonathan;
7
8  /**
9   *
10  * @author edaII05alu09
11  */
12  public class Practica6CalzadaJonathan {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          int V;
19          // TODO code application logic here
20          V = 5;
21
22
23          Graph1 graph = new Graph1 (V);
24          graph.addEdge(0,1);
25          graph.addEdge(0,4);
26          graph.addEdge(1,2);
27          graph.addEdge(1,3);
28          graph.addEdge(1,4);
29          graph.addEdge(2,3);
30          graph.addEdge(3,4);
31          graph.printGraph (graph);f
32      }
33
34  }
35
```

En estas lineas en la clase graph se llama al metodo addEdge y se le agrega el nodo y con cual va a estar conectado

```
graph.addEdge(3,4);  
graph.addEdge(2,4);  
// ...
```

En la funcion Graph se crea la lista adjArray, y se crea un bucle para crear en cada nodo una lista.

En el metodo addEdge se utiliza para poder recibir del main el nodo y el nodo al que va estar conectado o unido.

```
Graph1(int v){  
    V=v;  
    adjArray = new LinkedList[v];  
    for(int i=0; i<v;++i)  
        adjArray[i] = new LinkedList();  
  
}  
  
void addEdge (int v,int w){  
    adjArray[v].add(w);  
    adjArray[w].add(v);  
}
```

Se crea un metodo para la impresion de la lista de adyacencia (grafo)

b) IMPLEMENTAR QUE FUNCIONE CON GRAFOS DIRIGIDOS

```
V=5;  
Graph1 graphDir = new Graph1(V);  
graphDir.addDir(1,3);  
graphDir.addDir(1,4);  
graphDir.addDir(2,1);  
graphDir.addDir(4,1);  
graphDir.addDir(4,5);
```

Como se ve en la imagen anterior primero se mando el nodo de origen y el nodo de destino.

Para el nodo 1 se dirigio hacia el 3 y el 4

Para el nodo 2 se dirigio hacia el nodo 1

para el nodo 4 se dirigio hacia el nodo 1 y el 5

Para esto solo se creo un metodo que

```
void addDir (int s, int r) {  
    adjArray[s].add(r);  
    / adjArray[s].add(r);  
}
```

```
Grafica Dirvista en clase  
Lista de adyacencia del vertice 0  
0
```

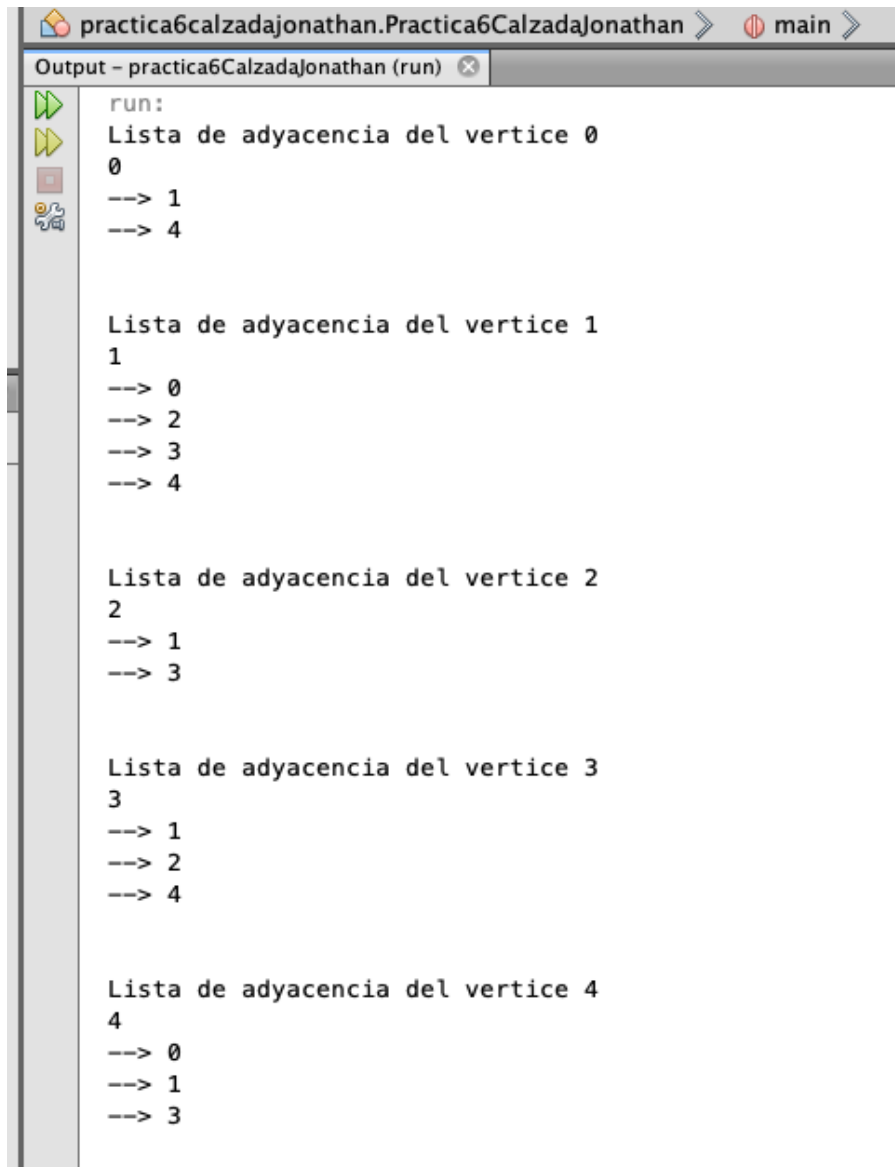
```
Lista de adyacencia del vertice 1  
1  
--> 3  
--> 4
```

```
Lista de adyacencia del vertice 2  
2  
--> 1
```

```
Lista de adyacencia del vertice 3  
3
```

```
Lista de adyacencia del vertice 4  
4  
--> 1  
--> 5
```

-----El código compilado es el siguiente.



```
practica6calzadajonathan.Practica6CalzadaJonathan > main >  
Output - practica6CalzadaJonathan (run) x  
run:  
Lista de adyacencia del vertice 0  
0  
--> 1  
--> 4  
  
Lista de adyacencia del vertice 1  
1  
--> 0  
--> 2  
--> 3  
--> 4  
  
Lista de adyacencia del vertice 2  
2  
--> 1  
--> 3  
  
Lista de adyacencia del vertice 3  
3  
--> 1  
--> 2  
--> 4  
  
Lista de adyacencia del vertice 4  
4  
--> 0  
--> 1  
--> 3
```

Ejercicio 2

```
42
43 void BFS (int s){
44     boolean visited[] = new boolean[V];
45     LinkedList<Integer> queue = new LinkedList<>();
46
47     visited[s] = true;
48     queue.add(s);
49     while (!queue.isEmpty()){
50         s = queue.poll();
51         System.out.print(s + " ");
52
53         Iterator<Integer> i = adjArray[s].listIterator();
54         while (i.hasNext()){
55             int n = i.next();
56             if (!visited[n]){
57                 visited[n] = true;
58                 queue.add(n);
59             }
60         }
61     }
62 }
63 }
64 }
```

En la función principal se agregaron las instrucciones necesarias para que se ejecute la función:

```
System.out.println("\n BFS desde vert
graph.BFS(1);
System.out.println("\n BFS desde verti
graph.BFS(3);
System.out.println("\n BFS desde vert
graph.BFS(4);
System.out.println("\n BFS desde vert
graph.BFS(5);
```

EL grafo va haciendo el recorrido a partir de el nodo dado.

```
BFS desde vertice 1 ejemplo)
1 2 3 4 0
BFS desde vertice 3 ejemplo)
3 1 2 4 0
BFS desde vertice 4 ejemplo)
4 0 1 2 3
```

El algoritmo de recorrido BFS trabajadapor bloques, realizando bloque por bloque hasta acabar con el grafo

El algoritmo es igual al visto en clase solo cambien pequeñas cosas como por ejemplo el return visitados.

ejercicio 3. Grafos ponderados

Se realizo la clase con el nombre graph2 y se intento realizar el algoritmo de grafos ponderados, no se logro el objetivo, en la imagen de abajo se ve que se intento mandar 3 datos al llmar al metodo agregar que son los dos vertices y el peso de la arista.

```
    }
    void addDir (int s, int r, int peso){
        adjArray[s].add(r);
        adjArray[r].add(s);
        adjArray[s].add(peso);
    //    adjArray[s].add(r);
    }
```


Conclusiones:

Los grafos son una gran ayuda en la programación ya que nos permiten realizar búsquedas en algún mapa, por ejemplo las líneas del metro. Si queremos llegar a un lugar en donde nos ayude un grafo nos ayudaría a encontrar la estación más cercana.

El objetivo de la práctica no se cumplió del todo ya que el tercer ejercicio no se pudo concluir por una confusión que tuve al hacer el cambio del algoritmo ponderado, en esa parte me atore bastante y ya no pude completarla.