

**UNIVERSIDAD SANTO TOMAS
DUAD
FACULTAD DE CYT
PROGRAMA INGENIERIA EN INFORMATICA**

**BUENAS PRÁCTICAS EN ALGORITMOS DE TEORIA DE GRAFOS CON NETBEANS
REGISTRO DNDA No. 10-727-56
FECHA DE REGISTRO: 09-JUL-2018**

**MARIO DUSTANO CONTRERAS CASTRO
DOCENTE TIEMPO COMPLETO**

ACTUALIZACION 2019

Bogotá, Octubre 2019

**UNIVERSIDAD SANTO TOMAS
DUAD
FACULTAD DE CYT
PROGRAMA INGENIERIA EN INFORMATICA**

PRACTICAS CON LISTAS TDA

**TOMADO DE: BUENAS PRÁCTICAS EN ESTRUCTURAS DE DATOS CON
NETBEANS 8
DNDA No. 10-654-228**

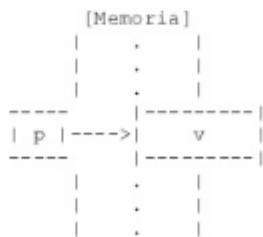
**MARIO DUSTANO CONTRERAS CASTRO
DOCENTE TIEMPO COMPLETO**

ACTUALIZACION Abril 2020

PUNTEROS O APUNTADORES

Los punteros son de amplia utilización en programación, muchos lenguajes permiten la manipulación directa o indirecta de los mismos. La razón de ser principal de los punteros reside en manejar datos alojados en la zona de memoria **dinámica** o **heap** (aunque también se pueden manipular objetos en la zona estática), bien sean datos elementales, objetos pertenecientes a una **clase** (en lenguajes Orientados a Objetos).

Un **puntero** o **apuntador** es una variable que referencia una región de memoria; en otras palabras es una variable cuyo valor es una dirección de memoria. Si se tiene una variable '**p**' de tipo puntero que contiene una dirección de memoria en la que se encuentra almacenado un valor '**v**' se dice que '**p**' *apunta* a '**v**'. El programador utilizará punteros para guardar datos en memoria en muchas ocasiones, de la forma que se describe a continuación.



Trabajar con punteros implica la no manipulación de los datos en sí, sino manejar las **direcciones** de memoria en la cuales estos residen.

NODO

En programación concretamente en estructura de datos, un **nodo** es uno de los elementos de una **lista enlazada**, de un **árbol** o de un **grafo**. Cada nodo será una estructura o **registro** que dispondrá de varios **campos**, y al menos uno de esos campos será un **puntero** o referencia a otro nodo, de forma que, conocido un nodo, a partir de esa referencia, será posible en teoría tener acceso a otros nodos de la estructura. Los nodos son herramientas esenciales para la construcción de estructuras de datos dinámicas.



DEFINICION DE CLASE CONFORMANDO NODO

```
class NodosLista // se define la clase Nodo
{
    Object datos; // Campo Información
    NodosLista siguiente; //Campo Nodo
        // datos: que almacena la información
        // siguiente : Apuntador o enlace a otros nodos
```



```
NodosLista(Object valor) // Se define un nodo
```

```
{  
    datos=valor;  
    siguiente=null;
```

Como la lista es una consecución de muchos nodos es necesario establecer nombre a los nodos y colocarlos a apuntar a algún sitio, en el caso del único nodo debe apuntar a NULL. En este caso se crea un nodo llamado P, indicando que es el primero de la lista.

P =new NodosLista;



NODO P

Para acceder al nodo y escribir valores en sus campos es necesario identificar al Nodo, que en este caso es P y colocar un punto para poder acceder a los campos del nodo, como se muestra a continuación.

```
P.dato= 25;
```

```
P.siguiente = Null;
```

La representación grafica del Nodo queda de la siguiente forma:



NODO P

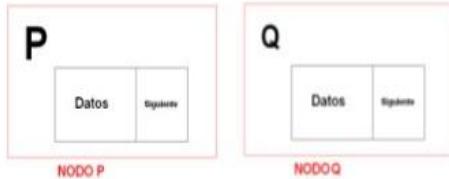
CAMPO APUNTADOR DE UN NODO

P ESTE ES EL NODO COMPLETO

Esto quiere decir que un nodo debe tener alguna dirección de memoria asignada ¿Cuál? no sabemos, pero se puede saber si se hace referencia a P, cuando se hace referencia a P, se indica todo el nodo tanto el campo info como el campo apuntador.

Si se define un nodo, el nodo creado contiene el campo información y el campo siguiente

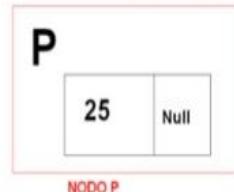
Nodo P y Nodo Q



Se puede hacer referencia o apuntar un puntero con el campo siguiente



`P.siguiente= Q;`



Cuando se coloca el nombre del nodo haciendo referencia a otro realmente se apunta al nodo indicado

`P.siguiente = Null;`



Tomado de: <https://es.slideshare.net/BorisSalleg/conceptos-de-punteros-y-nodos-10104804>

Punteros o Apuntadores en Java

Java no tiene un tipo de dato apuntador, es decir, por lo tanto no se puede declarar una variable de tipo apuntador.

Sin embargo, las variables que se declaran de un tipo que es una clase, en nuestro nuevo vocabulario se llaman instancias de una clase u objetos, son realmente apuntadores al objeto. Por eso, se deben inicializar con un new.

Ejemplo

Una de las operaciones que usted conoce es anadir un nodo al comienzo de una lista ligada. Suponga que ptr apunta al node de la lista despues del cual se añadirá un nuevo nodo. La clase se definió así:

```
class Nodo {  
    int m,n;  
    Nodo sig; // apuntador al siguiente nodo  
}
```

Las variables cab, p ,q ,r se definen como apuntadores, así:

```
Nodo cab,p,q,r;
```

En el siguiente código se crea un nodo con los respectivos punteros para una lista:

```
r = new Nodo();
```

```
...
```

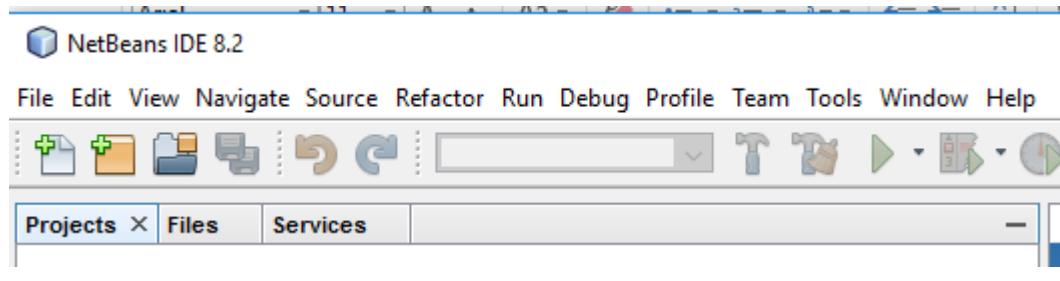
```
r.sig=p.sig;
```

```
p.sig=r;
```

PRIMER MOMENTO. INGRESO A NETBEANS

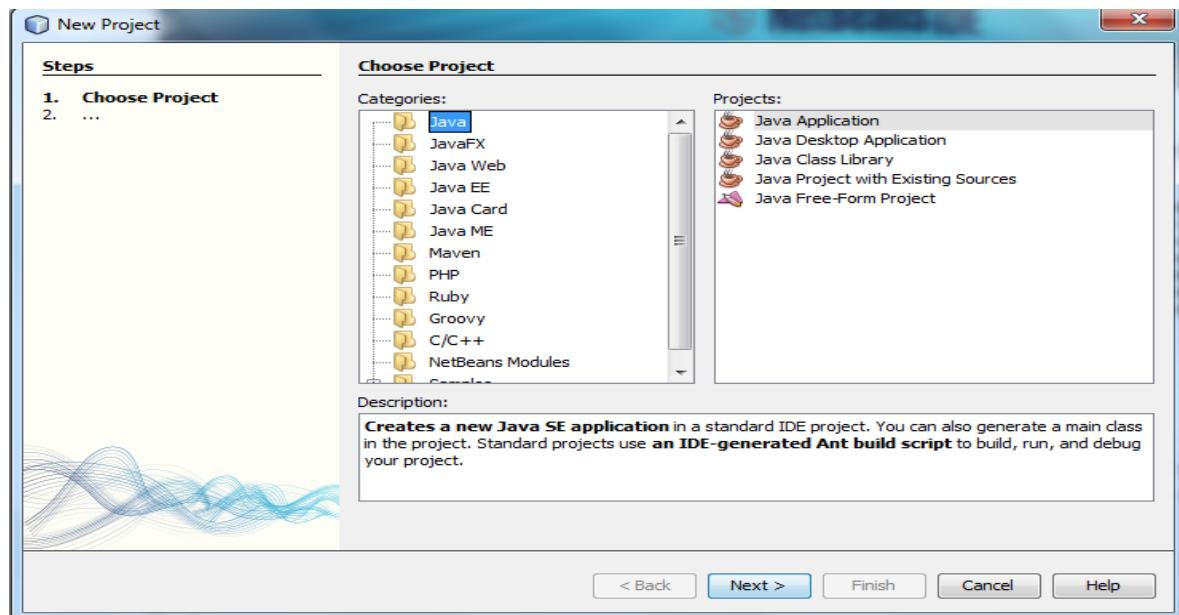
Menú Inicio>>Todas las Aplicaciones>>Netbeans>>Netbeans 8.xx

Ventana de Netbeans



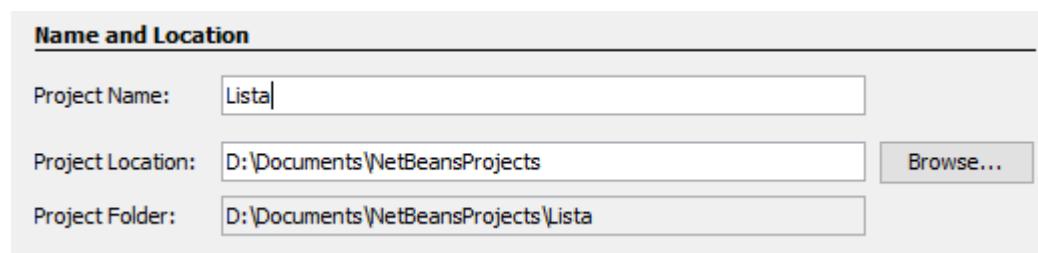
Se va a crear un nuevo proyecto. Seleccionar y picar sobre el icono .

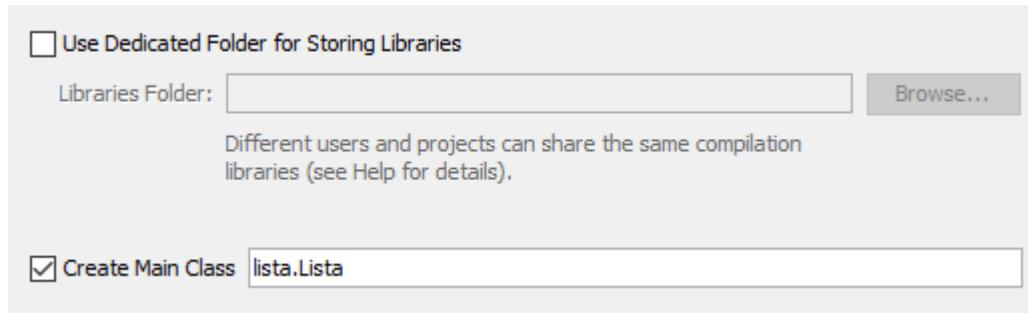
Se debe seleccionar Java de la ventana de categorías



Ahora, se selecciona Java Application de tipo de proyecto. Pulsar el Botón Next

Nombre del Proyecto: Lista

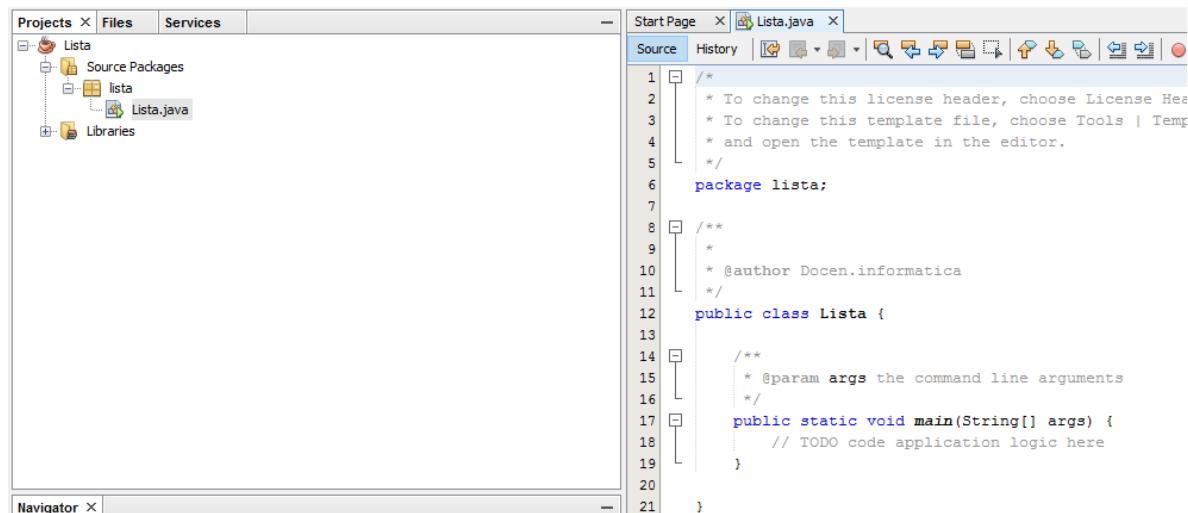




Pulsar el Botón Finish

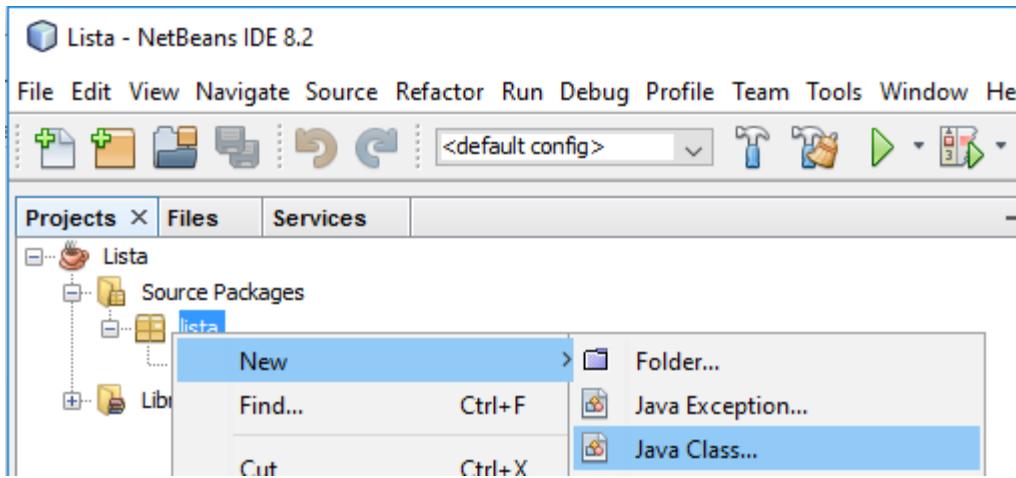
El Netbeans genera la estructura de un proyecto con:

- Paquete de Programas Fuentes(Source Packages):: lista
- Programa Principal:: Lista.java



CREACION DE LA CLASE NODO

Se debe posicionar sobre el paquete lista y pulsar el botón derecho del mouse>>opción new>>Java Class



La clase de negocio sobre la que se va a trabajar se denomina cuenta, por tanto:

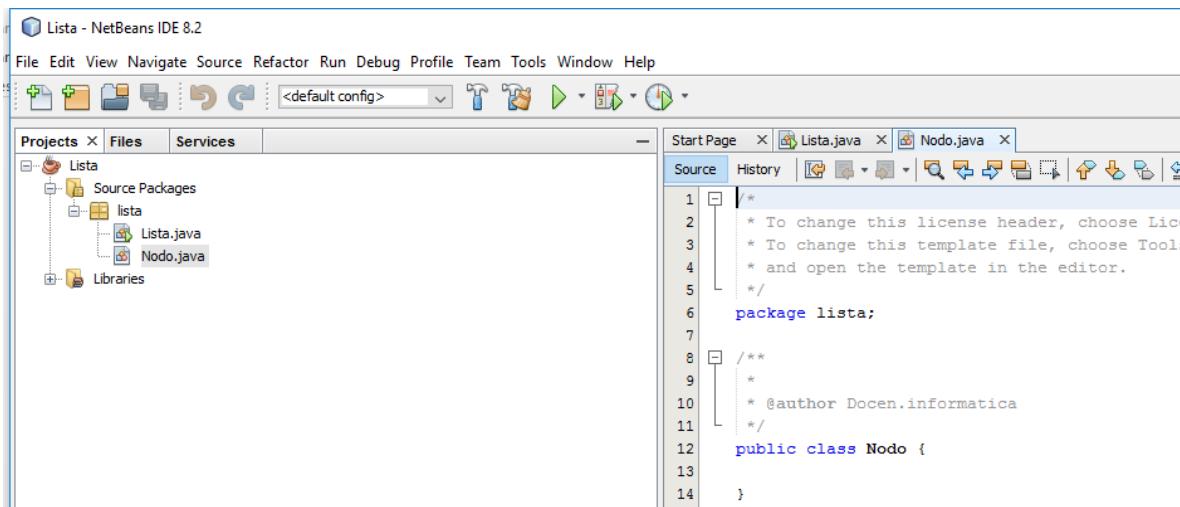
Name and Location

Class Name:	Nodo
Project:	Lista
Location:	Source Packages
Package:	lista
Created File:	D:\Documents\NetBeansProjects\Lista\src\lista\Nodo.java

Nombre de la Clase: Nodo

Pulsar el Botón Finish

El Netbeans genera:



```
1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package lista;
7
8  /**
9  *
10 * @author Docen.informatica
11 */
12 public class Nodo {
13
14 }
```

La definición de la variables se deben colocar después de

public class Nodo { quedando:

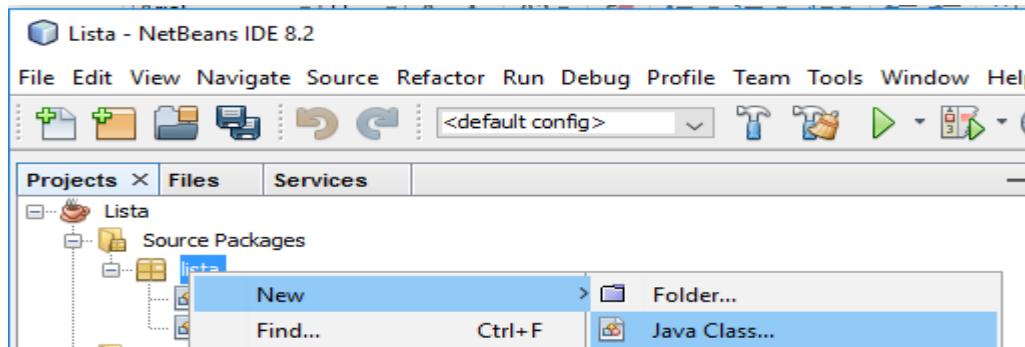
```
public class Nodo {
    int info;
    Nodo(int info) {
        this.info = info;
    }
    Nodo sig;
}

6     package lista;
7
8     /**
9      *
10     * @author funcionario
11     */
12    public class Nodo {
13        int info;
14        Nodo(int info) {
15            this.info = info;
16        }
17        Nodo sig;
18    }
19}
```

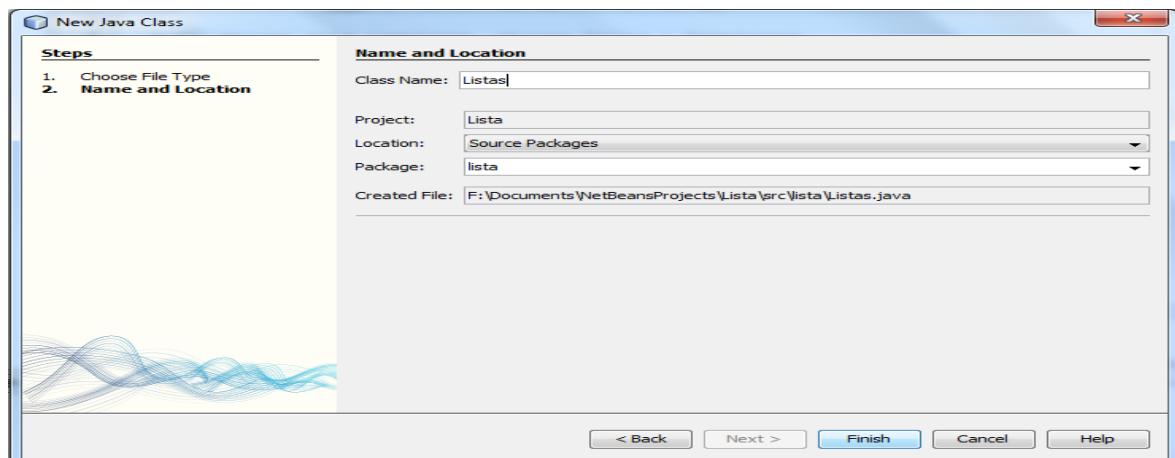
ADMINISTRADOR DE LA CLASE NODO– APLICACIÓN DEL TDA LISTA-

Se diseña una clase que administre por medio de un TDA lista la clase nodo (cada nodo es un elemento de la lista) entonces:

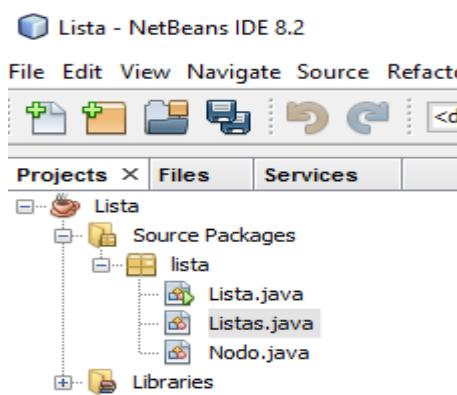
Se debe posicionar sobre el paquete lista y pulsar el botón derecho del mouse>>opción new>>Java Class



El nombre de la clase es Listas



Pulsar el botón Finish



Colocar dentro de public class Listas lo siguiente:

```
Nodo p,q,r;
Nodo cab=null;
boolean buscaInfo (int x)
{
    p=q=cab;
    while(p!=null && p.info!=x)
    {
        q=p;
        p=p.sig;
    }
    if(p!=null)
        return true;
    else
        return false;
}
void recorreLista()
{
    p=q=cab;
    while(p!=null)
    {
        q=p;
        p=p.sig;
    }
}
void adicion(int x)
{
    if(!buscaInfo (x))
    {
        r=new Nodo(x);
        if(cab==null)
            cab=r;
        else
            q.sig=r;
        r.sig=null;
    }
}
void borrar(int x)
{
    if(buscaInfo(x))
    {
        if(p==cab)
            cab=cab.sig;
        else
            q.sig=p.sig;
    }
}
```

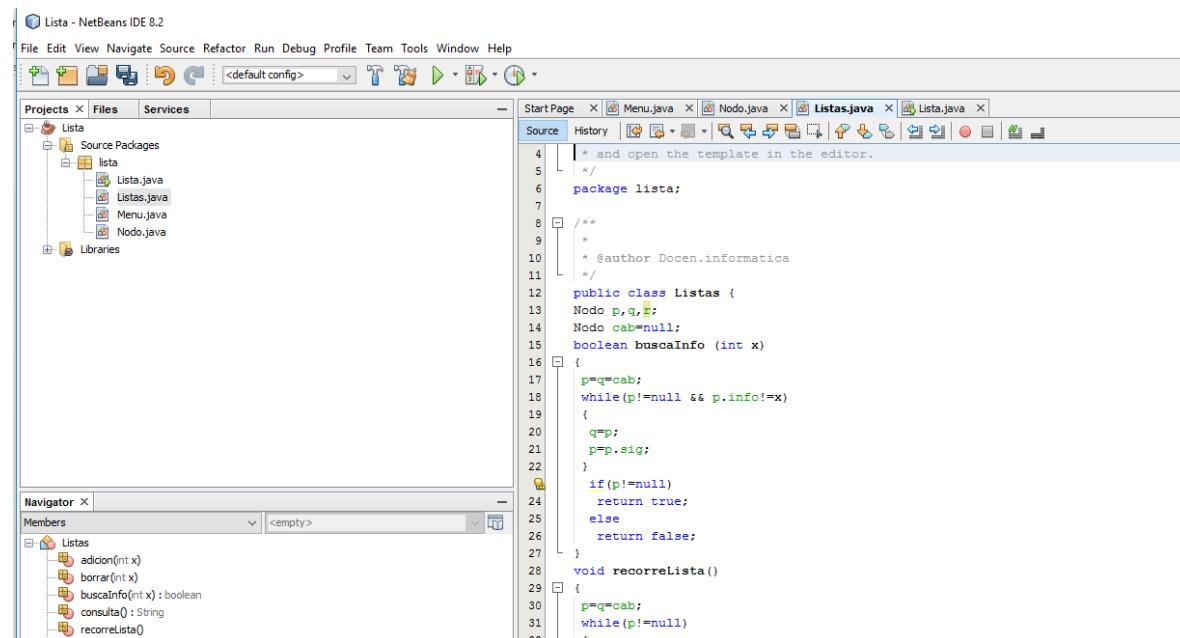
```

String consulta()
{
    String S="C O N S U L T A \n";
    p=cab;
    while(p!=null)
    {

        S+=p.info+"\n";
        p=p.sig;
    }
    return S.toString();
}

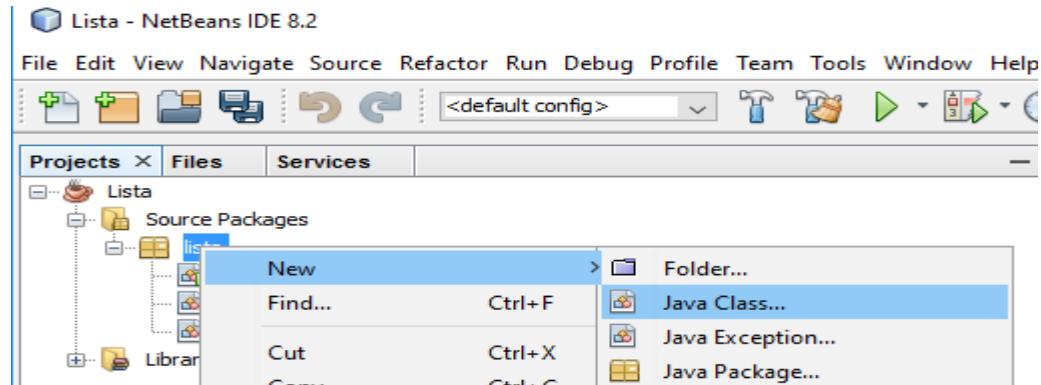
```

Quedando así:

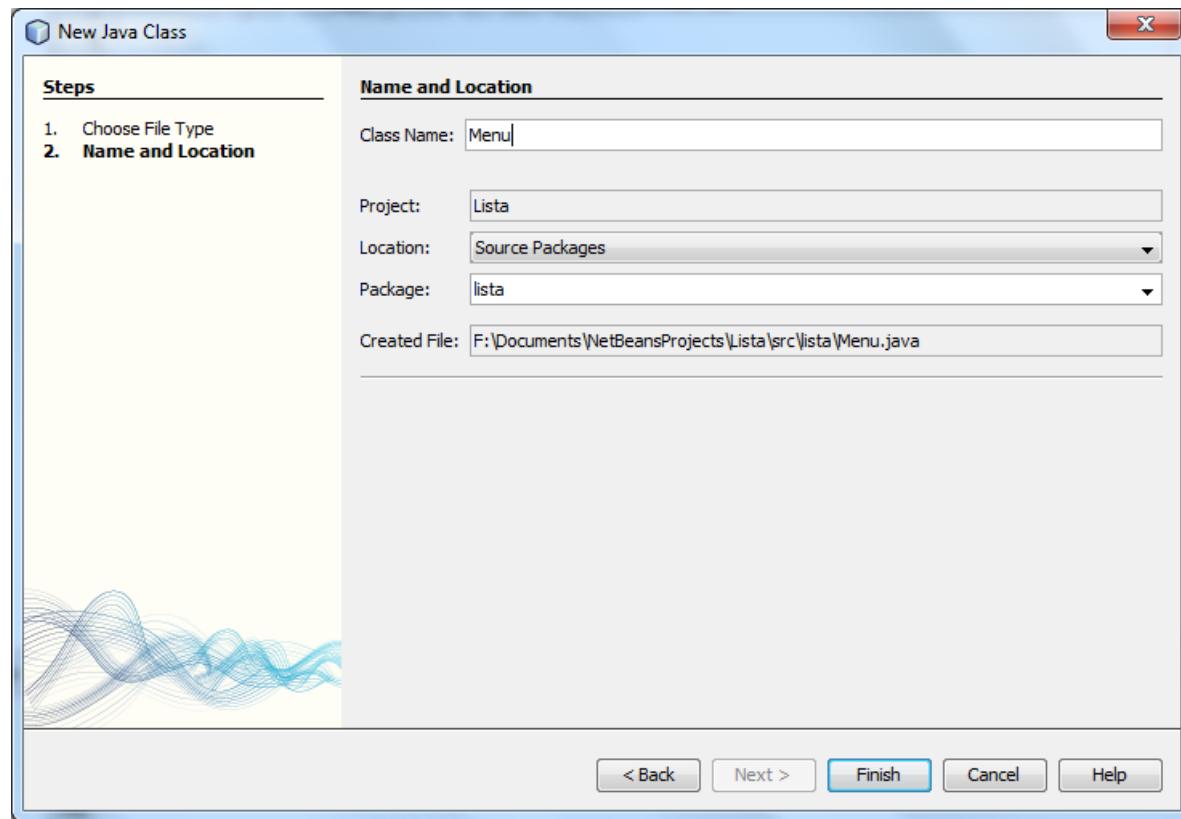


CREACION DE LA CLASE MENU

Se debe posicionar sobre el paquete lista y pulsar el botón derecho del mouse>>opción new>>Java Class



El nombre de la clase es Menu



Pulsar el Botón Finish

```
Colocar antes de public class Menu {
```

```
import javax.swing.*;
```

```
6   package lista;  
7   import javax.swing.*;  
8   public class Menu {  
9  
10  }  
11
```

El siguiente método se debe colocar después de public class Menu {

```
void opciones()  
{  
int i,opc,info;  
Listas L=new Listas();  
Object [] valores = {"1. Adicion","2. Borrar","3. Consulta","4.Salir"};  
do  
{  
    String resp=(String) JOptionPane.showInputDialog(null,"Elija la Opcion", "Entrada de  
datos",JOptionPane.QUESTION_MESSAGE, null, valores,valores[0]);  
    opc=Character.digit(resp.charAt(0),10);  
    switch(opc)  
    {  
        case 1:  
            info=Integer.parseInt(JOptionPane.showInputDialog(null," Digite Info a Adicionar:"));  
            L.adicion(info);  
            break;  
        case 2:  
            info=Integer.parseInt(JOptionPane.showInputDialog(null," Digite Info a Borrar:"));  
            L.borrar(info);  
            break;  
        case 3:  
            JOptionPane.showMessageDialog(null,L.consulta());  
            break;  
    }  
}  
while(opc!=4);  
System.exit(1);  
}
```

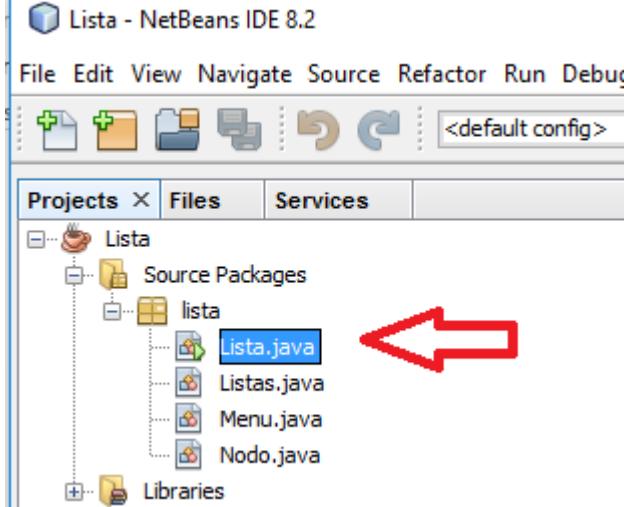
```

1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6   package lista;
7   import javax.swing.*;
8   public class Menu {
9     void opciones()
10    {
11      int i,opc,info;
12      Listas L=new Listas();
13      Object [] valores = {"1. Adicionar","2. Borrar","3. Consulta","4. Salir"};
14      do
15      {
16        String resp=(String) JOptionPane.showInputDialog(null,"Elija la Opcion", "Entrada");
17        opc=Character.digit(resp.charAt(0),10);
18        switch(opc)
19        {
20          case 1:
21            info=Integer.parseInt(JOptionPane.showInputDialog(null," Dигite Info a Adicionar"));
22            L.adicionar(info);
23            break;
24          case 2:
25            info=Integer.parseInt(JOptionPane.showInputDialog(null," Dигite Info a Borrar"));
26            L.borrar(info);
27            break;
28          case 3:

```

ACTUALIZAR LA CLASE Main

Seleccione la clase Lista.java del proyecto y de doble click

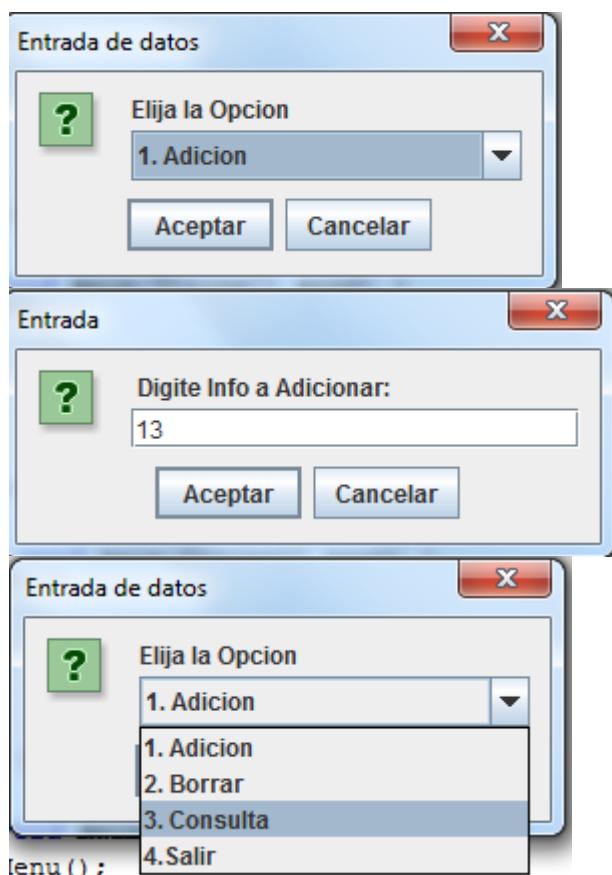


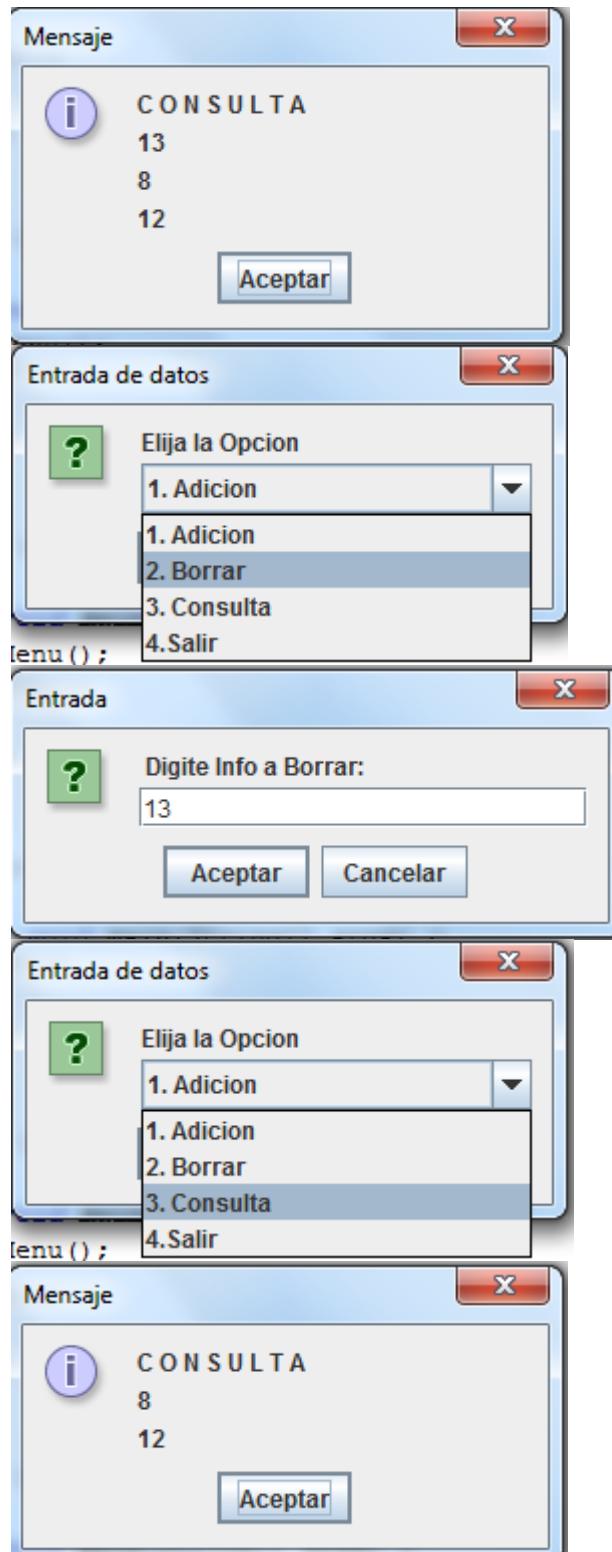
Escriba lo siguiente en // TODO code application logic here
 Menu M=new Menu();
 M.opciones();

Quedando:

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5
6  package lista;
7
8  /**
9   *
10  * @author funcionario
11  */
12 public class Main {
13
14     /**
15      * @param args the command line arguments
16     */
17     public static void main(String[] args) {
18         Menu M=new Menu();
19         M.opciones();
20     }
}
```

Pulsar la tecla F6 o el botón 





Ahora bien, si se desea que la Lista sea ordenada se debe cambiar:

```
15     boolean buscaInfo (int x)
16     {
17         p=q=cab;
18         while(p!=null && p.info!=x)
19         {
20             q=p;
21             p=p.sig;
22         }
23         if(p!=null)
24             return true;
25         else
26             return false;
27     }
```

Por el siguiente código:

```
boolean buscaInfo (int x)
{
    p=q=cab;
    while(p!=null && p.info<=x)
    {
        q=p;
        p=p.sig;
    }
    if(p==null)
        return false;
    return p.info==x;
}
```

Como también, void adicion

```
36     void adicion(int x)
37     {
38         if(!buscaInfo (x) )
39         {
40             r=new Nodo (x) ;
41             if(cab==null)
42                 cab=r;
43             else
44                 q.sig=r;
45             r.sig=null;
46         }
47     }
```

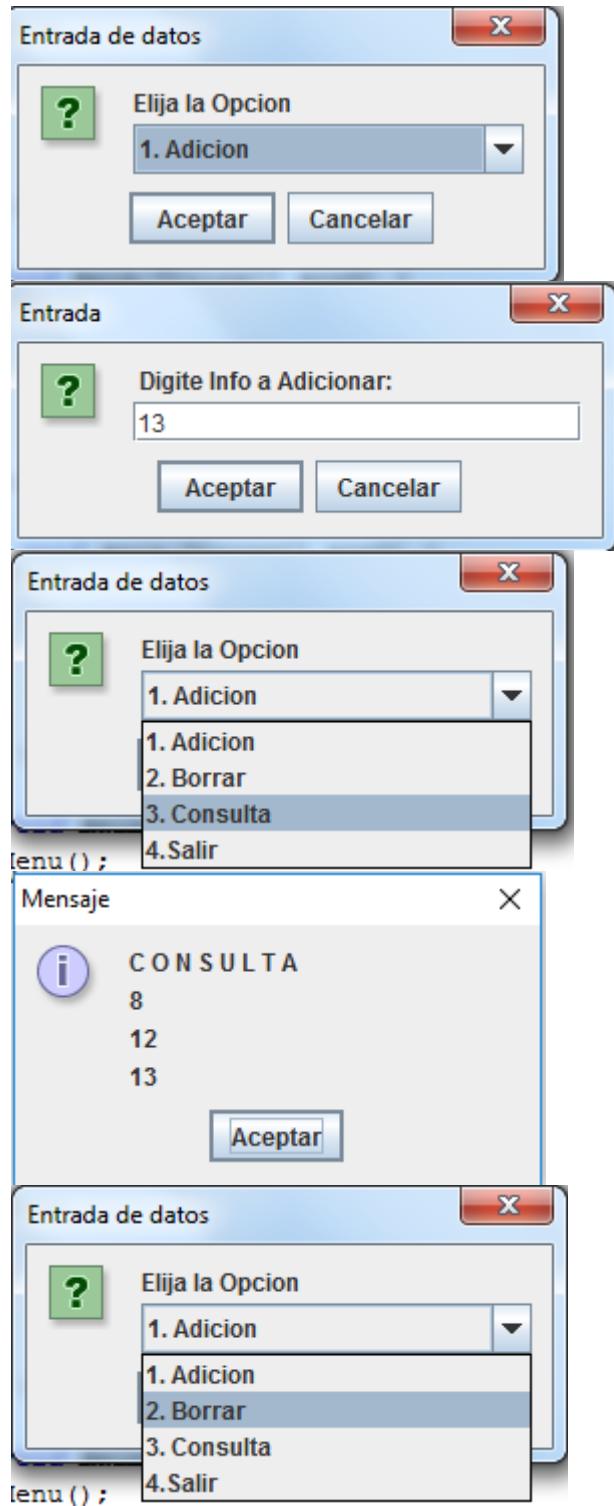
Por el siguiente código:

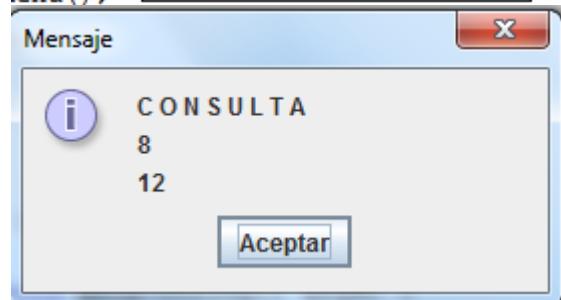
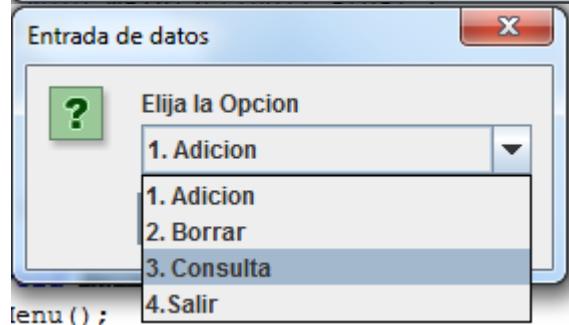
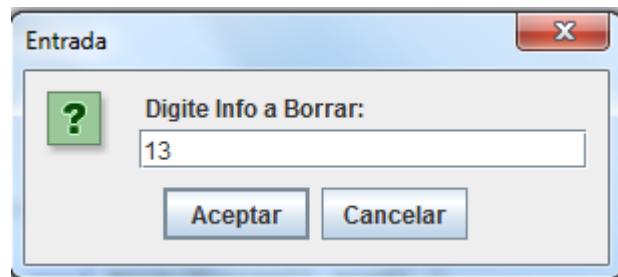
```
void adicion(int x)
{
if(!buscaInfo (x))
{
r=new Nodo(x);
if(cab==null)
cab=r;
else
if(p==q)
{
cab=r;
r.sig=p;
}
else
{
q.sig=r;
r.sig=null;
}
}
}
```

Quedando:

```
15     boolean buscaInfo (int x)
16 {
17     p=q=cab;
18     while(p!=null && p.info<=x)
19     {
20         q=p;
21         p=p.sig;
22     }
23     if(p==null)
24         return false;
25     return p.info==x;
26 }
27     void recorreLista()
28 {
29     p=q=cab;
30     while(p!=null)
31     {
32         q=p;
33         p=p.sig;
34     }
35 }
36     void adicion(int x)
37 {
38     if(!buscaInfo (x))
39     {
40         r=new Nodo(x);
41         if(cab==null)
42             cab=r;
43         else
44             if(p==q)
45             {
46                 cab=r;
47                 r.sig=p;
```

Pulsar la tecla F6 o el botón 





**UNIVERSIDAD SANTO TOMAS
DUAD
FACULTAD DE CYT
PROGRAMA INGENIERIA EN INFORMATICA**

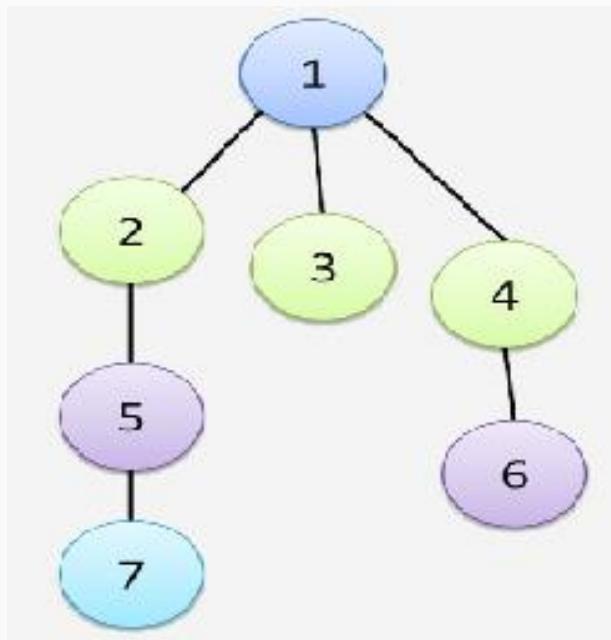
**PRACTICA ALGORITMO DE BÚSQUEDA EN ANCHURA (BFS) Y EN PROFUNDIDAD
((DFS))**

**MARIO DUSTANO CONTRERAS CASTRO
DOCENTE TIEMPO COMPLETO**

Algoritmo de Búsqueda en Anchura (BFS)

Tomado de: <https://www.bibliadelprogramador.com/2014/04/algoritmos-de-busqueda-en-anchura-bfs-y.html>

La **búsqueda en anchura** supone que el recorrido se haga por niveles. Para entender más fácilmente de que se trata, hemos indicado en la siguiente imagen un grafo ejemplo en donde cada color representa un nivel, tomando como raíz o nodo inicial el que tiene el número 1. El recorrido se hará en orden numérico de forma consecutiva hasta llegar al nodo número 7.



La estrategia que usaremos para garantizar este recorrido es utilizar una cola que nos permita almacenar temporalmente todos los nodos de un nivel, para ser procesados antes de pasar al siguiente nivel hasta que la cola esté vacía.

```
queue<State> q;  
q.push(State(raiz));  
mark[raiz] = true;
```

Inmediatamente después de declarar nuestra estructura de cola, agregamos el nodo raíz para poder iniciar el proceso de búsqueda. Esto se hace porque necesitamos tener al menos un elemento en nuestra cola, dado que la condición de salida es que la cola esté vacía. Luego marcamos el nodo raíz como visitado.

El algoritmo es el siguiente:

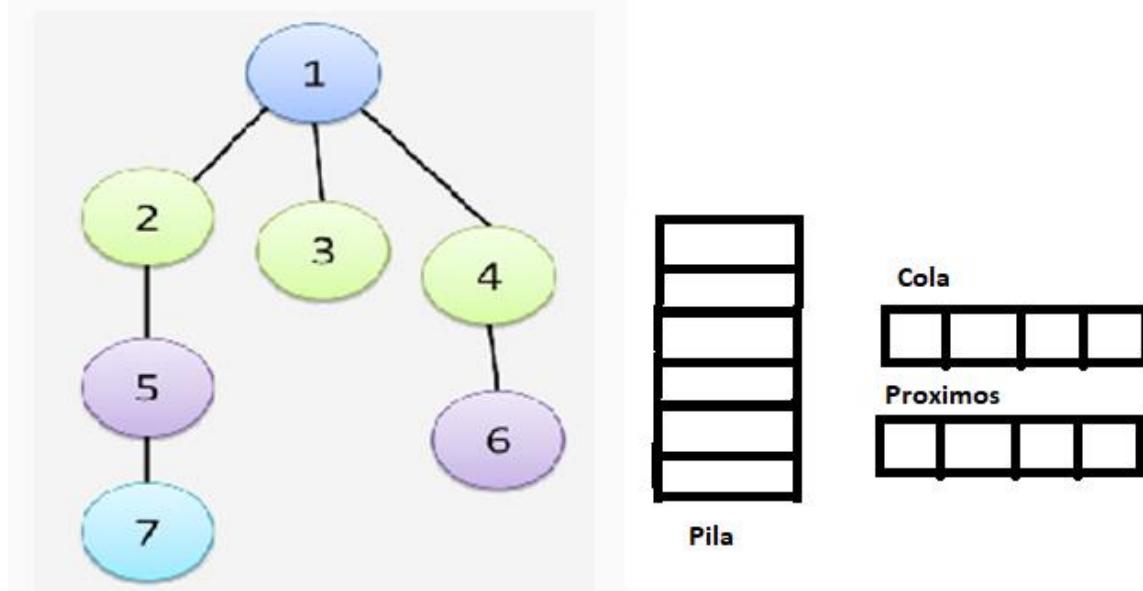
```
while(!q.empty())
{
    State st = q.front();
    q.pop();
    if (st.node == nodo){
        printf("%d\n",nodo);
        return;
    }else printf("%d ",st.node);
    int T = (int)graph.G[st.node].size();
    for(int i = 0; i < T; ++i)
    {
        if (!mark[graph.G[st.node][i].node])
        {
            mark[graph.G[st.node][i].node] = true;
            q.push(State(graph.G[st.node][i].node));
        }
    }
}
```

Cada vez que visitamos un nodo, lo desencolamos e imprimimos por pantalla el valor del nodo para ir indicando el recorrido. Luego agregamos a la cola todos los nodos del siguiente nivel y los marcamos como visitados antes de comenzar el ciclo de nuevo, en el que procesaremos estos nuevos nodos que hemos agregado a la cola.

Si encontramos el elemento buscado, la función BFS retornará al “main” e imprimirá entre comillas simples el valor del elemento buscado.

Podríamos hacer otro tipo de mensaje para indicar que el elemento ha sido encontrado, calcular el número de nodos que fueron visitados o incluso generar un mensaje especial en el caso de que el elemento no haya sido encontrado. Por el momento la función ha quedado lo más sencilla posible para enfocarnos únicamente en cómo hacer el recorrido.

Desarrollo del Docente



Momento 1. Pila 1

Imprime 1

Cola 2 3 4

Próximos 5 6

Momento 2. Pila 1 2 3 4

Imprime 2 3 4

Cola 5 6

Próximos 7

Momento 3. Pila 1 2 3 4 5 6

Imprime 5 6

Cola 7

Próximo: vacío

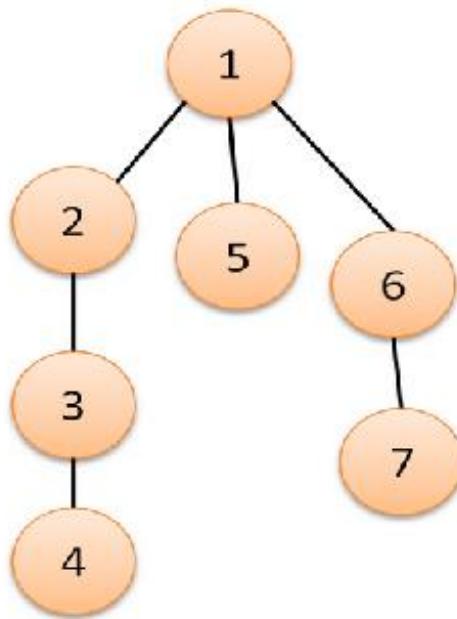
Momento 4. Pila 1 2 3 4 5 6 7

Imprime 7

Desempilar...

Algoritmo de Búsqueda en Profundidad (DFS)

En el caso de la **búsqueda en profundidad** lo que se quiere es recorrer desde la raíz hasta los nodos extremos u hojas por cada una de las ramas. En este caso los niveles de cada nodo no son importantes. En la siguiente imagen podemos ver el orden en que se hace el recorrido desde el nodo raíz, indicado con el número uno, hasta el nodo número siete.



La forma más intuitiva de hacer este algoritmo es de forma recursiva, de lo contrario tendríamos que usar en lugar de una cola una pila, pero con la recursión nos ahorramos la necesidad de utilizar esta estructura explícitamente y en lugar de ello nos valemos de la pila de recursión.

En este caso pasaremos por parámetro el nodo a buscar y el nodo actual (El nodo que está siendo visitado en cada ambiente de recursión), que en la primera llamada será el nodo raíz.

En cada llamada recursiva marcaremos el nodo actual como visitado y luego verificamos si es el nodo buscado para salir de la recursión, este será nuestro caso base. De no ser el nodo requerido, se hace la llamada recursiva con todos los nodos hijos del nodo actual, pero en este caso, a diferencia del **recorrido BFS**, no se visitarán todos los hijos de forma consecutiva, sino que el algoritmo recorrerá en profundidad hasta llegar a un nodo extremo o nodo hoja, antes de retornar al ambiente de recursión en donde se encuentran los otros nodos hijos.

El algoritmo es el siguiente:

```
void DFS(const int nodo, int nodoAct){  
    mark[nodoAct] = true;  
    if(mark[nodo]==true) return;  
    if (nodoAct == nodo){  
        printf("%d\n",nodo);  
        return;  
    }else if(mark[nodo]==true) return;  
    else{  
        printf("%d ",nodoAct);  
        int T = (int)graph.G[nodoAct].size();  
        for(int i = 0; i < T; ++i)  
        {  
            if (!mark[graph.G[nodoAct][i].node]) DFS(nodo, graph.G[nodoAct][i].node);  
        }  
    }  
}
```

El orden en que se eligen las ramas en un **recorrido DFS** está determinado por el tipo de recorrido de procesamiento de árbol que se haya elegido, estos pueden ser:

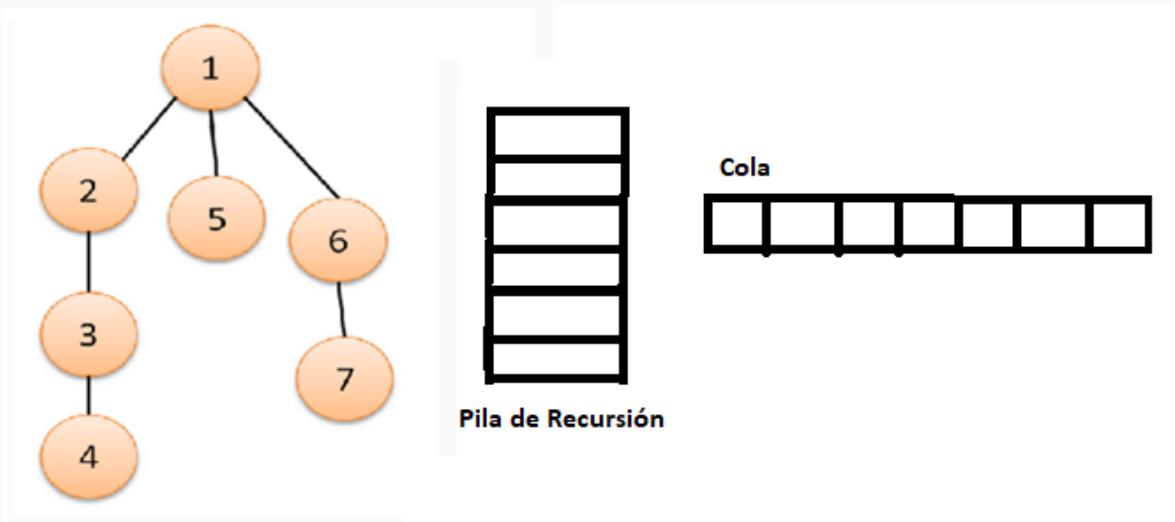
- **Pre-orden:** Se procesa primero la raíz, luego la rama izquierda y luego las ramas siguientes hasta llegar a la que se encuentra más a la derecha.
- **Post-orden:** Se procesa el árbol desde las ramas izquierdas hasta la que se encuentra más a la derecha. Finalmente se procesa el nodo raíz
- **Simétrico o In-orden:** Se procesa la rama de la izquierda, luego el nodo raíz y luego la rama derecha.

En este caso el recorrido es Pre-orden.

Tips:

1. La búsqueda en anchura se recomienda cuando lo que se necesita es buscar el camino más corto en grafos no ponderados.
2. La búsqueda en profundidad se puede utilizar para detectar ciclos en un grafo, determinar si un grafo es conexo o no y cuántas componentes conexas tiene, determinar puntos de articulación y biconexión de grafos, entre otras cosas.
3. Cuando no tiene importancia el orden en que visitemos los nodos y aristas del grafo, se puede usar cualquiera de los dos algoritmos.

Desarrollo del Docente



Padre 1

Cola 2 5 6

Primer momento de Pila de Recursión:

1 2 3 4

Como 4 no tiene hijos...se imprime

1 2 3

Como 3 no tiene más hijos...se desemPila de Recursión..se imprime

1 2

Como 2 no tiene más hijos ...se desemPila de Recursión ..se imprime

1 tiene más hijos ...

Ahora segunda momento de Pila de Recursión

1 5

Como 5 no tiene más hijos ...se desemPila de Recursión ..se imprime

1 tiene más hijos ...

Ahora tercer momento de Pila de Recursión...

1 6 7

Como 7 no tiene más hijos ...se desemPila de Recursión ..se imprime

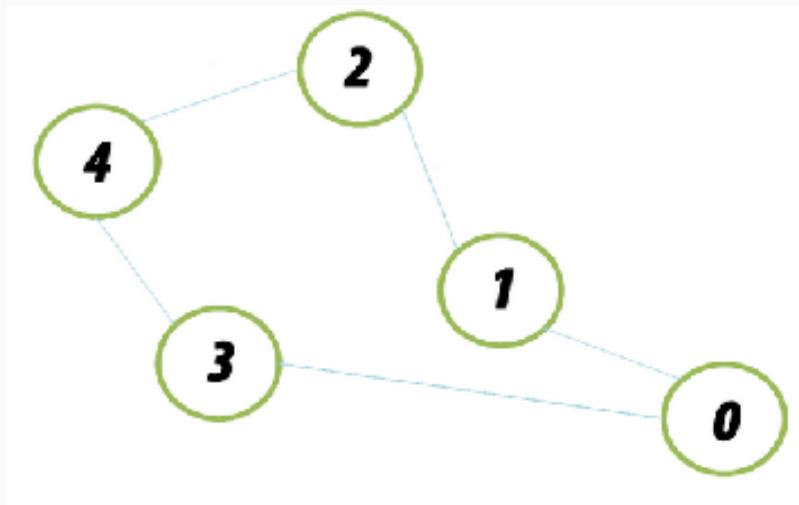
1 6

Como 6 no tiene más hijos ...se desemPila de Recursión ..se imprime

Como 1 no tiene más hijos ...se desemPila de Recursión ..se imprime

Un grafo de ejemplo

Supongamos que tenemos el siguiente grafo leído:



Queremos ver el orden en que cada algoritmo ha de recorrer el grafo. Llamaremos a la **función BFS** en el main y posteriormente la **función DFS**. Tomaremos el nodo '2' como nuestro nodo raíz para ambos algoritmos y la salida obtenida será:

```
5 5  
2 4  
2 1  
4 3  
1 0  
3 0  
2 0  
  
BFS  
2 4 1 3 '0'  
  
DFS  
2 4 3 '0'
```

Conclusiones

Los **algoritmos de búsqueda BFS y DFS** son una de las herramientas básicas a la hora de trabajar con grafos. No sólo podremos usarlos para recorrer grafos o buscar elementos, sino que también podemos adaptarlos y mejorarllos para resolver de manera eficiente cualquier tipo de situaciones que podamos **moldear como un grafo o un árbol**.

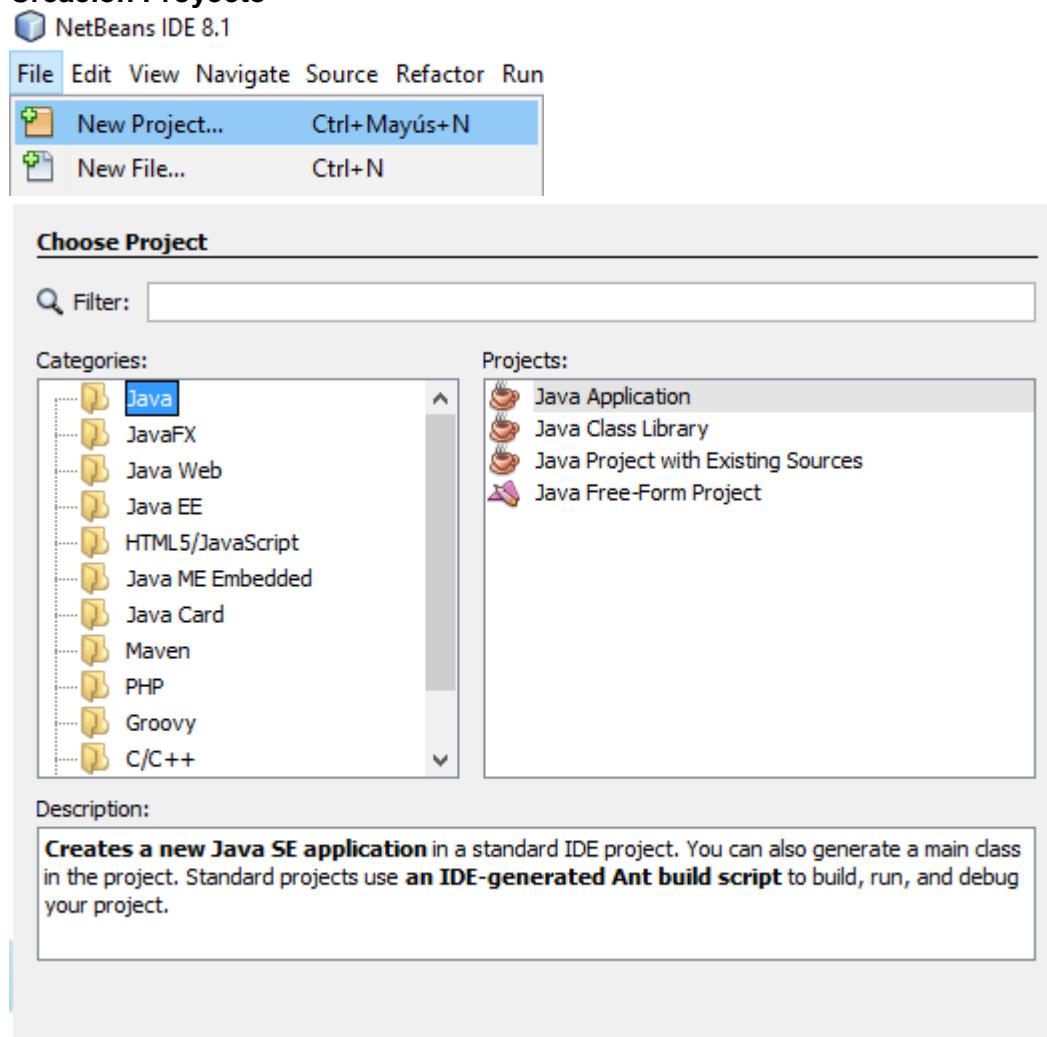
PRACTICA PLANTEADA

Estudiantes: Sergio Andrés Galvis Velásquez
Andrés Felipe Barreto Duarte
Programa: Ingeniería de Sistemas
Espacio Académico: Estructura de Datos
Universidad Autónoma de Colombia
ImplementacionGrafo (1) > ImplementacionGrafo > src > grafos

Nombre	Tipo
Estado	Archivo JAVA
Grafo	Archivo JAVA
ImplementacionGrafo	Archivo JAVA
Nodo	Archivo JAVA

PRACTICA AJUSTADA PLATAFORMA NETBEANS

Creación Proyecto



Pulsar Botón Next >

Project Name: AlgoritmoBusquedaAnchuraProfundidad

Name and Location

Project Name:

Project Location: Browse...

Project Folder:

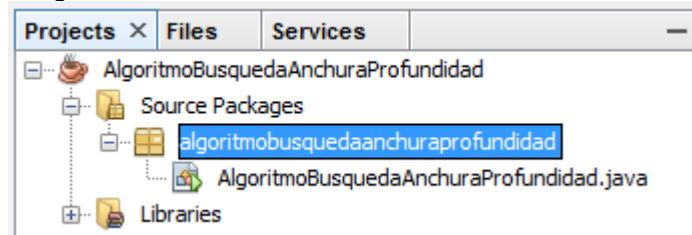
Use Dedicated Folder for Storing Libraries
Libraries Folder: Browse...

Different users and projects can share the same compilation libraries (see Help for details).

Create Main Class

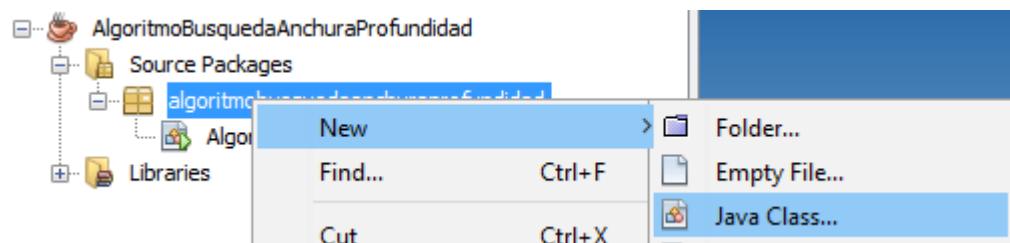
Pulsar Botón Finish

Se genera...



Creación Programa Enumeration Estado

Sobre `algoritmobusquedaanchuraprofundidad` pulsar botón derecho del mouse



Class Name: Estado

Name and Location

Class Name:	Estado
Project:	AlgoritmoBusquedaAnchuraProfundidad
Location:	Source Packages
Package:	algoritmobusquedaanchuraprofundidad
Created File:	ioBusquedaAnchuraProfundidad\src\algoritmobusquedaanchuraprofundidad\Estado.java

Pulsar Botón

Reemplazar contenido por:

```
package algoritmobusquedaanchuraprofundidad;
public enum Estado {
    NoVisitado, Visitando, Visitado;
```

}

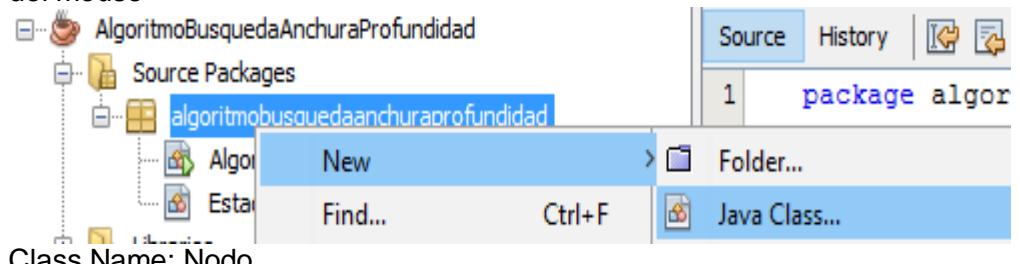
The screenshot shows the Eclipse IDE's code editor with the generated Java code for the `Estado` enum. The code is identical to the one provided above. The code editor has tabs for 'Source' and 'History' at the top, and various toolbars and status bars below.

```
1 package algoritmobusquedaanchuraprofundidad;
2 public enum Estado {
3     NoVisitado, Visitando, Visitado;
4 }
```

Dar click en

Adición Clase Nodo

Sobre  Source Packages
algoritmobusquedaanchuraprofundidad pulsar botón derecho del mouse



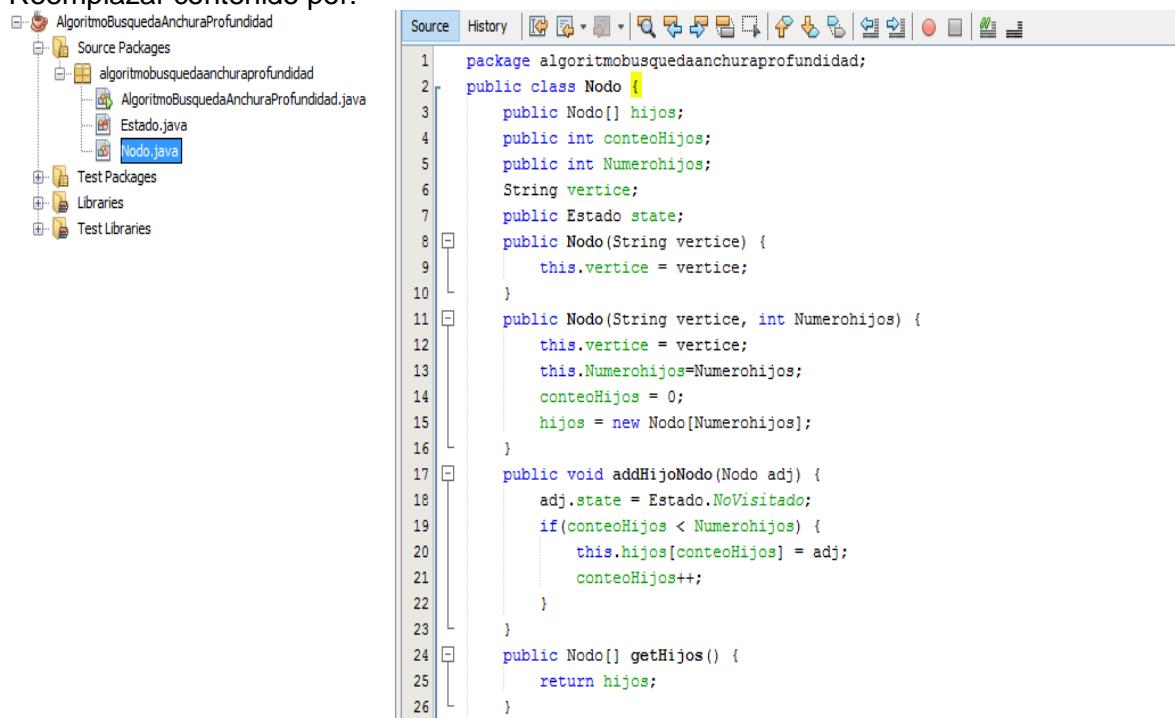
Class Name: Nodo

Name and Location	
Class Name:	Nodo
Project:	AlgoritmoBusquedaAnchuraProfundidad
Location:	Source Packages
Package:	algoritmobusquedaanchuraprofundidad
Created File:	moBusquedaAnchuraProfundidad\src\algoritmobusquedaanchuraprofundidad\Nodo.java

Pulsar Botón

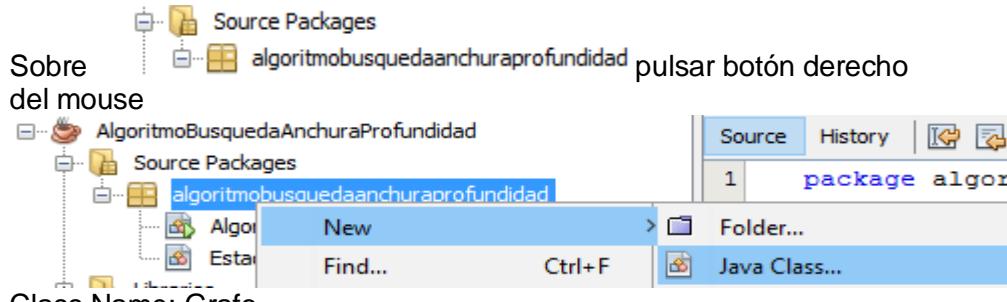
Finish

Reemplazar contenido por:



```
package algoritmobusquedaanchuraprofundidad;
public class Nodo {
    public Nodo[] hijos;
    public int conteoHijos;
    public int Numerohijos;
    String vertice;
    public Estado state;
    public Nodo(String vertice) {
        this.vertice = vertice;
    }
    public Nodo(String vertice, int Numerohijos) {
        this.vertice = vertice;
        this.Numerohijos=Numerohijos;
        conteoHijos = 0;
        hijos = new Nodo[Numerohijos];
    }
    public void addHijoNodo(Nodo adj) {
        adj.state = Estado.NoVisitado;
        if(conteoHijos < Numerohijos) {
            this.hijos[conteoHijos] = adj;
            conteoHijos++;
        }
    }
    public Nodo[] getHijos() {
        return hijos;
    }
    public String getVertice()
    {
        return vertice;
    }
    public int getNumeroHijos()
    {
        return Numerohijos;
    }
    void inicial()
    {
        state=Estado.NoVisitado;
    }
}
```

Adición Clase Grafo



Class Name: Grafo

The 'Name and Location' dialog is displayed. The 'Class Name' field contains 'Grafo'. The 'Project' field is 'AlgoritmoBusquedaAnchuraProfundidad'. The 'Location' field is 'Source Packages'. The 'Package' field is 'algoritmobusquedaanchuraprofundidad'. The 'Created File' field shows the full path: 'moBusquedaAnchuraProfundidad\src\algoritmobusquedaanchuraprofundidad\Grafo.java'. A 'Finish' button is visible at the bottom right.

Pulsar Botón

The code editor displays the generated Java code for the 'Grafo' class. The code includes a package declaration for 'algoritmobusquedaanchuraprofundidad', a class definition for 'Grafo' with a constructor that initializes an array of 'Nodo' objects and sets the 'conteo' variable to 0. The code editor interface is visible on the right.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package algoritmobusquedaanchuraprofundidad;

/**
 * @author Docen.informatica
 */
public class Grafo {
```

Reemplazar contenido por:

```
package algoritmobusquedaanchuraprofundidad;
import java.util.LinkedList;
import java.util.Queue;
import javax.swing.JOptionPane;
public class Grafo {
    int i,j,k,m,n;
    String c;
    public int conteo; //Número de vértices
    Nodo vertices[];
    int d;
    public Grafo(int n) {
        this.n=n;
        vertices = new Nodo[n];
        conteo = 0;
    }
    public Nodo[] getNodo() {
        return vertices;
    }
}
```

```

public int getNodos()
{
    return d;
}
void crearNodos()
{
    for(i=0;i<n;i++)
    {
        c=String.valueOf(i+1);
        m=Integer.parseInt(JOptionPane.showInputDialog(null,"Nodo:
"+c+" Numero de Hijos:"));
        vertices[i] = new Nodo(c, m);
    }
    for(i=0;i<n;i++)
    {
        c=String.valueOf(i+1);
        for(j=1;j<=vertices[i].getNumeroHijos();j++)
        {
            do
            {
                k=Integer.parseInt(JOptionPane.showInputDialog(null," Nodo
Adyacente de :" +c));
            }
            while(k<i || k>n);
            vertices[i].addHijoNodo(vertices[k-1]);
        }
    }
}
public void dfs(Nodo root) {
    //Evitar bucles infinitos en la visita de los nodos.
    if(root == null)
        return;
    System.out.print(root.getVertice() + "\t");
    root.state = Estado.Visitado;
    //Para todos los hijos.
    for(Nodo p: root.getHijos()) {
        //Si el estado de los hijos no se visita, se usa la función
        recursiva.
        if(p.state == Estado.NoVisitado) {
            dfs(p);
        }
    }
}
public void bfs(Nodo root) {
    //Dado que la cola es una interfaz.
    Queue<Nodo> queue = new LinkedList<>();
    if(root == null)
        return;
    root.state = Estado.Visitado;
    //Adiciona al final de la cola
    queue.add(root);
}

```

```

        while(!queue.isEmpty()) {
            //Quita del frente de la cola
            Nodo r = queue.remove();
            System.out.print(r.getVertice() + "\t");
            //Visita los hijos primero, antes que los nietos.
            for(Nodo p: r.getHijos()) {
                if(p.state == Estado.NoVisitado) {
                    queue.add(p);
                    p.state = Estado.Visitado;
                }
            }
        }
    }

    public Nodo primero()
    {
        return vertices[0];
    }

    void inicial()
    {
        for(i=0;i<n;i++)
            vertices[i].inicial();
    }
}
}

```

The screenshot shows an IDE interface with two main panes. On the left is a file browser showing a package named 'algoritmobusquedaanchuraprofundidad' containing four files: 'AlgoritmoBusquedaAnchuraProfundidad.java', 'Estado.java', 'Grafo.java', and 'Nodo.java'. Below this are sections for 'Test Packages', 'Libraries', and 'Test Libraries'. On the right is a code editor pane titled 'Source' with the following Java code:

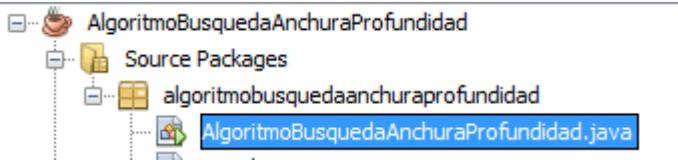
```

1 package algoritmobusquedaanchuraprofundidad;
2 import java.util.LinkedList;
3 import java.util.Queue;
4 import javax.swing.JOptionPane;
5 public class Grafo {
6     int i,j,k,m,n;
7     String c;
8     public int conteo; //Número de vértices
9     Nodo vertices[];
10    int d;
11    public Grafo(int n) {
12        this.n=n;
13        vertices = new Nodo[n];
14        conteo = 0;
15    }
16    public Nodo[] getNode() {
17        return vertices;
18    }
19    public int getNodos() {
20        return d;
21    }
22    void crearNodos()
23

```

Dar click en

Edición Programa Java –
AlgoritmoBusquedaAnchurayProfundidad –
Seleccionar dando un click....



Reemplazar contenido con:

```
package algoritmobusquedaanchuraprofundidad;
import javax.swing.*;
public class AlgoritmoBusquedaAnchuraProfundidad {
    public static void main(String[] args) {
        int n;
        Nodo T[];
        Nodo U[];
        do
        {
            n = Integer.parseInt(JOptionPane.showInputDialog(null,"Número
de Nodos(Como mínimo 5):"));
        }
        while(n<5);
        Grafo gDfs = new Grafo(n);
        gDfs.crearNodos();
        System.out.println("-----BÚSQUEDA POR PROFUNDIDAD-
-----");
        gDfs.inicial();
        gDfs.dfs(gDfs.primero());
        System.out.println("");
        System.out.println("");
        System.out.println("-----BÚSQUEDA POR ANCHURA-----
-----");
        gDfs.inicial();
        gDfs.bfs(gDfs.primero());
        System.out.println("");
    }
}
```

The screenshot shows the Eclipse IDE interface. On the left is the Project Explorer with a package named 'algoritmobusquedaanchuraprofundidad' containing files 'AlgoritmoBusquedaAnchuraProfundidad.java', 'Estado.java', 'Grafo.java', and 'Nodo.java'. Below it are 'Test Packages', 'Libraries', and 'Test Libraries'. On the right is the Source Editor showing the Java code for the main class:

```

1 package algoritmobusquedaanchuraprofundidad;
2 import javax.swing.*;
3 public class AlgoritmoBusquedaAnchuraProfundidad {
4     public static void main(String[] args) {
5         int n;
6         Nodo T[];
7         Nodo U[];
8         do
9         {
10            n = Integer.parseInt(JOptionPane.showInputDialog(null,"Numero de Nodos(Como minimo 5):"));
11        }
12        while(n<5);
13        Grafo gDfs = new Grafo(n);
14        gDfs.crearNodos();
15        System.out.println("-----BÚSQUEDA POR PROFUNDIDAD-----");
16        gDfs.inicial();
17        gDfs.dfs(gDfs.primer());
18        System.out.println("");
19        System.out.println("");
20        System.out.println("-----BÚSQUEDA POR ANCHURA-----");
21        gDfs.inicial();
22        gDfs.bfs(gDfs.primer());
23        System.out.println("");
24    }
25 }

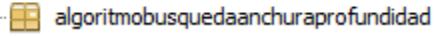
```

Dar click en

Adición Archivo Texto

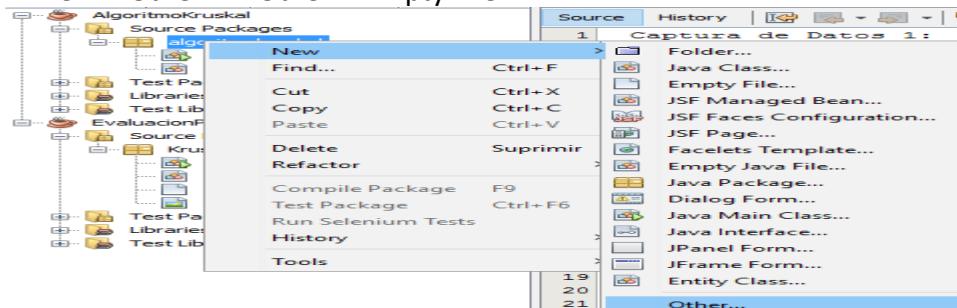


Sobre



pulsar botón derecho

>>New>>Other...>>Other>>Empty File

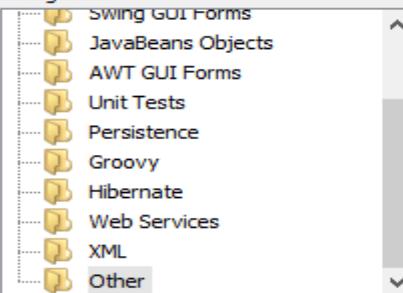


Choose File Type

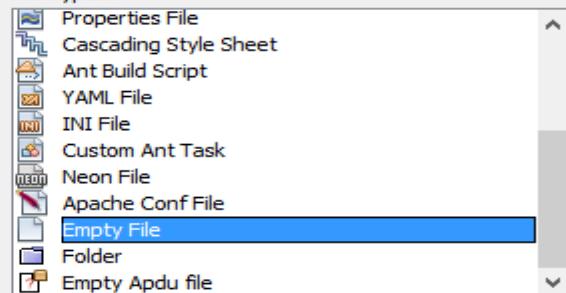
Project: AlgoritmoBusquedaAnchuraProfundidad

Filter:

Categories:



File Types:



Next >

Pulsar botón **Next >**

Name and LocationFile Name: Project: Folder: Created File:

FileName: grafo.txt

Pulsar Botón



Reemplazar contenido con:

Número de Nodos: 7

Nodo 1: 3

Nodo 2: 1

Nodo 3: 1

Nodo 4: 0

Nodo 5: 0

Nodo 6: 1

Nodo 7:0

Nodos Adyacentes de 1: 2, 5, 6

Nodos Adyacentes de 2: 3

Nodos Adyacentes de 3: 4

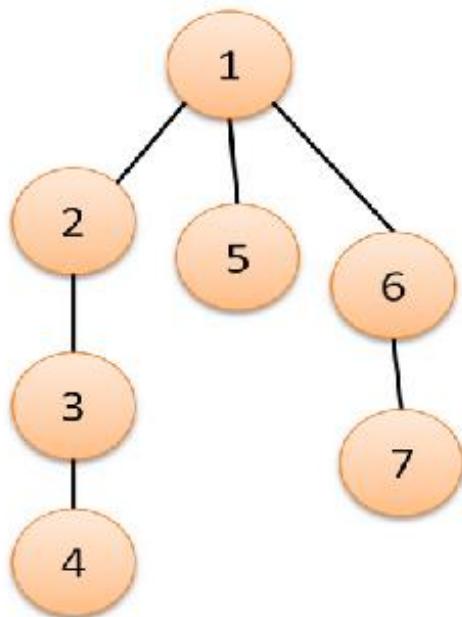
Nodos Adyacentes de 6: 7



Dar click en

Ejecución

Se realiza el siguiente grafo



Pulsar F5 o

Sin título: Bloc de ... Archivo Edición Formato Ver Ayuda

Número de Nodos: 7
Número 1: 3
Número 2: 1
Número 3: 1
Número 4: 0
Número 5: 0
Número 6: 1
Número 7: 0

Nodos Adyacentes de 1: 2, 5, 6
Nodos Adyacentes de 2: 3
Nodos Adyacentes de 3: 4
Nodos Adyacentes de 6: 7

Entrada

Numero de Nodos(Como mínimo 5):

Aceptar Cancelar

Output x grafo.txt x

AlgoritmoBusquedaAnchuraProfundidad (run) x AlgoritmoBusquedaAnchuraProfundidad (run) #2 x

run:
-----BÚSQUEDA POR PROFUNDIDAD-----
1 2 3 4 5 6 7
-----BÚSQUEDA POR ANCHURA-----
1 2 5 6 3 7 4
BUILD SUCCESSFUL (total time: 50 seconds)

**UNIVERSIDAD SANTO TOMAS
DUAD
FACULTAD DE CYT
PROGRAMA INGENIERIA EN INFORMATICA**

PRACTICA ALGORITMO DE BÚSQUEDA A*

**MARIO DUSTANO CONTRERAS CASTRO
DOCENTE TIEMPO COMPLETO**

Algoritmo de A*

Tomado de: https://es.wikipedia.org/wiki/Algoritmo_de_b%C3%BAqueda_A%2A

El problema de algunos algoritmos de búsqueda en grafos informados, como puede ser el algoritmo voraz, es que se guían en exclusiva por la función heurística, la cual puede no indicar el camino de coste más bajo, o por el coste real de desplazarse de un nodo a otro (como los algoritmos de escalada), pudiéndose dar el caso de que sea necesario realizar un movimiento de coste mayor para alcanzar la solución. Es por ello bastante intuitivo el hecho de que un buen algoritmo de búsqueda informada debería tener en cuenta ambos factores, el valor heurístico de los nodos y el coste real del recorrido.

Así, el algoritmo A* utiliza una función de evaluación $f(n)=g(n)+h'(n)$ donde $h'(n)$ representa el valor heurístico del nodo a evaluar desde el actual, n, hasta el final, y $g(n)$, el coste real del camino recorrido para llegar a dicho nodo, n, desde el nodo inicial. A* mantiene dos estructuras de datos auxiliares, que podemos denominar abiertos, implementado como una cola de prioridad (ordenada por el valor $f(n)$ de cada nodo), y cerrados, donde se guarda la información de los nodos que ya han sido visitados. En cada paso del algoritmo, se expande el nodo que esté primero en abiertos, y en caso de que no sea un nodo objetivo, calcula la $f(n)$ de todos sus hijos, los inserta en abiertos, y pasa el nodo evaluado a cerrados.

El algoritmo es una combinación entre búsquedas del tipo primero en anchura con primero en profundidad: mientras que $h'(n)$ tiende a primero en profundidad, $g(n)$ tiende a primero en anchura. De este modo, se cambia de camino de búsqueda cada vez que existen nodos más prometedores.

Propiedades

Como todo algoritmo de búsqueda en amplitud, A* es un algoritmo completo: en caso de existir una solución, siempre dará con ella.

Si para todo nodo n del grafo se cumple $g(n)=0$, nos encontramos ante una búsqueda voraz. Si para todo nodo n del grafo se cumple $h(n)=0$, A* pasa a ser una búsqueda de coste uniforme no informada.

Para garantizar la optimización del algoritmo, la función $h(n)$ debe ser heurística admisible, esto es, que no sobreestime el coste real de alcanzar el nodo objetivo.

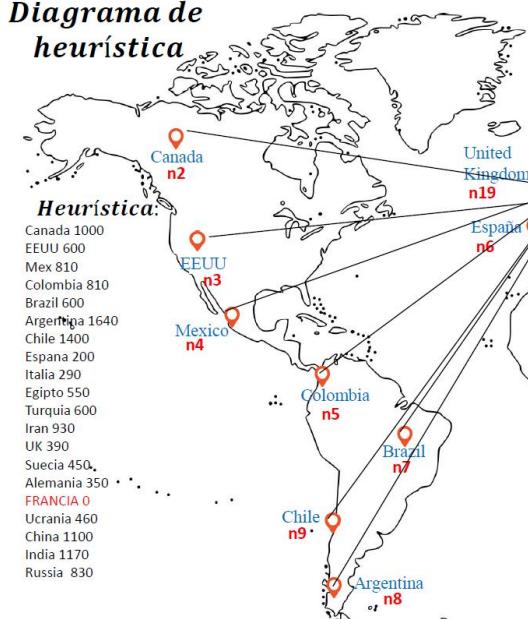
De no cumplirse dicha condición, el algoritmo pasa a denominarse simplemente A, y a pesar de seguir siendo completo, no se asegura que el resultado obtenido sea el camino de coste mínimo.

Asimismo, si garantizamos que $h(n)$ es consistente (o monótona), es decir, que para cualquier nodo n y cualquiera de sus sucesores, el coste estimado de alcanzar el objetivo desde n no es mayor que el de alcanzar el sucesor más el coste de alcanzar el objetivo desde el sucesor.

PRACTICA PLANTEADA

Estudiante: Rafael Ángel Ramírez Serna
 Programa: Ingeniería de Sistemas
 Espacio Académico: Estructura de Datos
 Universidad Autónoma de Colombia

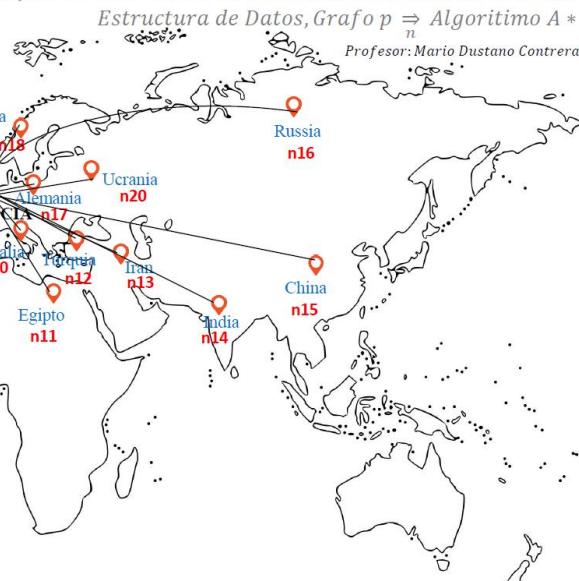
Diagrama de heurística



Rafael A. Ramirez Serna Universidad Autónoma de Colombia

Estructura de Datos, Grafo $p \Rightarrow n$ Algoritmo A *

Profesor: Mario Dusano Contreras

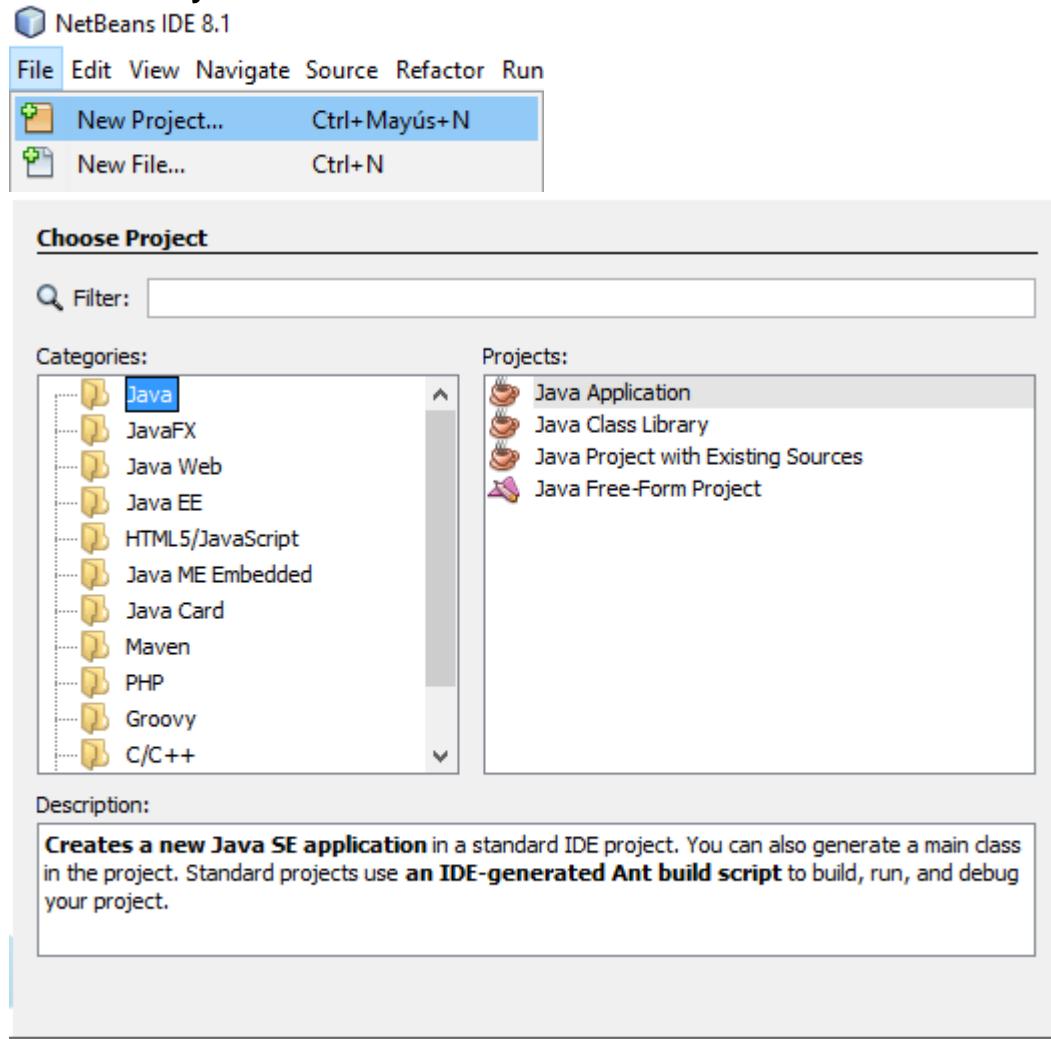


Grafo $P = (V, E)$

Rafael Ramirez Serna
 Estructura de Datos
 Prof. Héctor Dusano Contreras.

PRACTICA AJUSTADA PLATAFORMA NETBEANS

Creación Proyecto



Pulsar Botón

[Next >](#)

Project Name: GrafoBusquedaAPaises

Name and Location

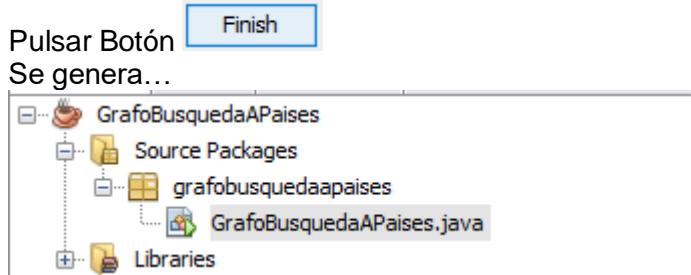
Project Name:

Project Location:

Project Folder:

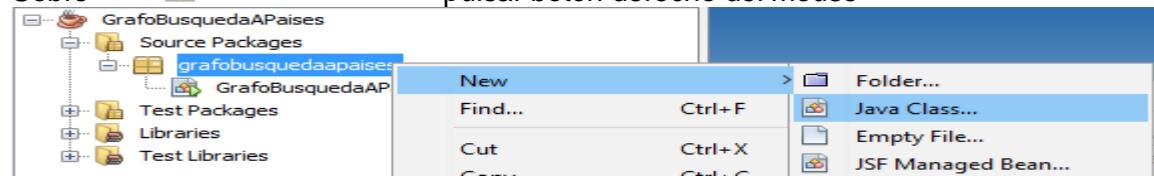
Use Dedicated Folder for Storing Libraries
Libraries Folder:
Different users and projects can share the same compilation libraries (see Help for details).

Create Main Class



Adición Clase Arista

Sobre **grafobusquedaapaises** pulsar botón derecho del mouse

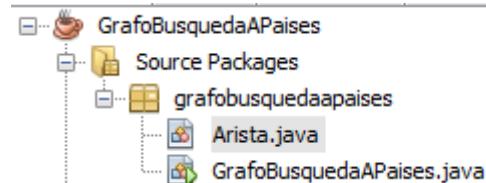


Class Name: Arista



Pulsar Botón

Finish



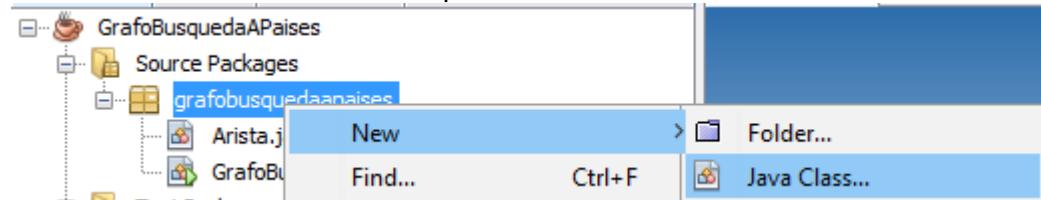
Reemplazar contenido por:

```
package grafobusquedaapaises;
class Arista {
    public int distancia;
    public Nodo destino;
    public Arista(Nodo destino, int distancia) {
        this.destino=destino;
        this.distancia=distancia;
    }
    public void actualiza(Nodo destino, int distancia) {
        this.destino=destino;
        this.distancia=distancia;
    }
}
```

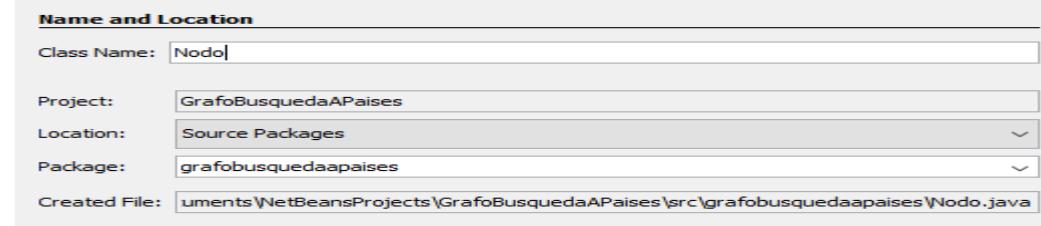
Se genera.... error por no se ha creado la clase Nodo

Adición Clase Nodo

Sobre **grafobusquedaapaises** pulsar botón derecho del mouse

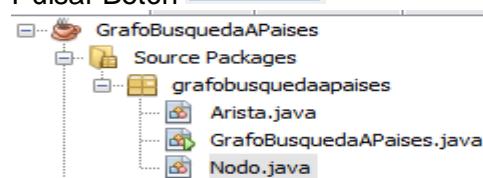


Class Name: Nodo



Pulsar Botón

Finish



Reemplazar contenido por:

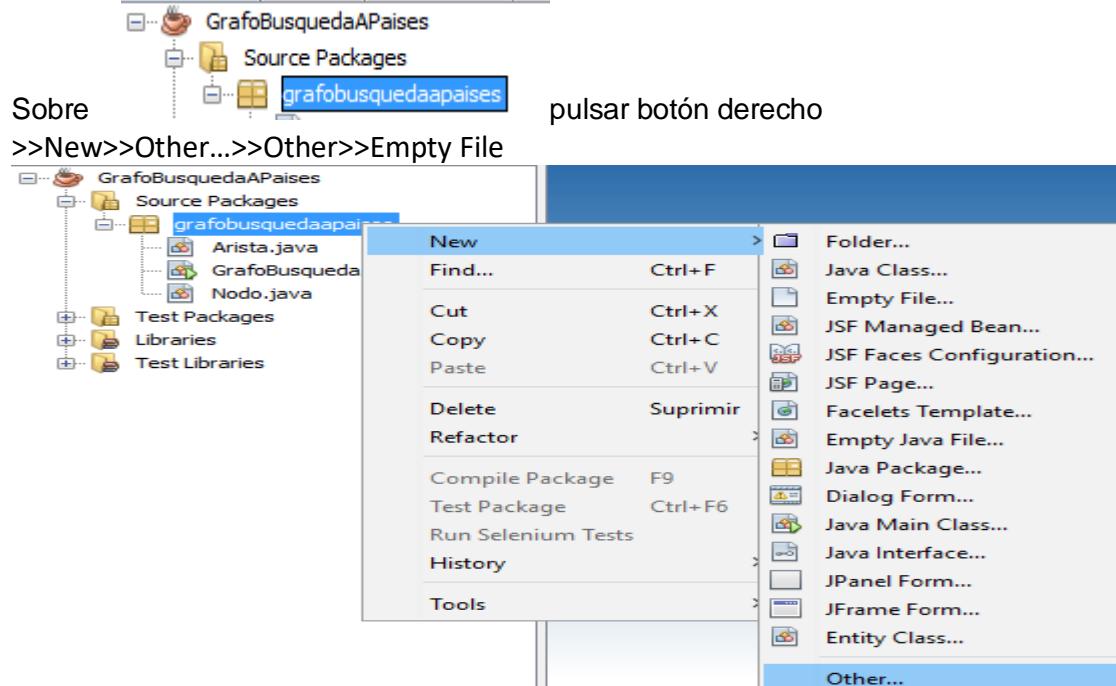
```
package grafobusquedaapaises;
class Nodo {
    public String pais;
    public int distancia;
    public int g_scores;
    public int h_scores;
    public int f_scores = 0;
    public Arista[] adyacencias;
    public Nodo parent;
    public Nodo(String pais, int distancia) {
        this.pais = pais;
        this.distancia = distancia;
    }
    @Override
    public String toString() {
        return pais+" "+String.valueOf(distancia);
    }
    public int dimension()
    {
        return adyacencias.length;
    }
}
```

Dar click en



Adición Archivo Texto Países

Sobre pulsar botón derecho
>>New>>Other...>>Other>>Empty File



Choose File Type

Project: GrafoBusquedaAPaises

Filter:

Categories:

- SWING GUI Forms
- JavaBeans Objects
- AWT GUI Forms
- Unit Tests
- Persistence
- Groovy
- Hibernate
- Web Services
- XML
- Other

File Types:

- Properties File
- Cascading Style Sheet
- Ant Build Script
- YAML File
- INI File
- Custom Ant Task
- Neon File
- Apache Conf File
- Empty File**
- Folder
- Empty Apdu file

Next >

Pulsar botón

Name and Location

File Name: pais.txt

Project: GrafoBusquedaAPaises

Folder: src\grafobusquedaapaises

Browse...

Created File: C:\documents\NetBeansProjects\GrafoBusquedaAPaises\src\grafobusquedaapaises\pais.txt

FileName: pais.txt

Pulsar Botón

Source History

```

1
2

```



Colocar el Texto Aquí

Francia
 Canadá
 EEUU
 México
 Colombia
 España
 Brasil
 Argentina
 Chile
 Italia
 Egipto
 Turquía
 Irán
 India
 China
 Rusia
 Alemania
 Suecia
 Reino Unido
 Ucrania

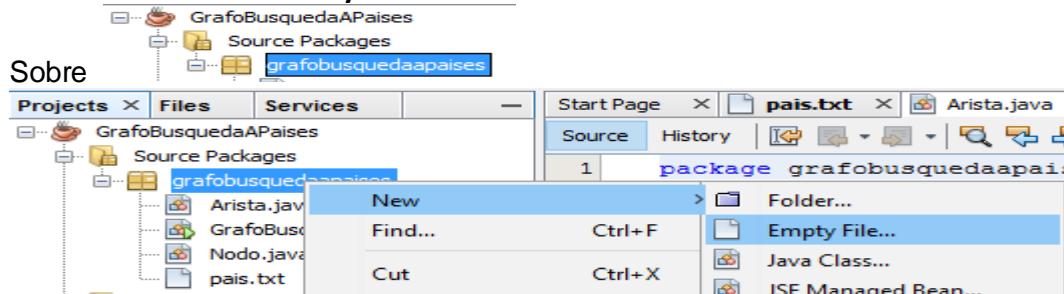
Source History

```

1 Francia
2 Canadá
3 EEUU
4 México
5 Colombia
6 España
7 Brasil
8 Argentina
9 Chile
10 Italia
11 Egipto
12 Turquía
13 Irán
14 India
15 China
16 Rusia
17 Alemania
18 Suecia
19 Reino Unido
20 Ucrania

```

Adición Archivo Texto Adyacencia



Name and Location

File Name:

Project:

Folder:

Created File:

FileName: adyacencia.txt

Colocar el siguiente contenido en el archivo:

Francia(0)

Adyacencias/Distancias (7)

Reino Unido(18)/400

Suecia(17)/450

Alemania(16)/350

Italia(9)/290

España(5)/200

Brasil(6)/610

México/810

Canadá(1)

Adyacencias/Distancias (2)

Reino Unido(18)/730

EEUU(2)/290

EEUU(2)

Adyacencias/Distancias (3)

Canadá(1)/290

México/210

Colombia(4)/450

México(3)

Adyacencias/Distancias (4)

EEUU(2)/210

Colombia(4)/290

Reino Unido(18)/730
Francia(0)/810

Colombia(4)
Adyacencias/Distancias (4)
EEUU(2)/450
México/290
Brasil(6)/500
Argentina(7)/780

España(5)
Adyacencias/Distancias (3)
Francia(0)/200
Italia(9)/300
Brasil(6)/450

Brasil(6)
Adyacencias/Distancias (6)
Colombia(4)/500
Argentina(7)/590
Chile(8)/510
Francia(0)/610
España(5)/450
Italia(9)/620

Argentina(7)
Adyacencias/Distancias (3)
Colombia(4)/780
Brasil(6)/590
Argentina(7)/270

Chile(8)
Adyacencias/Distancias (2)
Brasil(6)/510
Argentina(7)/270

Italia(9)
Adyacencias/Distancias (6)
Brasil(6)/620
España(5)/300
Francia(0)/290
Alemania(16)/440
Ucrania(19)/340
Egipto(10)/290

Egipto(10)
Adyacencias/Distancias (3)
Italia(9)/290
Turquía(11)/180
Irán(12)/450

Turquía(11)
Adyacencias/Distancias (3)
Egipto(10)/180
Irán(12)/350
Ucrania(19)/200

Irán(12)
Adyacencias/Distancias (5)
Egipto(10)/450
Turquía(11)/350
Rusia(15)/640
China(14)/470
India(13)/300

India(13)
Adyacencias/Distancias (1)
Irán(12)/300

China(14)
Adyacencias/Distancias (2)
Rusia(15)/440
Irán(12)/470

Rusia(15)
Adyacencias/Distancias (4)
Suecia(17)/640
Alemania(16)/490
China(14)/440
Irán(12)/640

Alemania(16)
Adyacencias/Distancias (5)
Francia(0)/350
Italia(9)/440
Ucrania(19)/270
Rusia(15)/490
Suecia(17)/310

Suecia(17)
Adyacencias/Distancias (3)
Francia(0)/450
Alemania(16)/310
Rusia(15)/640

Reino Unido(18)
Adyacencias/Distancias (4)
Suecia(17)/180
Francia(0)/350
México(3)/730
Canadá(1)/730

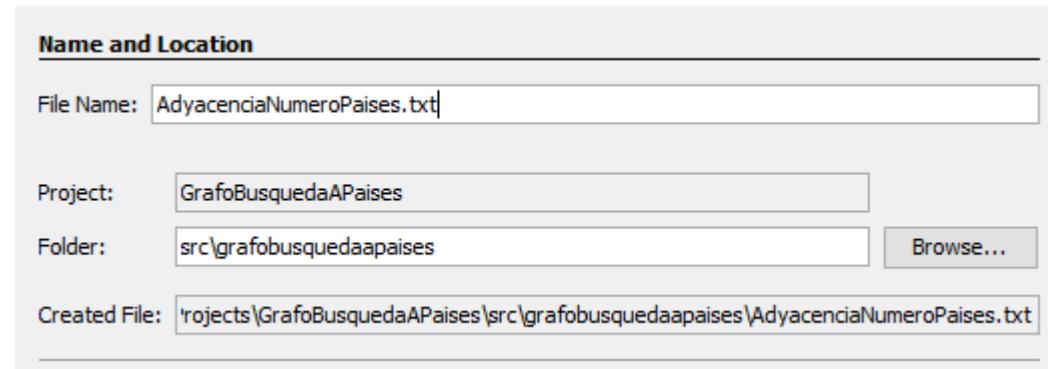
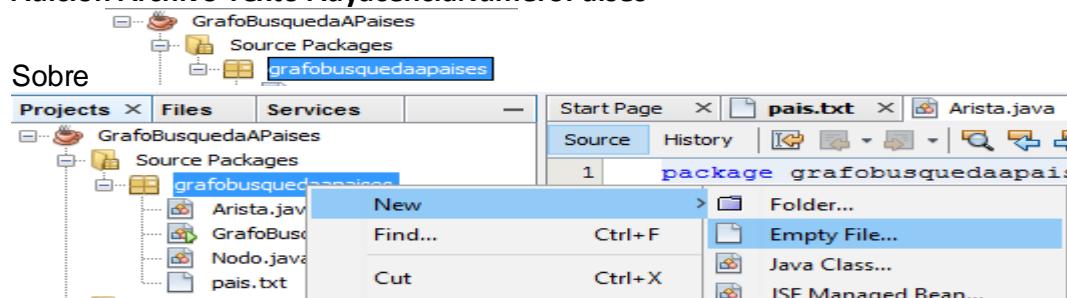
Ucrania(19)
Adyacencias/Distancias (3)
Alemania(16)/270
Italia(9)/340
Turquía(11)/200

```
1 Francia(0)
2 Adyacencias/Distancias (7)
3 Reino Unido(18)/400
4 Suecia(17)/450
5 Alemania(16)/350
6 Italia(9)/290
7 España(5)/200
8 Brasil(6)/610
9 México/810
10
11 Canadá(1)
12 Adyacencias/Distancias (2)
13 Reino Unido(18)/730
14 EEUU(2)/290
15
16 EEUU(2)
17 Adyacencias/Distancias (3)
18 Canadá(1)/290
19 México/210
20 Colombia(4)/450
```

```
1 Francia(0) ← Origen
2 Adyacencias/Distancias (7) ↑ Número de Adyacencias
3 Reino Unido(18)/400
4 Suecia(17)/450
5 Alemania(16)/350
6 Italia(9)/290
7 España(5)/200
8 Brasil(6)/610
9 México/810 ← Distancia
10
11
12 ↑ Ciudad Adyacencia
```

Este archivo permite documentar lo que a continuación serán los archivos de datos de Adyacencias como son: Número de Adyacencia por País (Archivo Texto AdyacenciaNúmeroPaises) y Distancia por Cada Adyacencia (AdyacenciaPaises).

Adición Archivo Texto AdyacenciaNumeroPaises



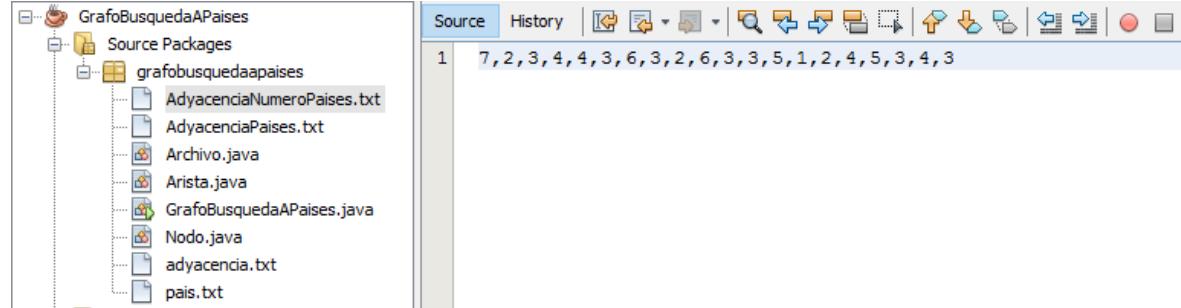
FileName: AdyacenciaNumeroPaises.txt

Finish

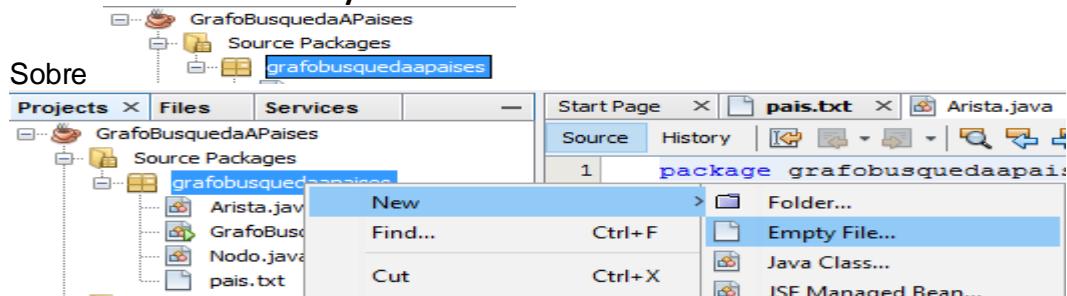
Pulsar Botón

Colocar el siguiente contenido en el archivo:

7,2,3,4,4,3,6,3,2,6,3,3,5,1,2,4,5,3,4,3



Adición Archivo Texto AdyacenciaPaises



Name and Location

File Name:

Project:

Folder:

Created File:

FileName: AdyacenciaPaises.txt

Pulsar Botón

Colocar el siguiente contenido en el archivo:

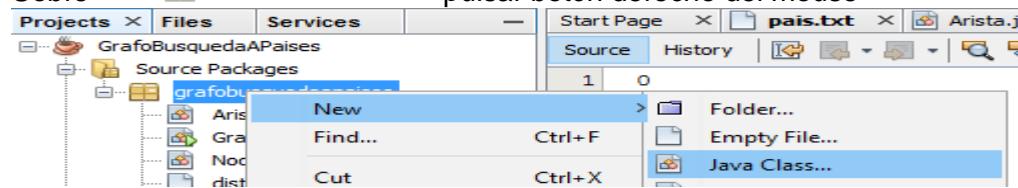
18/400,
17/450,
16/350,
9/290,
5/200,
6/610,
3/810;
18/730,
2/290;
11/290,
3/210,
4/450;
2/210,
4/290,
18/730,
0/810;
2/450,
3/290,
6/500,
7/780;
0/200,
9/300,
6/450;
4/500,
7/590,

8/510,
0/610,
5/450,
9/620;
4/780,
6/590,
7/270;
6/510,
7/270;
6/620,
5/300,
0/290,
16/440,
19/340,
10/290;
9/290,
11/180,
12/450;
10/180,
12/350,
19/200;
10/450,
11/350,
15/640,
14/470,
13/300;
12/300;
15/440,
12/470;
17/640,
16/490,
14/440,
12/640;
0/350,
9/440,
19/270,
15/490,
17/310;
0/450,
16/310,
15/640;
17/180,
0/350,
3/730,
11/730;
16/270,
9/340,
11/200;

El separador de país es el carácter ; (punto y coma) y el separado de destino/distancia es , (coma).

Adición Clase Archivo

Sobre **grafobusquedaapaises** pulsar botón derecho del mouse



Class Name: Archivo

Name and Location

Class Name:	Archivo
Project:	GrafoBusquedaAPaises
Location:	Source Packages
Package:	grafobusquedaapaises
Created File:	c:\users\NetBeansProjects\GrafoBusquedaAPaises\src\grafobusquedaapaises\Archivo.java

Pulsar Botón

Finish

The screenshot shows the NetBeans code editor with the file 'Archivo.java' open. The code is as follows:

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package grafobusquedaapaises;
7
8  /**
9   *
10  * @author Docen.informatica
11  */
12 public class Archivo {
13
14 }
```

Reemplazar contenido por:

```
package grafobusquedaapaises;
import java.util.*;
import java.io.*;
public class Archivo {
String A= new String();
int conteo=0;
String nomarch;
int numero,k,i;
public Archivo(String nomarch) {
this.nomarch=nomarch;
}
boolean bajarArchivoLinea() throws FileNotFoundException {
StringBuilder Lee=new StringBuilder();
String Directorio=System.getProperty("user.dir")+(char)
92+"src"+(char) 92+"grafobusquedaapaises"+(char) 92;
nomarch=Directorio+nomarch;
```

```

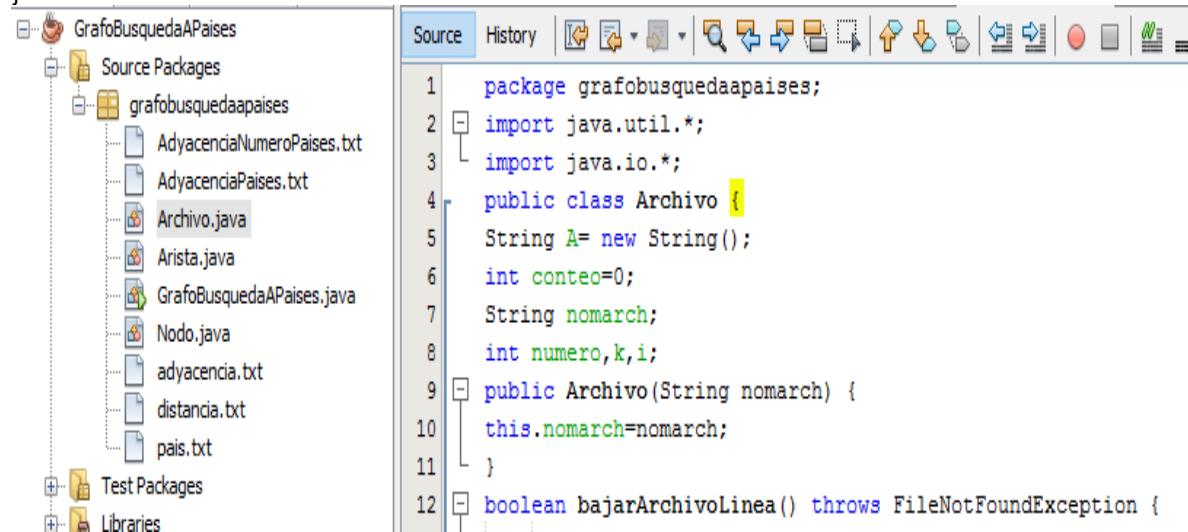
File Fi = new File(nomarch);
if (!Fi.exists())
{
return false;
}
try
{
    DataInputStream Entrada = new DataInputStream ( new
FileInputStream (Fi));
    numero=Entrada.available();
    for(k=1;k<=numero;k++)
        Lee.append((char) Entrada.read());
    }
    catch(Exception e)
    {
        System.out.println("excepcion...");
        return false;
    }
// Salto de Línea \n
    StringTokenizer registro = new
StringTokenizer(Lee.toString(),"\n");
    A="";
    while (registro.hasMoreTokens())
    {
if(conteo>0)
    A=A+ ",";
A=A+((String) registro.nextElement());
conteo++;
    }
    return true;
}
int contarArchivoLinea()
{
    return conteo;
}
String cadenaArchivoLinea()
{
    return A;
}
boolean bajarArchivoAdyacencia() throws FileNotFoundException {
    StringBuilder Lee=new StringBuilder();
    String Directorio=System.getProperty("user.dir")+(char)
92+"src"+(char) 92+"grafobusquedaapaises"+(char) 92;
    nomarch=Directorio+nomarch;
    File Fi = new File(nomarch);
    if (!Fi.exists())
    {
return false;
    }
    try
    {

```

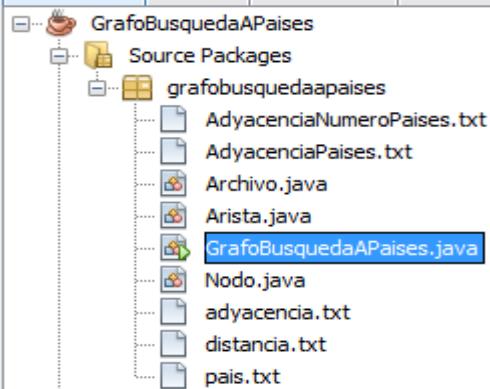
```

DataInputStream Entrada = new DataInputStream ( new
FileInputStream (Fi));
    numero=Entrada.available();
    for(k=1;k<=numero;k++)
        Lee.append((char) Entrada.read());
    }
    catch(Exception e)
    {
        System.out.println("excepcion... ");
        return false;
    }
// Salto de Línea \n
    StringTokenizer registro = new
StringTokenizer(Lee.toString(),"\n");
    A="";
    while (registro.hasMoreTokens())
    {
        A=A+((String) registro.nextElement());
    }
    return true;
}
String cadenaArchivoAdyacencia()
{
    return A;
}
}

```



Edición Programa Java – GrafoBusquedaAPaises – Seleccionar dando un click....



Reemplazar contenido con:

```
package grafobusquedapaises;
import java.io.FileNotFoundException;
import javax.swing.*;
import java.util.*;
public class GrafoBusquedaAPaises {
    //El puntaje h es la distancia entre cada pais y Francia
    public static void main(String[] args) throws FileNotFoundException
    {
        //Inicializamos el Grafo en base al mapa-mundi
        int i,j,k,m=0,p=0,n=0;
        String[] A;
        String[] B;
        String C;
        String D;
        String F;
        Archivo Arch;
        Nodo P;
        StringTokenizer registro;
        StringTokenizer campos;
        StringTokenizer variables;
        Arch=new Archivo("pais.txt");
        if(!Arch.bajarArchivoLinea())
        {
            System.out.println("No existe Archivo..."+Arch.nomarch);
            System.exit(0);
        }
        if(Arch.conteo==0)
        {
            System.out.println("No existe registros");
            System.exit(0);
        }
        C=Arch.cadenaArchivoLinea();
        A=new String[Arch.contarArchivoLinea()];
        i=-1;
        registro = new StringTokenizer(C, ",");
        while (registro.hasMoreTokens())
    }}
```

```

{
    i++;
    A[i]=((String) registro.nextElement());
}
Arch=new Archivo("distancia.txt");
if(!Arch.bajarArchivoLinea())
{
    System.out.println("No existe Archivo..."+Arch.nomarch);
    System.exit(0);
}
if(Arch.conteo==0)
    System.exit(0);
C=Arch.cadenaArchivoLinea();
B=new String[Arch.contarArchivoLinea()];
i=-1;
registro = new StringTokenizer(C, ",");
while (registro.hasMoreTokens())
{
    i++;
    B[i]=((String) registro.nextElement());
}
Nodo[] nodos=new Nodo[A.length];
Arch=new Archivo("AdyacenciaNumeroPaises.txt");
if(!Arch.bajarArchivoAdyacencia())
{
    System.out.println("No existe Archivo..."+Arch.nomarch);
    System.exit(0);
}
C=Arch.cadenaArchivoAdyacencia();
registro = new StringTokenizer(C, ",");
i=-1;
while (registro.hasMoreTokens())
{
    i++;
    D=registro.nextToken().trim();
    n=Integer.parseInt(D);
    nodos[i]=new Nodo(A[i],n);
    nodos[i].adyacencias=new Arista[n];
}
Arch=new Archivo("AdyacenciaPaises.txt");
if(!Arch.bajarArchivoAdyacencia())
{
    System.out.println("No existe Archivo..."+Arch.nomarch);
    System.exit(0);
}
C=Arch.cadenaArchivoAdyacencia();
registro = new StringTokenizer(C, ";");
i=-1;
while (registro.hasMoreTokens())
{
    i++;
}

```

```

campos=new StringTokenizer(registro.nextToken()," ");
System.out.println("Pais: "+A[i]+" Numero de Paises
Adyacentes: "+nodos[i].dimension());
k=-1;
while(campos.hasMoreTokens())
{
    D=campos.nextToken().trim(); // Determina Adyacencias
    variables=new StringTokenizer(D,"/");
    p=-1;
    while(variables.hasMoreTokens())
    {
        p++;
        j=Integer.parseInt(variables.nextToken());
        if(p==0)
        {
            m=j;
        }
        else
        {
            k++;
            if(k==0)
                System.out.print("{Pais:"+A[m]+ " Distancia:"+j);
            else
                System.out.print("}{Pais:"+A[m]+ " Distancia:"+j);
            nodos[i].adyacencias[k]=new Arista(nodos[m], j);
        }
    }
}
System.out.println("}");
}
do
{
do
{
    C=(String) JOptionPane.showInputDialog(null,"Pais Fuente: ",
    "Entrada de datos",JOptionPane.QUESTION_MESSAGE, null,
    A,A[0]);
    D=(String) JOptionPane.showInputDialog(null,"Pais Destino:",
    "Entrada de datos",JOptionPane.QUESTION_MESSAGE, null,
    A,A[0]);
    }
    while(C == null ? D == null : C.equals(D));
    i=busqueda(C,A);
    k=busqueda(D,A);
}
while(i==k);
busquedaA(nodos[i], nodos[k]);

System.out.println("=====
=====");
List<Nodo> path = printPath(nodos[k]);

```

```

Iterator<Nodo> iterator;
iterator = path.iterator();
n=0;
while (iterator.hasNext()) {
    P=iterator.next();
    if(!C.equals(P.pais))
    {
        i=busqueda(C,A);
        k=busqueda(P.pais,A);
        j=0;
        while(j<nodos[i].adyacencias.length &&
!nodos[i].adyacencias[j].destino.pais.equals(P.pais) )
        {
            j++;
        }
        System.out.println(C+" A "+nodos[k].pais+
Distancia:"+nodos[i].adyacencias[j].distancia);
        n=n+nodos[i].adyacencias[j].distancia;
    }
    C=P.pais;
}

System.out.println("=====
=====");
System.out.println("Total Distancia: "+n);
}

public static int busqueda(String H, String T[])
{
int i=0;
while(!T[i].equals(H))
    i++;
return i;
}

public static List<Nodo> printPath(Nodo target) {
List<Nodo> path = new ArrayList<Nodo>();

for (Nodo Nodo = target; Nodo != null; Nodo = Nodo.parent) {
    path.add(Nodo);
}

Collections.reverse(path);
return path;
}

public static void busquedaA(Nodo source, Nodo goal) {
Set<Nodo> explorado = new HashSet<Nodo>();
PriorityQueue<Nodo> queue = new PriorityQueue<Nodo>(20,
    new Comparator<Nodo>() {
        //Metodo sobre-escritto de comparacion
        public int compare(Nodo i, Nodo j) {
            if (i.f_scores > j.f_scores) {

```

Dar click en



Ejecución

```
Output - GrafoBusquedaPaises (run) x

... (output truncated for brevity)

Pais: Francia Numero de Paises Adyacentes: 7
(Pais:Reino Unido Distancia:400)(Pais:Suecia Distancia:450)(Pais:Alemania Distancia:350)(Pais:Italia Distancia:290)(Pais:España Distancia:200)(Pais:Brasil Distancia:610)(Pais:Méjico Distancia:810)
Pais: Canadá; Numero de Paises Adyacentes: 2
(Pais:Reino Unido Distancia:730)(Pais:EEUU Distancia:290)
Pais: EEUU Numero de Paises Adyacentes: 3
(Pais:Turquía Distancia:290)(Pais:Méjico Distancia:210)(Pais:Colombia Distancia:450)
Pais: Méjico Numero de Paises Adyacentes: 4
(Pais:EEUU Distancia:210)(Pais:Colombia Distancia:290)(Pais:Reino Unido Distancia:730)(Pais:Francia Distancia:810)
Pais: Colombia Numero de Paises Adyacentes: 4
(Pais:EEUU Distancia:450)(Pais:Méjico Distancia:290)(Pais:Brasil Distancia:600)(Pais:Argentina Distancia:780)
Pais: España Numero de Paises Adyacentes: 3
(Pais:Francia Distancia:200)(Pais:Italia Distancia:300)(Pais:Brasil Distancia:600)
Pais: Brasil Numero de Paises Adyacentes: 6
(Pais:Colombia Distancia:500)(Pais:Argentina Distancia:590)(Pais:Chile Distancia:780)(Pais:Uruguay Distancia:590)(Pais:Reino Unido Distancia:300)
Pais: Argentina Numero de Paises Adyacentes: 3
(Pais:Colombia Distancia:780)(Pais:Brasil Distancia:590)(Pais:Argentina Distancia:510)
Pais: Chile Numero de Paises Adyacentes: 5
(Pais:Brasil Distancia:510)(Pais:Argentina Distancia:270)
Pais: Italia Numero de Paises Adyacentes: 6
(Pais:Brasil Distancia:200)(Pais:España Distancia:300)(Pais:Francia Distancia:450)
Pais: Egipcio Numero de Paises Adyacentes: 3
(Pais:Italia Distancia:290)(Pais:Turquía Distancia:180)(Pais:Irán Distancia:300)
Pais: Turquía Numero de Paises Adyacentes: 3
(Pais:España Distancia:180)(Pais:Irán Distancia:350)(Pais:Ucrania Distancia:450)
Pais: Irán Numero de Paises Adyacentes: 5
(Pais:España Distancia:450)(Pais:Turquía Distancia:350)(Pais:Rusia Distancia:640)
Pais: India Numero de Paises Adyacentes: 1
(Pais:Irán Distancia:300)
Pais: China Numero de Paises Adyacentes: 2
(Pais:Rusia Distancia:440)(Pais:Irán Distancia:470)
Pais: Rusia Numero de Paises Adyacentes: 4
(Pais:Suecia Distancia:640)(Pais:Alemania Distancia:490)(Pais:China Distancia:440)(Pais:Irán Distancia:640)
Pais: Alemania Numero de Paises Adyacentes: 5
(Pais:Francia Distancia:350)(Pais:Italia Distancia:440)(Pais:Ucrania Distancia:270)(Pais:Rusia Distancia:490)(Pais:Suecia Distancia:310)
Pais: Suecia Numero de Paises Adyacentes: 3
(Pais:Francia Distancia:450)(Pais:Alemania Distancia:310)(Pais:Rusia Distancia:640)
Pais: Reino Unido Numero de Paises Adyacentes: 4
(Pais:Suecia Distancia:180)(Pais:Francia Distancia:350)(Pais:Méjico Distancia:730)(Pais:Turquía Distancia:730)
Pais: Ucrania Numero de Paises Adyacentes: 3
(Pais:Alemania Distancia:270)(Pais:Italia Distancia:340)(Pais:Turquía Distancia:200)
(Pais:Irán Distancia:300)
Pais: China Numero de Paises Adyacentes: 2
(Pais:Rusia Distancia:440)(Pais:Irán Distancia:470)
Pais: Rusia Numero de Paises Adyacentes: 4
(Pais:Suecia Distancia:640)(Pais:Alemania Distancia:490)(Pais:China Distancia:440)(Pais:Irán Distancia:640)
Pais: Alemania Numero de Paises Adyacentes: 5
(Pais:Francia Distancia:350)(Pais:Italia Distancia:440)(Pais:Ucrania Distancia:270)(Pais:Rusia Distancia:490)(Pais:Suecia Distancia:310)
Pais: Suecia Numero de Paises Adyacentes: 3
(Pais:Francia Distancia:450)(Pais:Alemania Distancia:310)(Pais:Rusia Distancia:640)
Pais: Reino Unido Numero de Paises Adyacentes: 4
(Pais:Suecia Distancia:180)(Pais:Francia Distancia:350)(Pais:Méjico Distancia:730)(Pais:Turquía Distancia:730)
Pais: Ucrania Numero de Paises Adyacentes: 3
(Pais:Alemania Distancia:270)(Pais:Italia Distancia:340)(Pais:Turquía Distancia:200)
=====
Francia A Alemania Distancia:350
Alemania A Rusia Distancia:490
Rusia A China Distancia:440
=====
Total Distancia: 1280
BUILD SUCCESSFUL (total time: 7 seconds)
```

**UNIVERSIDAD SANTO TOMAS
DUAD
FACULTAD DE CIENCIAS Y TECNOLOGIAS
PROGRAMA INGENIERIA EN INFORMATICA**

PRACTICA MATRIZ DE ADYACENCIA

**MARIO DUSTANO CONTRERAS CASTRO
DOCENTE TIEMPO COMPLETO**

ACTUALIZACION 2019

GRAFOS

Un grafo es una pareja de conjuntos $G = (V, A)$, donde V es el conjunto de vértices, y A es el conjunto de aristas, este último es un conjunto de pares de la forma (u, v) tal que , tal que. Para simplificar, notaremos la arista (a, b) como ab .



-QUE ES UNA ARISTA:

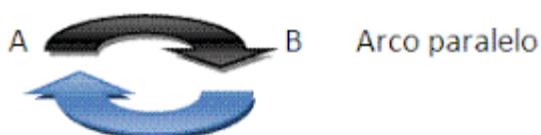
Son las líneas con las que se unen los vértices de un grafo, los vértices v_1 y v_3 son los extremos.

»Arista Adyacente: 2 aristas son adyacentes si convergen en el mismo vértice.

»Arista Paralelas: Son dos aristas conjuntas si el vértice inicial y final son el mismo.

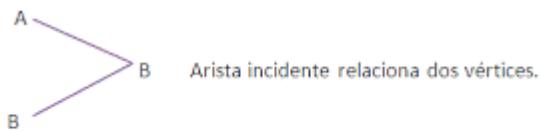
»Arista Cílicos: Es la arista que parte de un vértice para entrar en el mismo.

»Cruce: Son 2 aristas que cruzan en un mismo punto.



-QUE ES UN VERTICE:

Los vértices son los dos elementos que forman un grafo. Como ocurre con el resto de las ramas de las matemáticas, a la Teoría de Grafos no le interesa saber qué son los vértices. Diferentes situaciones en las que pueden identificarse objetos y relaciones que satisfagan la definición de grafo pueden verse como grafos y así aplicar la Teoría de Grafos en ellos.



Arista incidente relaciona dos vértices.



vértice adyacente o es adyacente.

D, C

A, C

CAMINO

Un camino en un grafo es una sucesión finita en la que aparecen alternadamente vértices y aristas de dicho grafo.

»Longitud del Camino: Está dada por número de aristas, parecido al tamaño.

»CAMINO ABIERTO:

Diferente punto de partida al de llegada, Que no llega a su principio.

»CAMINO CERRADO:

Cuando su punto de llegada es el mismo de partida.

»CAMINO SIMPLE:

No tiene aristas repetidas pero si puede tener vértices repartidos

»CAMINO ELEMENTAL:

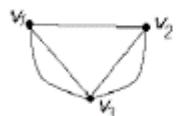
No puede repetir ni aristas ni vértices, tiene que ser abierto.

»Todo camino elemental es simple, pero no todo caminos simple es elemental.

CASOS EN GRAFOS

»Multígrafo: Cuando hay 2 o más aristas paralelas, o cuando 2 vértices están relacionados más veces consigo mismo.

G

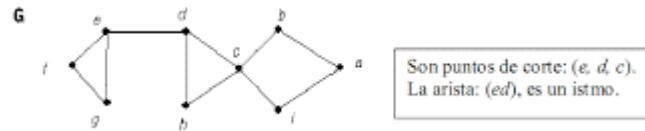


$$\begin{aligned} V &= \{v_1, v_2, v_3\} \\ E &= \{v_1v_2, v_2v_3, v_2v_3, v_1v_3, v_1v_3\} \end{aligned}$$

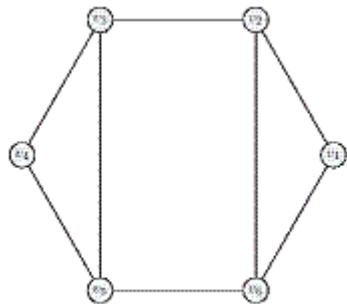
»Dígrafo: Hay un punto de origen y uno de destino final, es decir: no pueden ser $a,b = b,a$.

TIPOS DE GRAFOS

»GRAFO CIRCULO:
Camino simple y cerrado



»GRAFO CICLO:
Camino elemental y cerrado

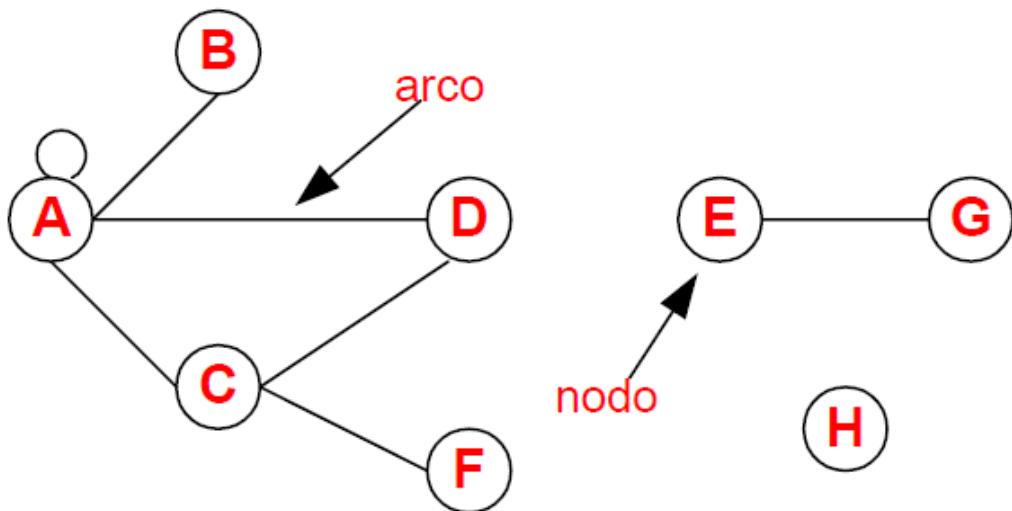


»GRAFO CADENA:
Camino elemental y abierto

GRAFOS COMO ESTRUCTURA DE DATOS

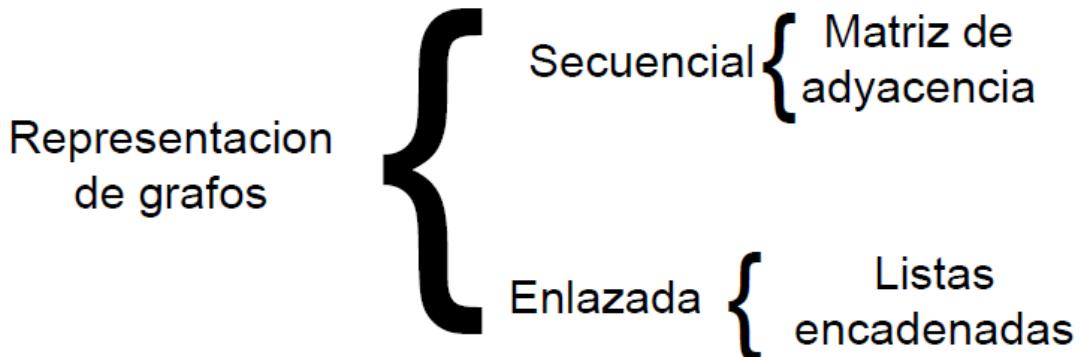
Un grafo está formado por un conjunto de nodos(o vértices) y un conjunto de arcos. Cada arco en un grafo se especifica por un par de nodos.

El conjunto de nodos es {A, B, C, D, F, G, H} y el conjunto de arcos {(A, B), (A, D), (A, C), (C, D), (C, F), (E, G), (A, A)} para el siguiente grafo:

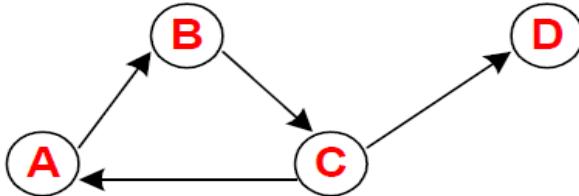


TERMINOLOGÍA

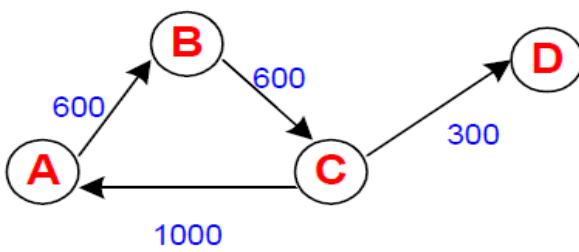
- *.-Al número de nodos del grafo se le llama orden del grafo.
- *.-Un grafo nulo es un grafo de orden 0 (cero).
- *.-Dos nodos son adyacentes si hay un arco que los une.
- *.-En un grafo dirigido, si A es adyacente de B, no necesariamente B es adyacente de A
- *.-Camino es una secuencia de uno o más arcos que conectan dos nodos.
- *.-Un grafo se denomina conectado cuando existe siempre un camino que une dos Nodos cualesquiera y desconectado en caso contrario.
- *.-Un grafo es completo cuando cada nodo está conectado con todos y cada uno de los nodos restantes.
- *.-El camino de un nodo así mismo se llama ciclo.



MATRIZ DE ADYACENCIA



	A	B	C	D
A	0	1	0	0
B	0	0	1	0
C	1	0	0	1
D	0	0	0	0

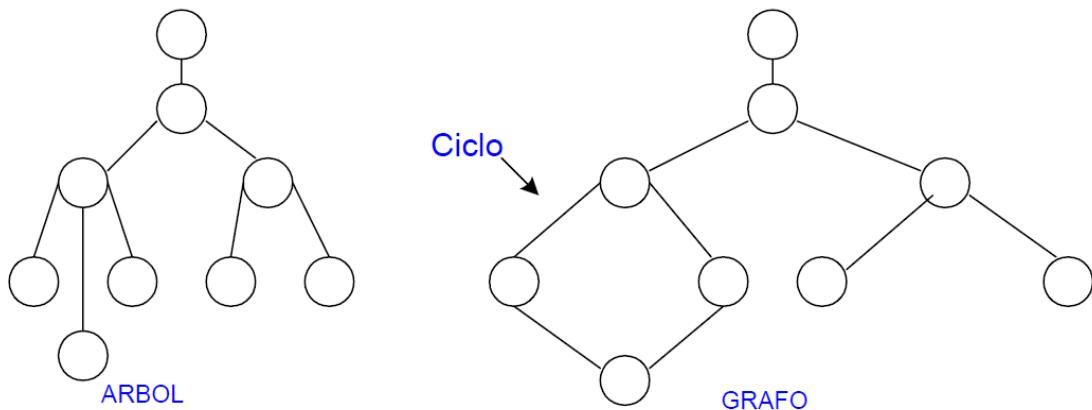


	A	B	C	D
A	0	600	100 0	0
B	600	0	600	0
C	100 0	600	0	300
D	0	0	300	0

Es una matriz cuadrada de n filas x n columnas donde, n es la máxima cantidad de nodos que tiene el grafo. Es la forma más sencilla de representar un grafo, pero a la vez, requiere más espacio de memoria que otros métodos, por lo que requiere al menos n^2 bits de memoria.

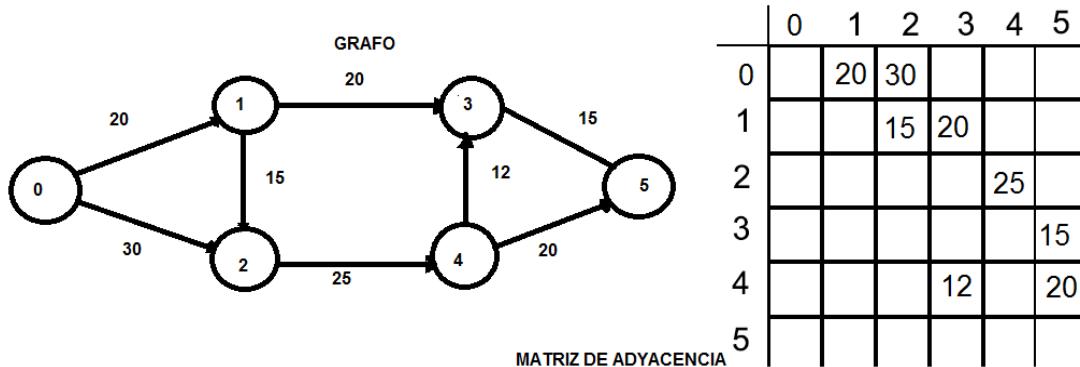
Se debe tener en cuenta:

- *.-Un grafo sin ciclos es un árbol.
- *.-El entregado de un nodo indica el número de arcos que llegan a ser nodo.
- *.-El fuera de grado de un nodo indica el número de arcos que salen de él.
- *.-Un grafo de N vértices o nodos es un árbol si cumple las siguientes condiciones
 - Tiene $N-1$ arcos
 - Existe una trayectoria entre cada par de nodos.
 - Está mínimamente conectado.



PRACTICA DE CLASE

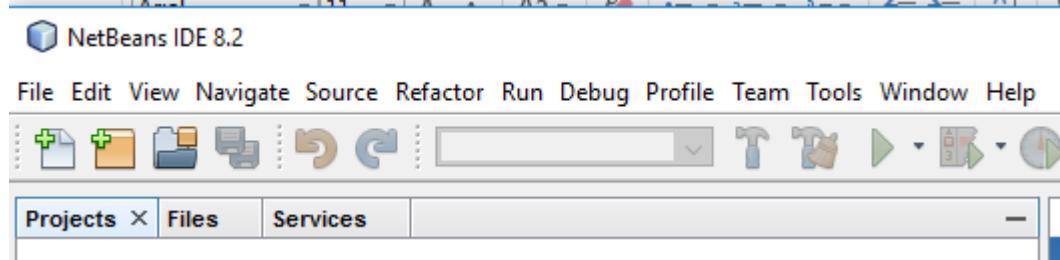
Haciendo uso del lenguaje de programación Java realizar un grafo:



PRIMER MOMENTO. INGRESO A NETBEANS

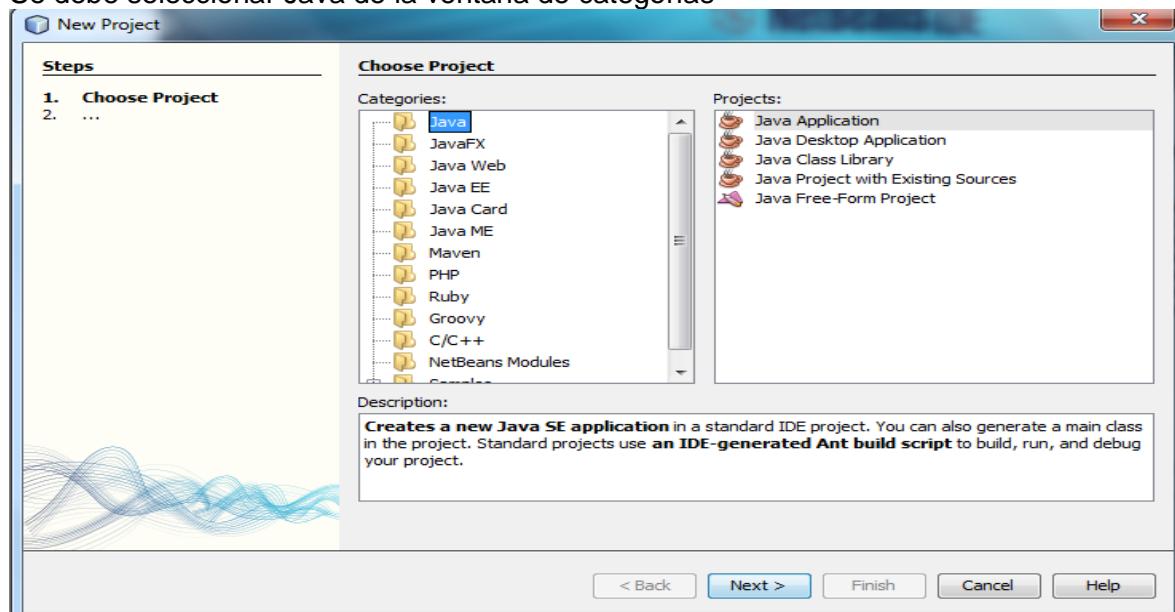
Menú Inicio>>Todas las Aplicaciones>>Netbeans>>Netbeans 8.xx

Ventana de Netbeans



Se va a crear un nuevo proyecto. Seleccionar y picar sobre el icono 

Se debe seleccionar Java de la ventana de categorías



Ahora, se selecciona Java Application de tipo de proyecto. Pulsar el Botón Next
Nombre del Proyecto: Grafos

Name and Location

Project Name:

Project Location:

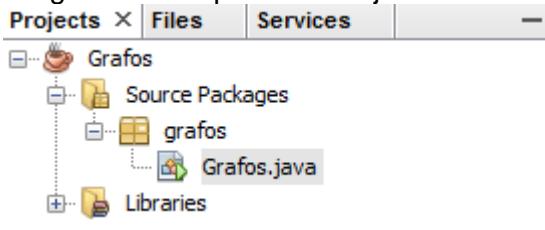
Project Folder:

Use Dedicated Folder for Storing Libraries
Libraries Folder:
Different users and projects can share the same compilation libraries (see Help for details).

Create Main Class

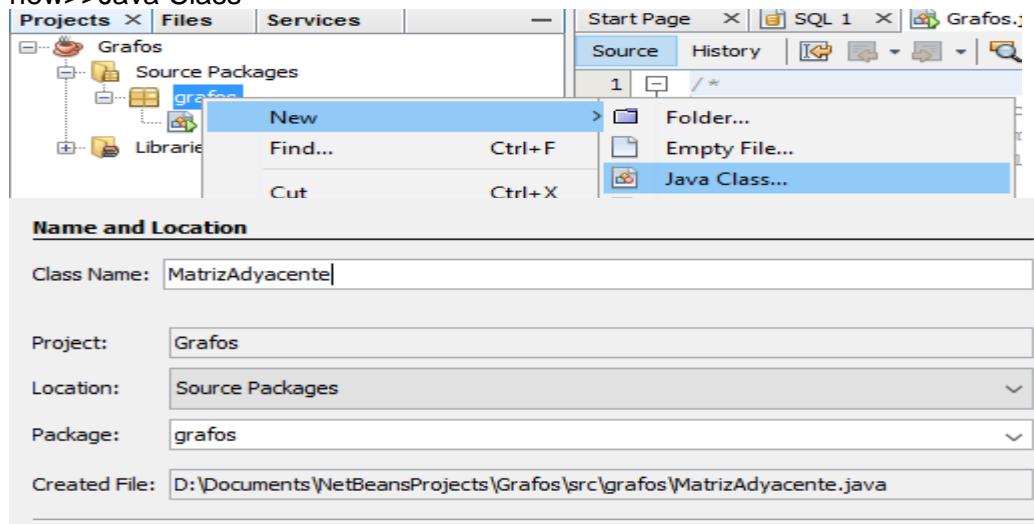
Pulsar el Botón Finish

El Netbeans genera la estructura de un proyecto con:
Paquete de Programas Fuentes(Source Packages):: lista
Programa Principal:: Grafos.java



CREACION DE LA CLASE MatrizAdyacente

Se debe posiciona sobre el paquete grafos y pulsar el botón derecho del mouse>>opción new>>Java Class



Nombre de la Clase: MatrizAdyacente

Pulsar el Botón Finish. Ahora, reemplazar el contenido del archivo por lo siguiente:

```
package grafos;
import javax.swing.JOptionPane;
public class MatrizAdyacente {
    private int i,n;
    private int[][] matriz;
    int arreglo[];
    int pos[];
    int fila, columna,suma;
    public MatrizAdyacente(int n) {
        this.n = n;
        matriz = new int[this.n][this.n];
        arreglo=new int[this.n];
        pos=new int[this.n];
        inicializa();
    }
    public void inicializa()
    {
        //se inicializa matriz en 0
        for(int i=0; i<n; i++){
            arreglo[i]=0;
            pos[i]=0;
            for(int j=0; j<n; j++){
                matriz[i][j] = 0;
            }
        }
    }
    public int valor(int i, int j){
        return matriz[i][j];
    }
    public void agregar(int i, int j, int valor){
        matriz[i][j]=valor;
    }
}
```

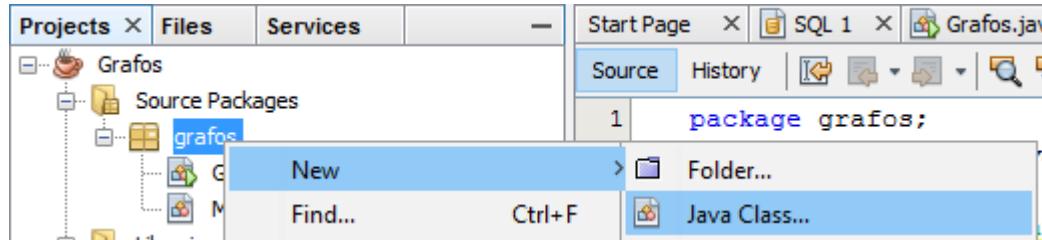
```

    }
    public void remover(int i, int j){
        if(matriz[i][j]>0)
            matriz[i][j] -= 1;
    }
    public void imprimir(){
        for(int i=0; i< n; i++){
            for(int j=0; j< n; j++){
                System.out.print( matriz[i][j] + " " );
            }
            System.out.println();
        }
    }
    public void calculo()
    {
        String resp,sitio;
        fila=0;
        suma=0;
        do
        {
            columnna=-1;
            for(i=0;i<n;i++)
            if(valor(fila,i)!=0)
            {
                columnna++;
                arreglo[columnna]=valor(fila,i);
                pos[columnna]=i;
            }
            Object recorrido[] = new Object[columnna+1];
            for(i=0;i<=columnna;i++)
            {
                recorrido[i]=(fila+1)+ " a "+(pos[i]+1)+" Carga de "+arreglo[i];
            }
            resp=(String) JOptionPane.showInputDialog(null,"Elija la Opcion", "Entrada de
            datos",JOptionPane.QUESTION_MESSAGE, null, recorrido,recorrido[0]);
            sitio=resp.substring(4,5);
            fila=Character.digit(sitio.charAt(0),10)-1;
            sitio=resp.substring(15);
            suma=suma+Integer.parseInt(sitio);
        }
        while(fila!=n-1);
        System.out.println(" Total en recorrido: "+suma);
    }
}

```

ACTUALIZACION DE LA CLASE Menu

Se debe posicionar sobre el paquete grafos y pulsar el botón derecho del mouse>>opción new>>Java Class

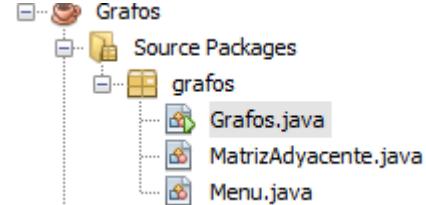


El nombre de la clase es Menu

Name and Location

Class Name:	Menu
Project:	Grafos
Location:	Source Packages
Package:	grafos
Created File:	D:\Documents\NetBeansProjects\Grafos\src\grafos\Menu.java

Pulsar el Botón Finish



Se debe reemplazar el contenido del archivo Menu.java por:

```
package grafos;
import javax.swing.*;
public class Menu {
    @SuppressWarnings("empty-statement")
    void opciones()
    {
        int n=6;
        int i,opc,info,fila, suma,columna=0;
        int arreglo[]={new int[n]};
        int pos[]={new int[n]};
        MatrizAdyacente M =new MatrizAdyacente(6);
        String resp, sitio;
        M.agregar(0,1,20);
        M.agregar(0,2,30);
        M.agregar(1,3,20);
        M.agregar(1,2,15);
        M.agregar(2,4,25);
        M.agregar(3,5,15);
        M.agregar(4,3,12);
```

```

M.agregar(4,5,20);
do
{
Object [] valores= {"1.Generación", "2.Salir"};
resp=(String) JOptionPane.showInputDialog(null,"Elija la Opcion", "Entrada de
datos",JOptionPane.QUESTION_MESSAGE, null, valores,valores[0]);
opc=Character.digit(resp.charAt(0),10);
if(opc==1)
    M.calculo();
}
while(opc!=2);
System.exit(1);
}
}

```

ACTUALIZAR LA CLASE Grafos

Seleccione la clase Grafos.java del proyecto y de doble click

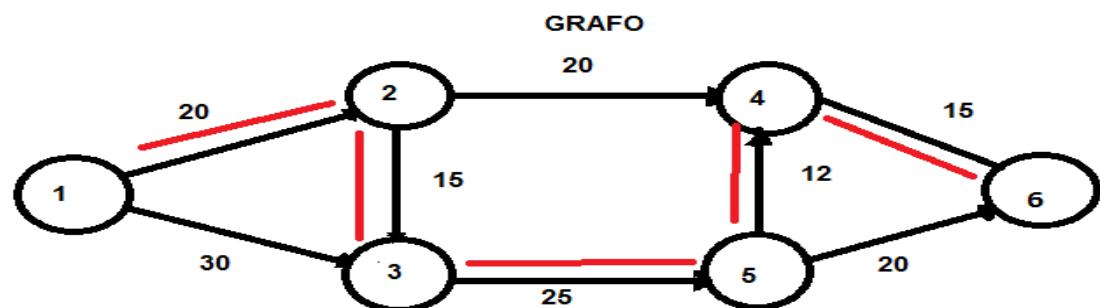
Escriba lo siguiente en // TODO code application logic here

```
Menu M=new Menu();
```

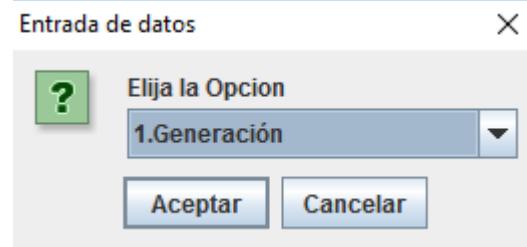
```
M.opciones();
```

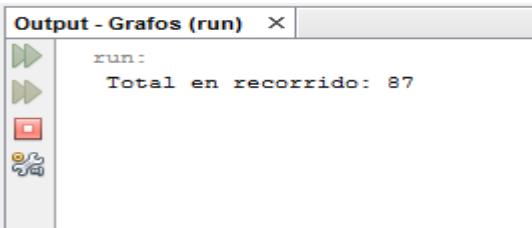
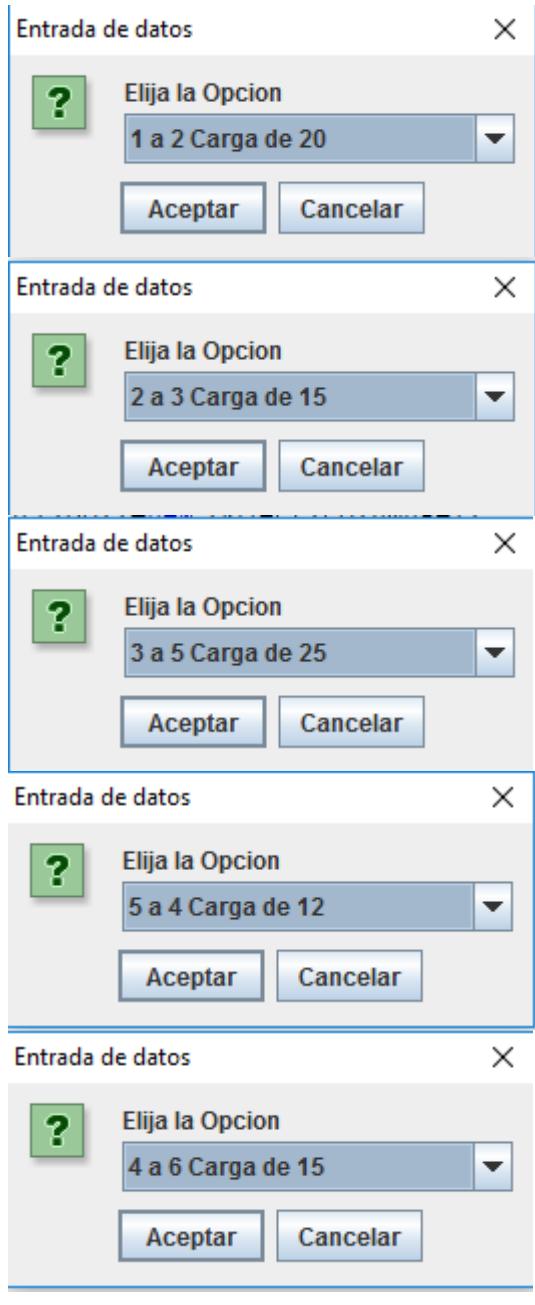
Se van a realizar la prueba acorde con:

	1	2	3	4	5	6
1		20	30			
2			15	20		
3					25	
4						15
5			12			20
6						



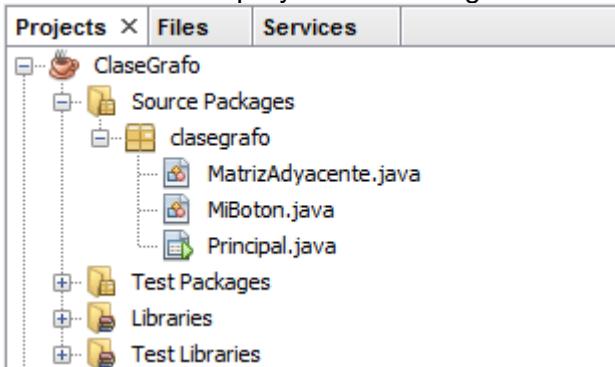
Pulsar F5 o





INTERFAZ GRAFICA

Se debe crear un proyecto con la siguiente estructura:



Clase MatrizAdyacente

El código es el siguiente:

```
import javax.swing.JOptionPane;
public class MatrizAdyacente {
    private int i,n;
    private int[][] matriz;
    int arreglo[];
    int pos[];
    int fila, columna,suma;
    public MatrizAdyacente(int n) {
        this.n = n;
        matriz = new int[this.n][this.n];
        arreglo=new int[this.n];
        pos=new int[this.n];
        inicializa();
    }
    public void inicializa()
    {
        //se inicializa matriz en 0
        for(int i=0; i< n; i++){
            arreglo[i]=0;
            pos[i]=0;
            for(int j=0; j< n; j++){
                matriz[i][j] = 0;
            }
        }
    }
    public int valor(int i, int j){
        return matriz[i][j];
    }
    public void agregar(int i, int j, int valor){
        matriz[i][j]=valor;
    }
    public void remover(int i, int j){
        if(matriz[i][j]>0)
            matriz[i][j] -= 1;
    }
}
```

```

public void imprimir(){
    for(int i=0; i< n; i++){
        for(int j=0; j< n; j++){
            System.out.print( matriz[i][j] + " " );
        }
        System.out.println();
    }
}
public void calculo()
{
String resp,sitio;
fila=0;
suma=0;
do
{
columna=-1;
for(i=0;i<n;i++)
if(valor(fila,i)!=0)
{
    columna++;
    arreglo[columna]=valor(fila,i);
    pos[columna]=i;
}
Object recorrido[] = new Object[columna+1];
for(i=0;i<=columna;i++)
{
    recorrido[i]=(fila+1)+ " a "+(pos[i]+1)+" Carga de "+arreglo[i];
}
resp=(String) JOptionPane.showInputDialog(null,"Elija la Opcion", "Entrada de
datos",JOptionPane.QUESTION_MESSAGE, null, recorrido,recorrido[0]);
sitio=resp.substring(4,5);
fila=Character.digit(sitio.charAt(0),10)-1;
sitio=resp.substring(15);
suma=suma+Integer.parseInt(sitio);
}
while(fila!=n-1);
System.out.println(" Total en recorrido: "+suma);
}
}

```

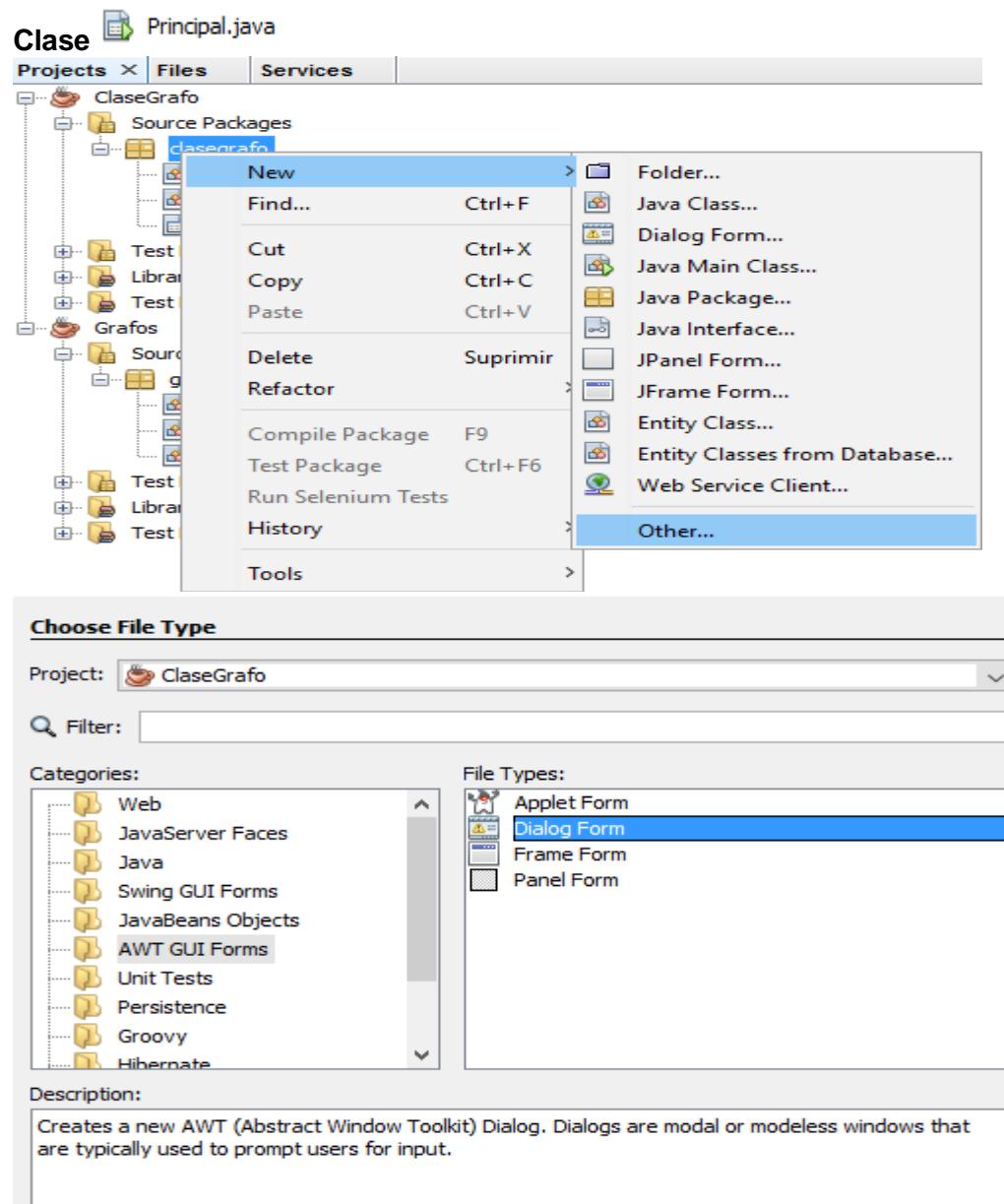
Clase MiBoton

El código es el siguiente:

```

package clasegrafo;
import javax.swing.JToggleButton;
class MiBoton extends JToggleButton{
    int i;
    int j;
    MiBoton(int fil, int col){
        i = fil;
        j = col;
    }
}

```



El código es el siguiente:

```
package clasegrafo;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JToggleButton;
public class Principal extends javax.swing.JFrame implements ActionListener {
    int size = 8;
    MiBoton [][]botones;
    MatrizAdyacente M =new MatrizAdyacente(size-2);
    int conteo=0;
    public Principal() {
        initComponents();
        int i, j;
        setLayout( new GridLayout(size, size-1) );
        botones = new MiBoton[size][size-1];
        for(i = 0; i < size; ++i )
        {
            for(j = 0; j < size-1; ++j )
            {
                botones[i][j] = new MiBoton(i,j);
                if(i<=6)
                {
                    botones[i][j] = new MiBoton(i,j);
                    botones[i][j].setText("0");
                }
                else
                switch(j)
                {
                    case 1:
                        botones[i][j].setText("CLS");
                        break;
                    case 2:
                        botones[i][j].setText("VER");
                        break;
                }
                botones[i][j].addActionListener(this);
                add(botones[i][j]);
            }
        }
        for(i=1; i < size-1; ++i )
        {
            botones[i][0].setText(" " + i );
        }
        for(i=1; i < size-1; ++i )
        {
            botones[0][i].setText(" " + i );
        }
    }
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
```

```

private void initComponents() {
    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 400, Short.MAX_VALUE)
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 300, Short.MAX_VALUE)
    );
}

    pack();
}// </editor-fold>

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
    /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and
    feel.
     * For details see
     http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
    */
    try {
        for (javax.swing.UIManager.LookAndFeelInfo info :
            javax.swing.UIManager.getInstalledLookAndFeels()) {
            if ("Nimbus".equals(info.getName())) {
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
            }
        }
    } catch (ClassNotFoundException ex) {

        java.util.logging.Logger.getLogger(Principal.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (InstantiationException ex) {

        java.util.logging.Logger.getLogger(Principal.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (IllegalAccessException ex) {

        java.util.logging.Logger.getLogger(Principal.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {
}

```

```

        java.util.logging.Logger.getLogger(Principal.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
    }
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Principal().setVisible(true);
        }
    });
}
// Variables declaration - do not modify
// End of variables declaration

@Override
public void actionPerformed(ActionEvent e) {
    MiBoton b = (MiBoton) e.getSource();
    int k,x,y,i,j;
    y=b.i-1;
    x=b.j-1;
    if(y<=5)
    {
        if(e.getSource().toString().contains("MiBoton"))
        {
            if(x!=y)
            {
                if( b.getText().equals("0") ){
                    conteo++;
                    k=(int) (Math.random()*15+1);
                    b.setText(String.valueOf(k));
                    M.agregar(y, x,k);
                }
                else{
                    b.setText("0");
                    M.agregar(y, x,0);
                }
            }
        }
    }
    else
    switch(x)
    {
        case 0:
        if(conteo>2)
        {
            M.inicializa();
            for(i = 1; i < size-1; ++i )
            for(j = 1; j < size-1; ++j )
            {
                botones[i][j].setText("0");
            }
            conteo=0;
        }
    }
}

```

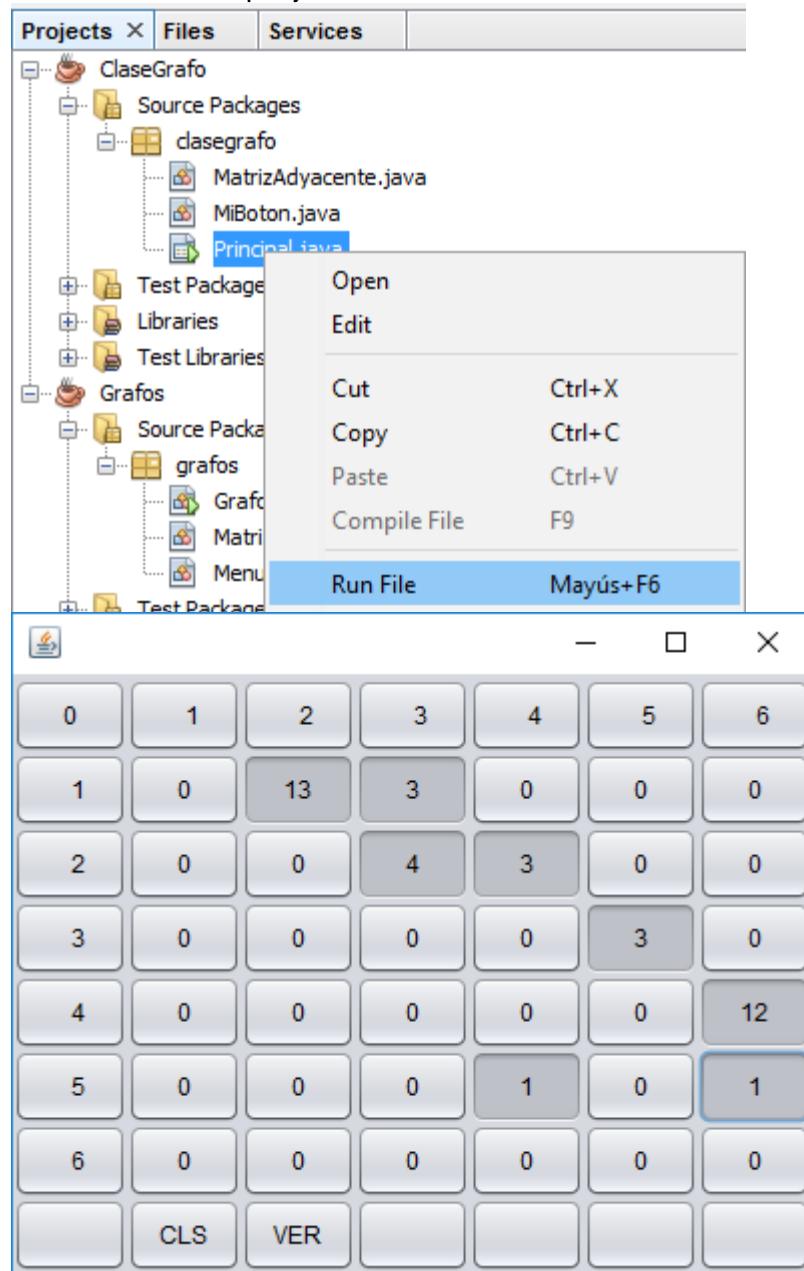
```

        }
        break;
    case 1:
        if(conteo>2)
            M.calculo();
    }
}
}

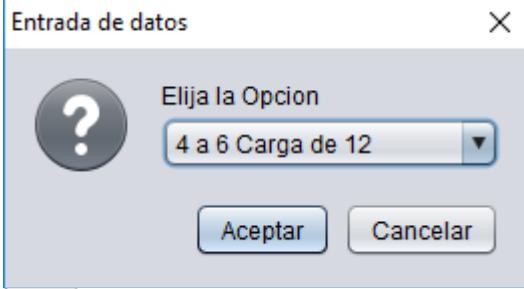
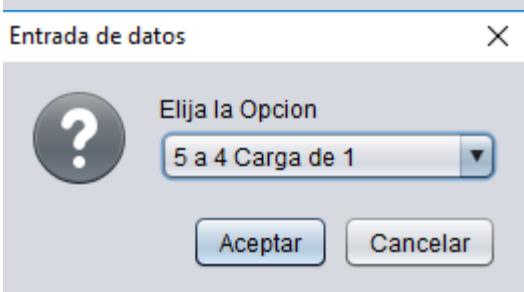
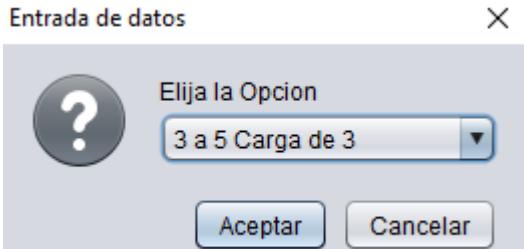
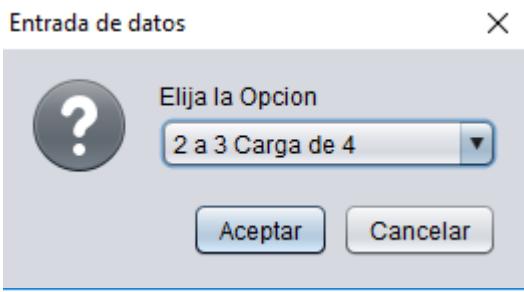
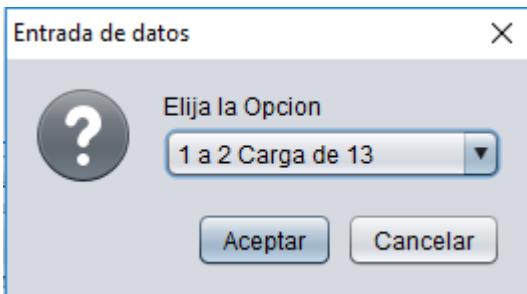
```

Ejecución.

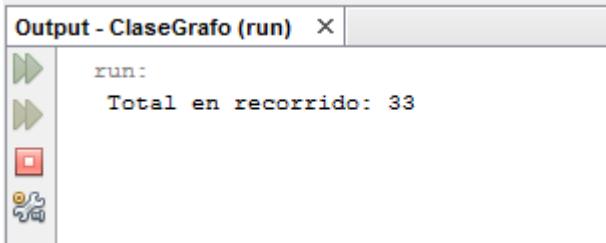
Seleccionar Principal.java>>Mouse Derecho>>Run File



Se Pulsa Botón VER



Output - ClaseGrafo (run) X



```
run:  
Total en recorrido: 33
```

The output window displays the results of a run named "ClaseGrafo". It shows a single line of text: "Total en recorrido: 33", indicating the total distance traveled during the traversal process.

**UNIVERSIDAD SANTO TOMAS
DUAD
FACULTAD DE CYT
PROGRAMA INGENIERIA EN INFORMATICA**

PRACTICA ALGORITMO DE DIJKSTRA

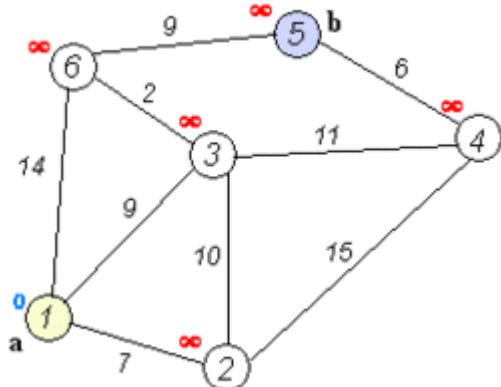
**MARIO DUSTANO CONTRERAS CASTRO
DOCENTE TIEMPO COMPLETO**

CAMINO MAS CORTO: ALGORITMO DE DIJKSTRA

<https://jariasf.wordpress.com/2012/03/19/camino-mas-corto-algoritmo-de-dijkstra/>

Descripción

El algoritmo de dijkstra determina la ruta más corta desde un nodo origen hacia los demás nodos para ello es requerido como entrada un grafo cuyas aristas posean pesos.



Como trabaja

Primero marcamos todos los vértices como no utilizados. El algoritmo parte de un vértice origen que será ingresado, a partir de ese vértice evaluaremos sus adyacentes, como dijkstra usa una técnica greedy – *La técnica greedy utiliza el principio de que para que un camino sea óptimo, todos los caminos que contiene también deben ser óptimos-* entre todos los vértices adyacentes, buscamos el que esté más cerca de nuestro punto origen, lo tomamos como punto intermedio y vemos si podemos llegar más rápido a través de este vértice a los demás. Después escogemos al siguiente más cercano (con las distancias ya actualizadas) y repetimos el proceso. Esto lo hacemos hasta que el vértice no utilizado más cercano sea nuestro destino. Al proceso de actualizar las distancias tomando como punto intermedio al nuevo vértice se le conoce como **relajación (relaxation)**.

Dijkstra usa una Cola de Prioridad o Heap, este Heap debe tener la propiedad de Min-Heap es decir cada vez que extraiga un elemento del Heap me debe devolver el de menor valor, en nuestro caso dicho valor será el peso acumulado en los nodos.

Tanto java como C++ cuentan con una cola de prioridad ambos implementan un Binary Heap aunque con un Fibonacci Heap la complejidad de dijkstra se reduce haciéndolo más eficiente, pero en un concurso más vale usar la librería que intentar programar una nueva estructura como un Fibonacci Heap, claro que particularmente uno puede investigar y programarlo para saber cómo funciona internamente.

Algoritmo en pseudocódigo

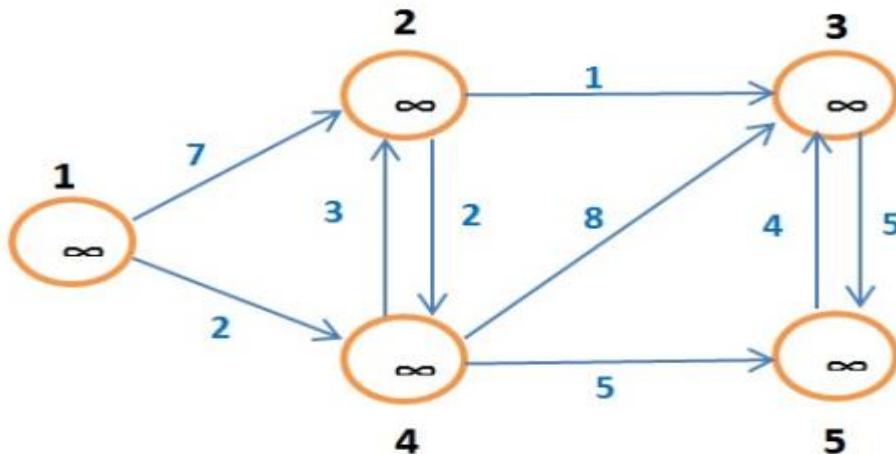
Considerar distancia[i] como la distancia más corta del vértice origen ingresado al vértice i.

```
1 método Dijkstra(Grafo,origen):
2    creamos una cola de prioridad Q
3    agregamos origen a la cola de prioridad Q
4    mientras Q no este vacío:
5        sacamos un elemento de la cola Q llamado u
6        si u ya fue visitado continuo sacando elementos de Q
7        marcamos como visitado u
8        para cada vértice v adyacente a u en el Grafo:
9            sea w el peso entre vértices (u,v)
10           si v no ah sido visitado:
11               Relajacion( u , v , w )

1 método Relajacion( actual , adyacente , peso ):
2   si distancia[ actual ] + peso < distancia[ adyacente ]
3       distancia[ adyacente ] = distancia[ actual ] + peso
4   agregamos adyacente a la cola de prioridad Q
```

Ejemplo y Código pasó a paso

Tengamos el siguiente grafo, cuyos ID están en color negro encima de cada vértice, los pesos esta en color azul y la distancia inicial en cada vértice es infinito



Algunas consideraciones para entender el código que se explicara junto con el funcionamiento del algoritmo.

```
#define MAX 10005 //maximo numero de vértices
```

```
#define Node pair< int , int > //definimos el nodo como un par( primero , segundo ) donde primero es el vértice adyacente y segundo el peso de la arista
```

```
#define INF 1<<30 //definimos un valor grande que represente la distancia infinita inicial, basta con que sea superior al máximo valor del peso en alguna de las aristas
```

Inicializamos los valores de nuestros arreglos

Vértices	1	2	3	4	5
Distancia[u]	∞	∞	∞	∞	∞
Visitado[u]	0	0	0	0	0
Previo[u]	-1	-1	-1	-1	-1

Donde:

- **Vértices:** ID de los vértices.
- **Distancia[u]:** Distancia mas corta desde vértice inicial a vértice con ID = u.
- **Visitado[u]:** 0 si el vértice con ID = u no fue visitado y 1 si ya fue visitado.
- **Previo[u]:** Almacenara el ID del vértice anterior al vértice con ID = u, me servirá para impresión del camino más corto.

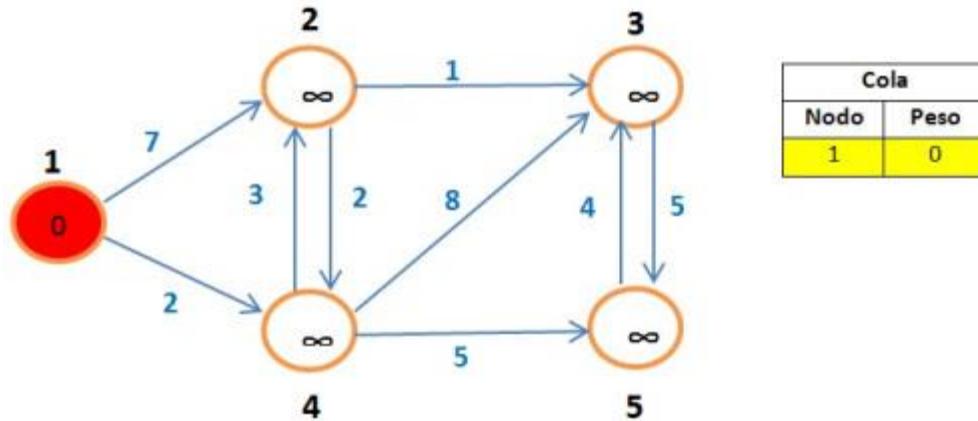
En cuanto al código los declaramos de la siguiente manera:

```
vector< Node > ady[ MAX ]; //lista de adyacencia  
int distancia[ MAX ]; //distancia[ u ] distancia de vértice inicial a vértice con ID = u  
bool visitado[ MAX ]; //para vértices visitados  
int previo[ MAX ]; //para la impresion de caminos  
priority_queue< Node , vector<Node> , cmp > Q; //priority queue propia del c++, usamos  
el comparador definido para que el de menor valor este en el tope  
int V; //numero de vértices
```

Y la función de inicialización será simplemente lo siguiente:

```
//función de inicialización  
void init(){  
    for( int i = 0 ; i <= V ; ++i ){  
        distancia[ i ] = INF; //inicializamos todas las distancias con valor infinito  
        visitado[ i ] = false; //inicializamos todos los vértices como no visitado  
        previo[ i ] = -1; //inicializamos el previo del vértice i con -1  
    }  
}
```

De acuerdo al vértice inicial que elijamos cambiara la distancia inicial, por ejemplo la ruta más corta partiendo del vértice 1 a todos los demás vértices:



El vértice 1 es visitado, la distancia de vértice 1 -> vértice 1 es 0 por estar en el mismo lugar.

```
Q.push( Node( inicial , 0 ) ); //Insertamos el vértice inicial en la Cola de Prioridad
distancia[ inicial ] = 0; //Este paso es importante, inicializamos la distancia del inicial como 0
```

Extraemos el tope de la cola de prioridad

```
while( !Q.empty() ){ //Mientras cola no este vacia
actual = Q.top().first; //Obtengo de la cola el nodo con menor peso, en un comienzo será el inicial
Q.pop(); // sacamos de la cola
```

Si el tope ya fue visitado entonces no tengo necesidad de evaluarlo, por ello continuaría extrayendo elementos de la cola:

```
if( visitado[ actual ] ) continue; //Si el vértice actual ya fue visitado sigo sacando elementos de la cola
```

En este caso al ser el tope el inicial no está visitado por lo tanto marcamos como visitado.
`visitado[actual] = true;` //Marco como visitado el vértice actual
Hasta este momento la tabla quedaría de la siguiente manera:

Vértices	1	2	3	4	5
Distancia[u]	0	∞	∞	∞	∞
Visitado[u]	1	0	0	0	0
Previo[u]	-1	-1	-1	-1	-1

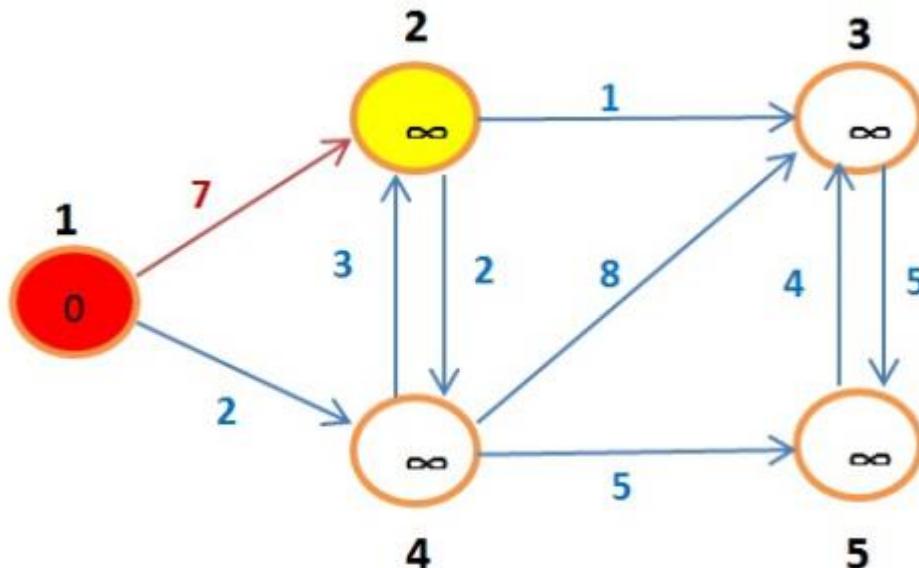
Ahora vemos sus adyacentes que no hayan sido visitados. Tendríamos 2 y 4.

```

for( int i = 0 ; i < ady[ actual ].size() ; ++i ) { //reviso sus adyacentes del vertice actual
    adyacente = ady[ actual ][ i ].first; //id del vertice adyacente
    peso = ady[ actual ][ i ].second; //peso de la arista que une actual con adyacente ( actual , adyacente )
    if( !visitado[ adyacente ] ) { //si el vertice adyacente no fue visitado
        ...
    }
}

```

Evaluamos primero para vértice 2

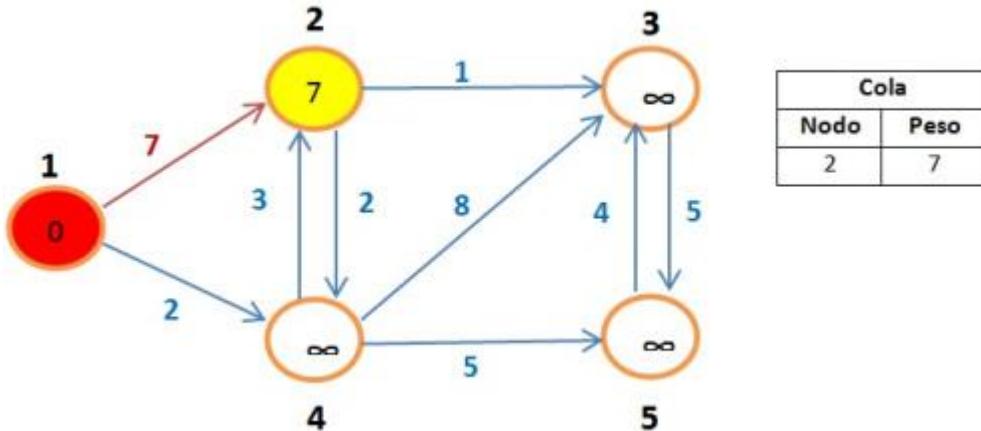


Vemos que la distancia actual desde el vértice inicial a 2 es ∞ , verifiquemos el paso de relajación:

$$\text{distancia}[1] + 7 < \text{distancia}[2] \rightarrow 0 + 7 < \infty \rightarrow 7 < \infty$$

El paso de relajación es posible realizarlo entonces actualizamos la distancia en el vértice 2 y agregando el vértice en la cola de prioridad con nueva distancia quedando:

Vértices	1	2	3	4	5
Distancia[u]	0	7	∞	∞	∞
Visitado[u]	1	0	0	0	0
Previo[u]	-1	1	-1	-1	-1



El paso de relajación se daría en la siguiente parte:

```

for( int i = 0 ; i < ady[actual].size() ; ++i ){ //reviso sus adyacentes del vértice actual
    adyacente = ady[actual][ i ].first; //id del vértice adyacente
    peso = ady[actual ][ i ].second; //peso de la arista que une actual con adyacente (
    actual , adyacente )
    if( !visitado[adyacente] ){ //si el vértice adyacente no fue visitado
        relajacion( actual , adyacente , peso ); //realizamos el paso de relajación
    }
}

```

Donde la función de relajación seria:

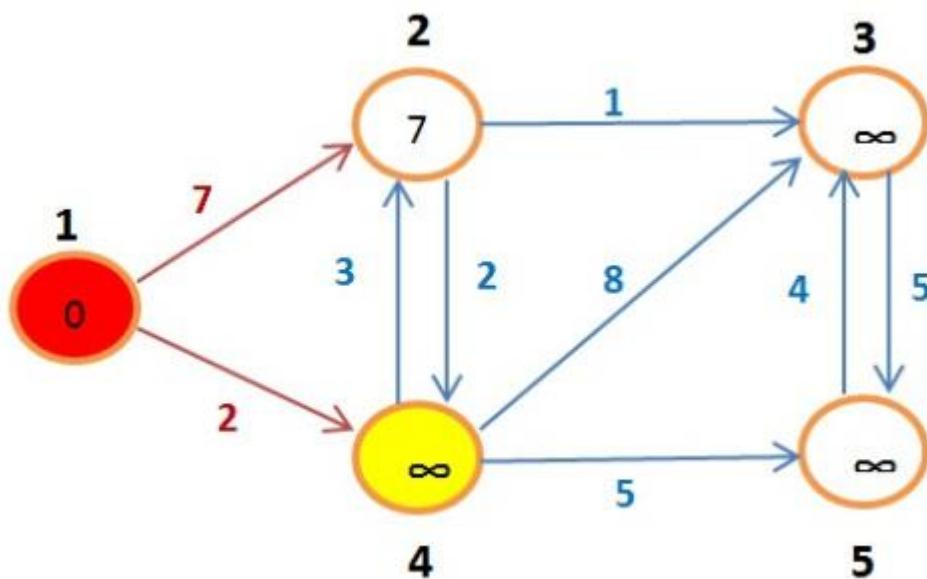
//Paso de relajacion

```

void relajacion( int actual , int adyacente , int peso ){
    //Si la distancia del origen al vértice actual + peso de su arista es menor a la distancia
    //del origen al vertice adyacente
    if( distancia[actual] + peso < distancia[adyacente] ){
        distancia[adyacente] = distancia[actual] + peso; //relajamos el vértice actualizando la
        distancia
        previo[adyacente] = actual; //a su vez actualizamos el vértice previo
        Q.push(Node( adyacente , distancia[ adyacente ] ) ); //agregamos adyacente a la cola
        de prioridad
    }
}

```

Ahora evaluamos al siguiente adyacente que es el vértice 4:



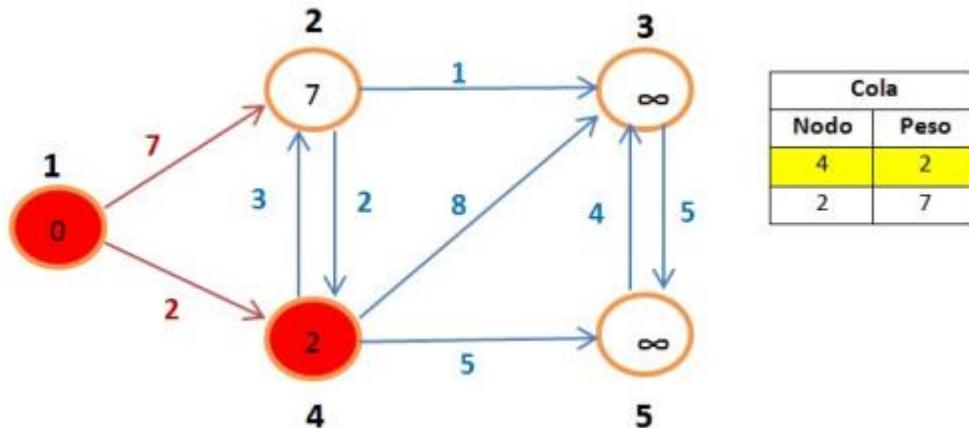
De manera similar al anterior vemos que la distancia actual desde el vértice inicial a 4 es ∞ , verifiquemos el paso de relajación:

$$\text{distancia}[1] + 2 < \text{distancia}[4] \rightarrow 0 + 2 < \infty \rightarrow 2 < \infty$$

El paso de relajación es posible realizarlo entonces actualizamos la distancia en el vértice 4 quedando:

Vértices	1	2	3	4	5
Distancia[u]	0	7	∞	2	∞
Visitado[u]	1	0	0	0	0
Previo[u]	-1	1	-1	1	-1

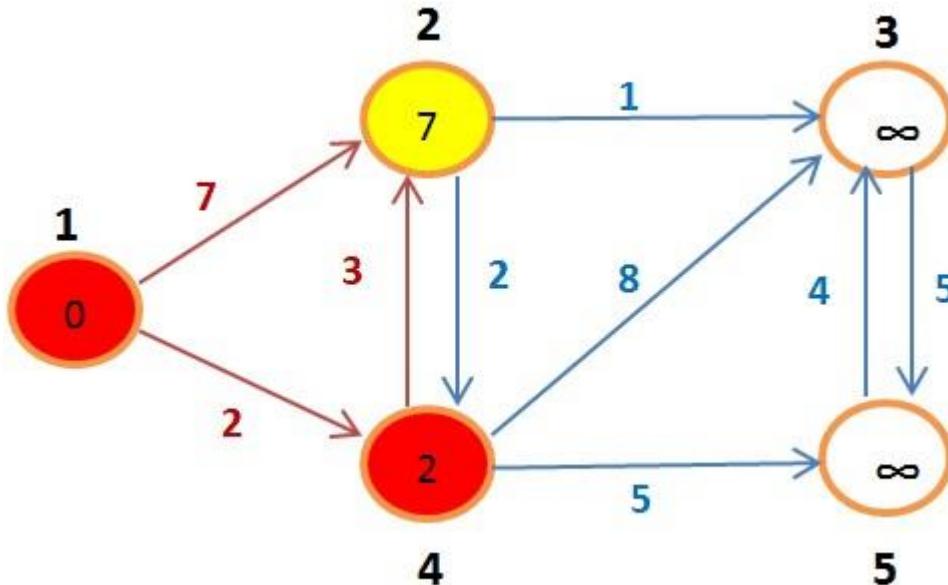
En cuanto a la cola de prioridad como tenemos un vértice con menor peso este nuevo vértice irá en el tope de la cola:



Revisamos

sus adyacentes no visitados que serían vértices 2, 3 y 5.

Empecemos por el vértice 2:



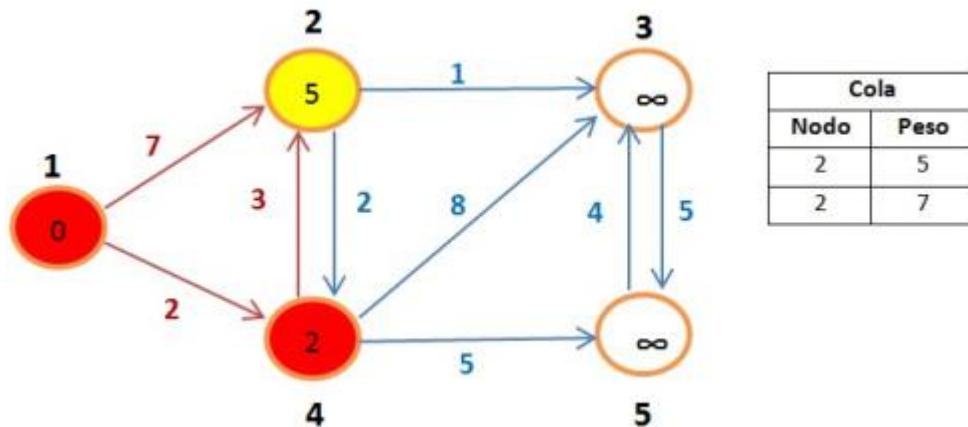
Ahora vemos que la distancia actual del vértice inicial al vértice 2 es 7, verifiquemos el paso de relajación:

$$\text{distancia}[4] + 3 < \text{distancia}[2] \rightarrow 2 + 3 < 7 \rightarrow 5 < 7$$

En esta oportunidad hemos encontrado una ruta mas corta partiendo desde el vértice inicial al vértice 2, actualizamos la distancia en el vértice 2 y actualizamos el vértice previo al actual quedando:

Vértices	1	2	3	4	5
Distancia[u]	0	5	∞	2	∞
Visitado[u]	1	0	0	1	0
Previo[u]	-1	4	-1	1	-1

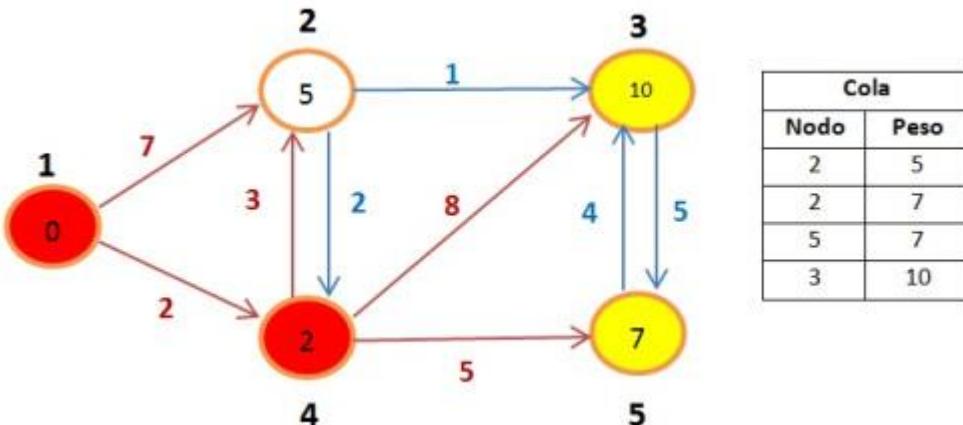
En cuanto a la cola de prioridad como tenemos un vértice con menor peso este nuevo vértice irá en el tope de la cola, podemos ver que tenemos 2 veces el mismo vértice pero como usamos una técnica greedy siempre usaremos el valor óptimo:



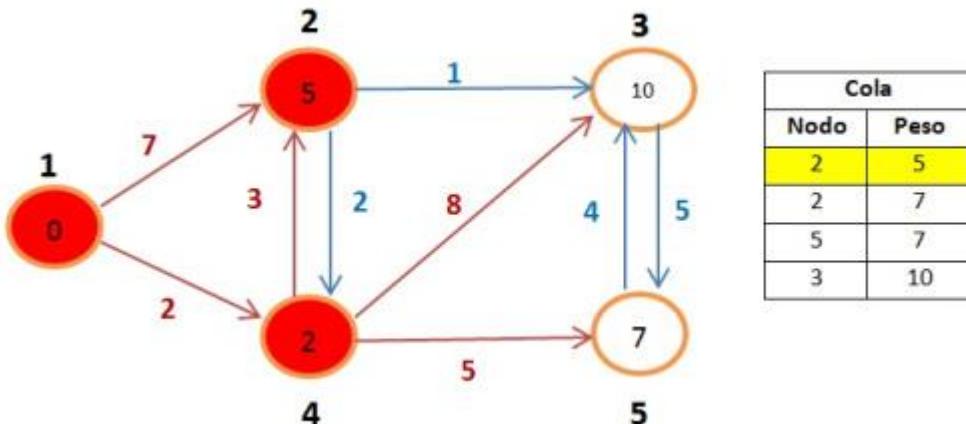
Continuamos

con los Vértices 3 y 5 como tienen valor ∞ si será posible relajarlos por lo que sería similar a los pasos iniciales solo que en los pasos iniciales distancia[1] era 0 en este caso distancia[4] es 2, quedando:

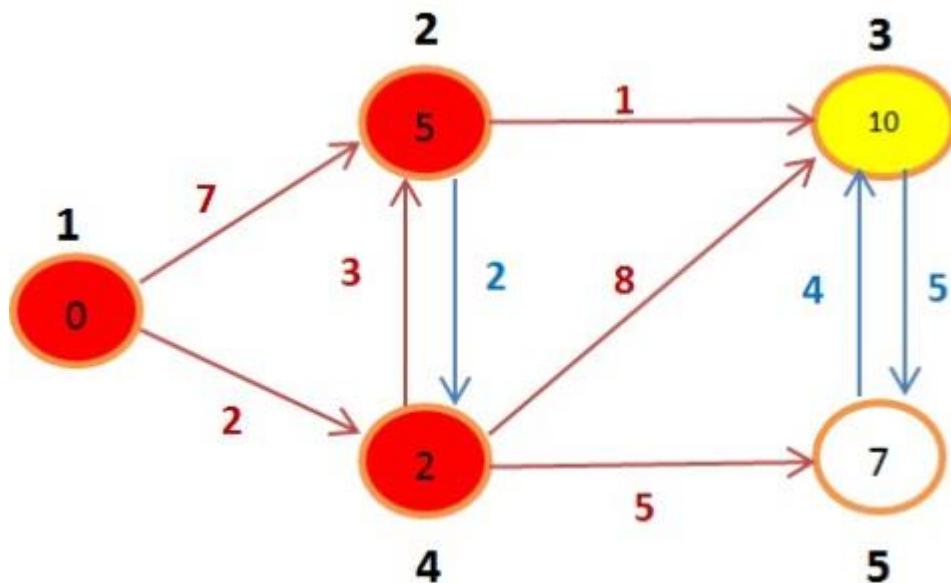
Vértices	1	2	3	4	5
Distancia[u]	0	5	10	2	7
Visitado[u]	1	0	0	1	0
Previo[u]	-1	4	4	1	4



Hemos terminado de evaluar al vértice 4, continuamos con el tope de la cola que es vértice 2, el cual marcamos como visitado.



Los adyacentes no visitados del vértice 2 son los vértices 3 y 5. Comencemos con el vértice 3

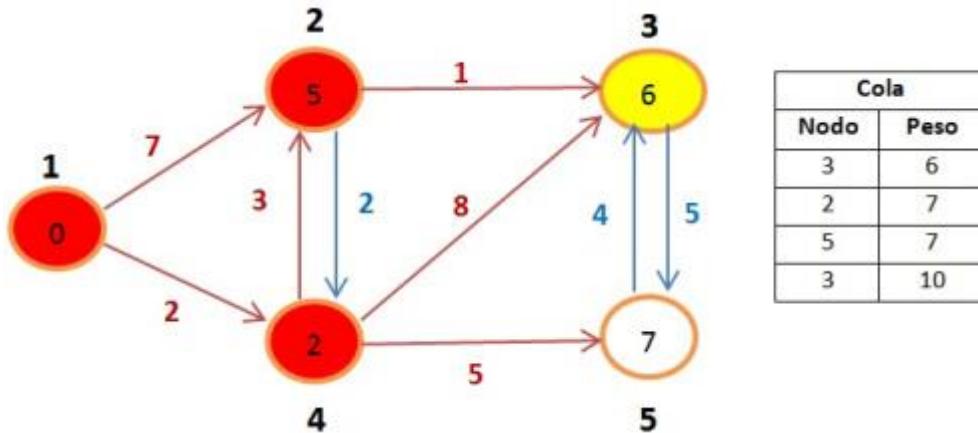


Ahora vemos que la distancia actual del vértice inicial al vértice 3 es 10, verifiquemos el paso de relajación:

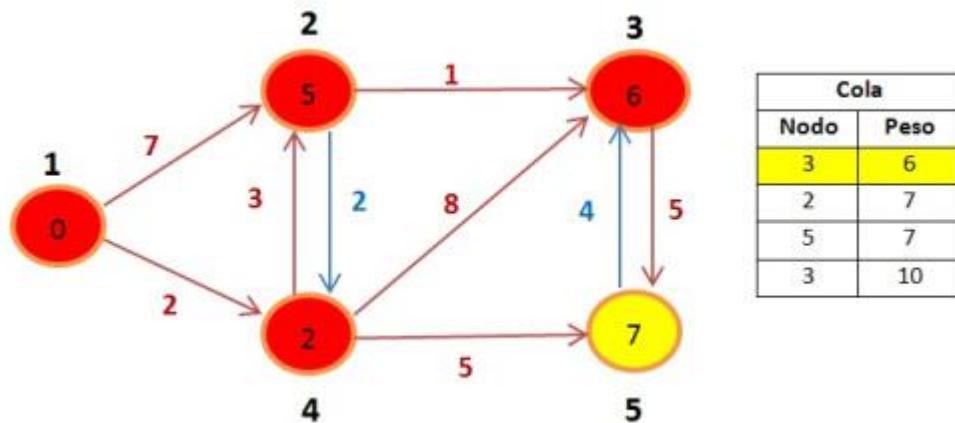
$$\text{distancia}[2] + 1 < \text{distancia}[3] \rightarrow 5 + 1 < 10 \rightarrow 6 < 10$$

En esta oportunidad hemos encontrado una ruta más corta partiendo desde el vértice inicial al vértice 3, dicha ruta sería $1 \rightarrow 4 \rightarrow 2 \rightarrow 3$ cuyo peso es 6 que es mucho menor que la ruta $1 \rightarrow 4 \rightarrow 3$ cuyo peso es 10, actualizamos la distancia en el vértice 3 quedando:

Vértices	1	2	3	4	5
Distancia[u]	0	5	6	2	7
Visitado[u]	1	1	0	1	0
Previo[u]	-1	4	2	1	4



El siguiente vértice de la cola de prioridad es el vértice 3 y su único adyacente no visitado es el vértice 5.



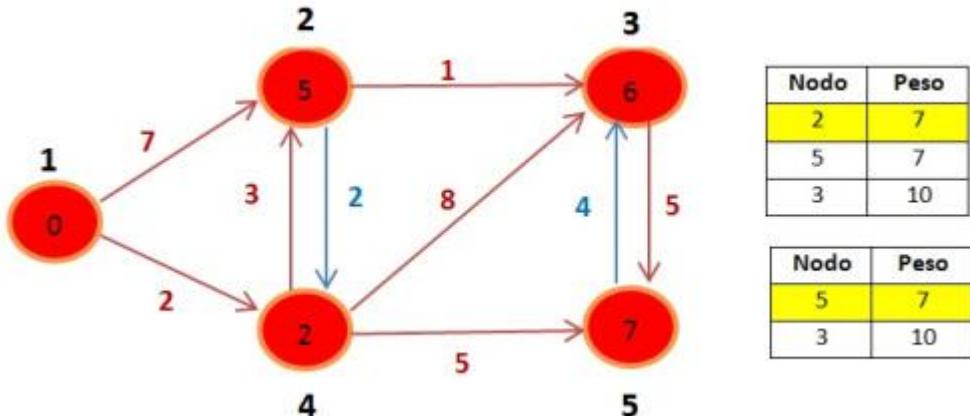
Vemos que la distancia actual del vértice inicial al vértice 5 es 7, verifiquemos el paso de relajación:

$$\text{distancia}[3] + 5 < \text{distancia}[5] \rightarrow 6 + 5 < 7 \rightarrow 11 < 7$$

En esta oportunidad se no cumple por lo que no relajamos el vértice 5, por lo que la tabla en cuanto a distancias no sufre modificaciones y no agregamos vértices a la cola:

Vértices	1	2	3	4	5
Distancia[u]	0	5	6	2	7
Visitado[u]	1	1	1	1	0
Previo[u]	-1	4	2	1	4

Ahora tocaría el vértice 2 pero como ya fue visitado seguimos extrayendo elementos de la cola, el siguiente vértice será el 5.



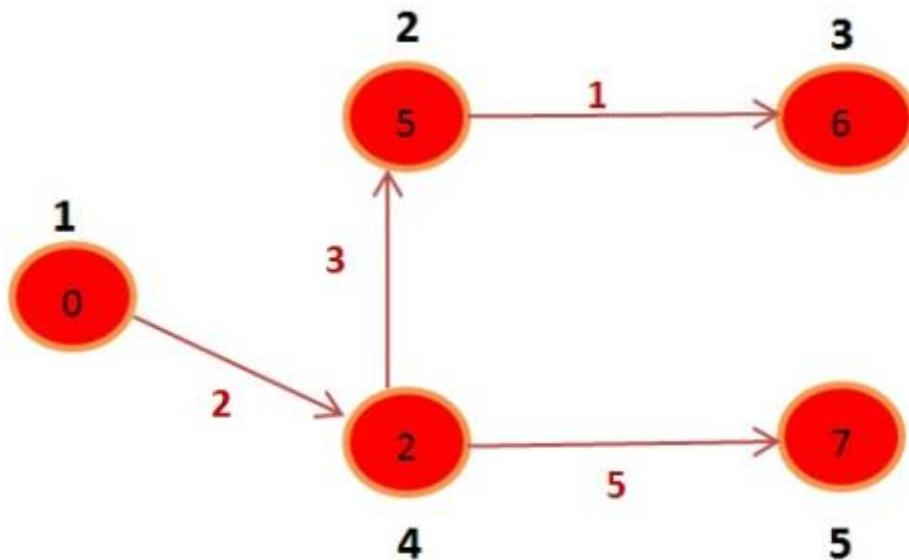
Al ser el último vértice a evaluar no posee adyacentes sin visitar por lo tanto hemos terminado el algoritmo. En el grafico anterior observamos que 2 aristas no fueron usadas para la relajación, las demás si fueron usadas. La tabla final quedaría de la siguiente manera:

Vértices	1	2	3	4	5
Distancia[u]	0	5	6	2	7
Visitado[u]	1	1	1	1	1
Previo[u]	-1	4	2	1	4

De la tabla si deseo saber la distancia más corta del vértice 1 al vértice 5, solo tengo que acceder al valor del arreglo en su índice respectivo (distancia [5]).

Shortest Path Tree

En el proceso anterior usábamos el arreglo previo[u] para almacenar el ID del vértice previo al vértice con ID = u, ello me sirve para formar el árbol de la ruta mas corta y además me sirve para imprimir caminos de la ruta más corta.



Shortest Path Tree

Impresión del camino encontrado.

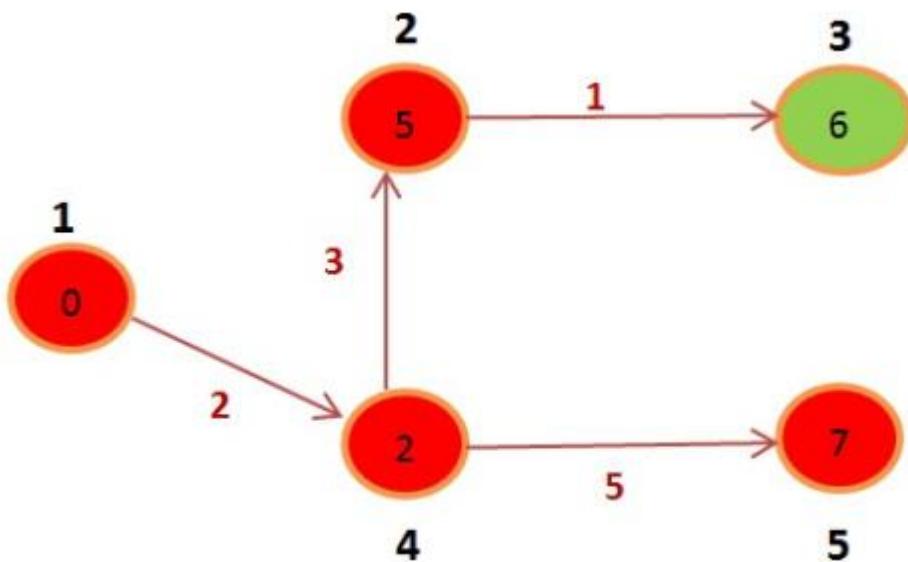
Para imprimir el camino más corto deseado usamos el arreglo previo[u], donde u tendrá el ID del vértice destino, o sea si quiero imprimir el camino más corto desde vértice 1 -> vértice 3 partiré desde previo[3] hasta el previo[1]. De manera similar a lo que se explicó en el algoritmo BFS, en este caso se realizará de manera recursiva:

//Impresión del camino más corto desde el vértice inicial y final ingresados

```
void print( int destino ){

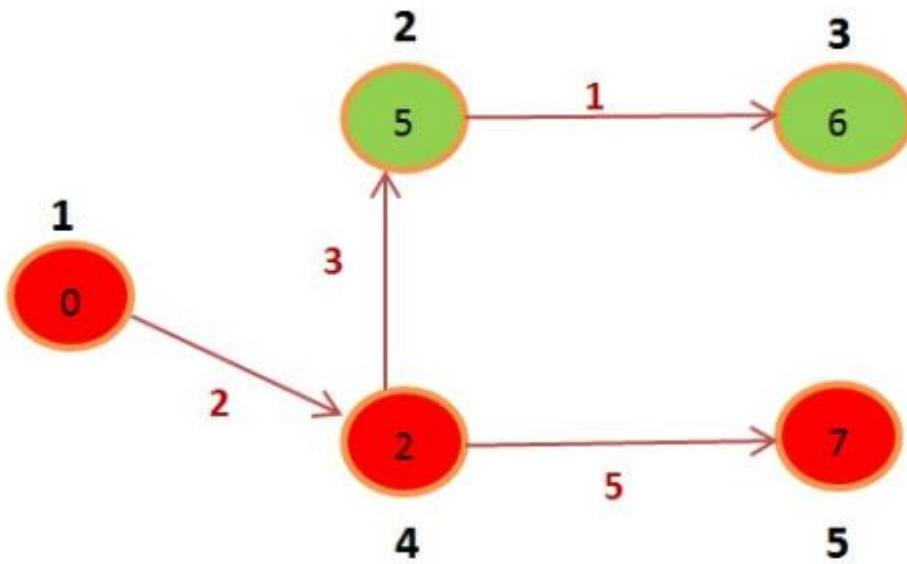
    if( previo[ destino ] != -1 ) //si aún poseo un vértice previo
        print( previo[ destino ] ); //recursivamente sigo explorando
    printf("%d ", destino ); //terminada la recursión imprimo los vértices recorridos
}
```

Veamos gráficamente el funcionamiento, desde el grafo comenzamos en 3



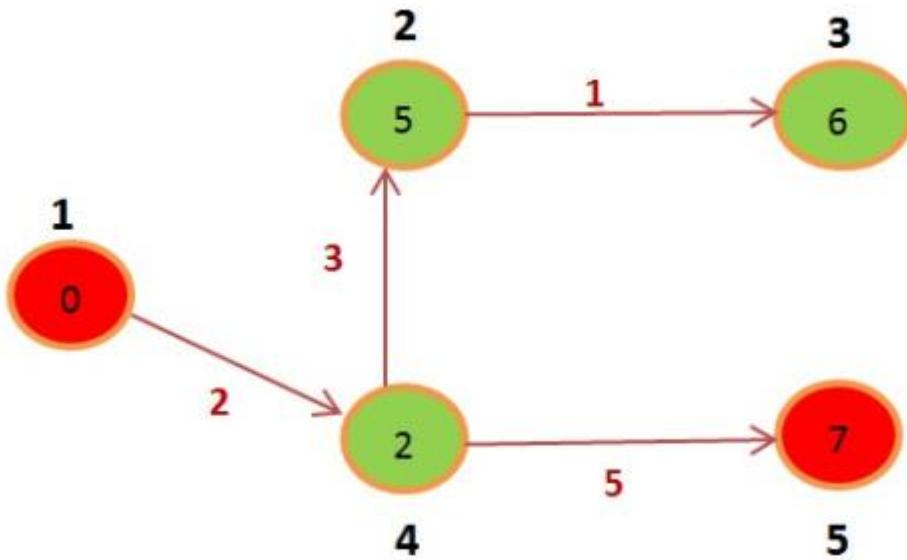
Vértices	1	2	3	4	5
Previo[u]	-1	4	2	1	4

El previo de 3 es el vértice 2, por lo tanto ahora evalúo 2:



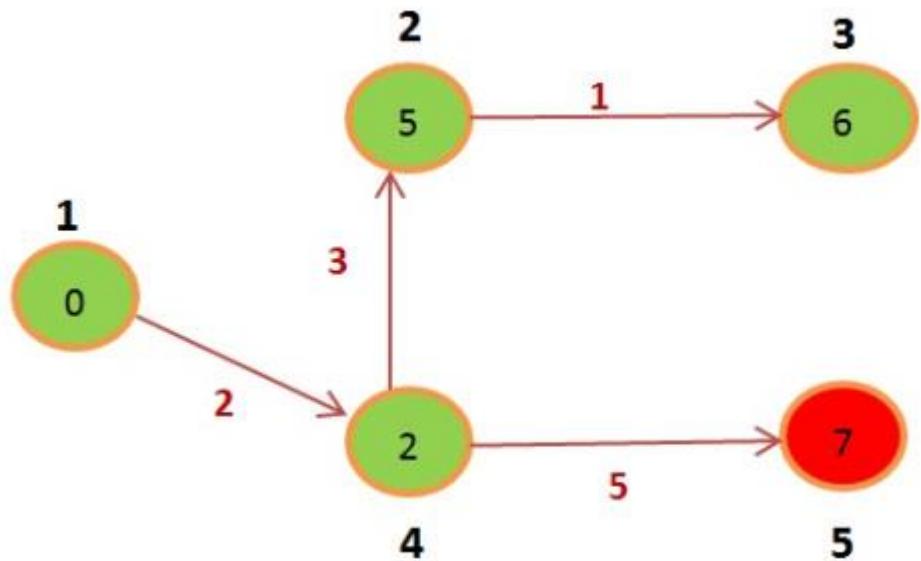
Vértices	1	2	3	4	5
Previo[u]	-1	4	2	1	4

Ahora el previo de 2 es el vértice 4:



Vértices	1	2	3	4	5
Previo[u]	-1	4	2	1	4

El previo de 4 es el vértice inicial 1



Vértices	1	2	3	4	5
Previo[u]	-1	4	2	1	4

Como el previo de 1 es -1 terminamos el recorrido, ahora en el retorno de las llamadas recursivas imprimo el camino: 1 4 2 3

Videos:

<https://www.youtube.com/watch?v=skF82H7dt1s>

<https://www.youtube.com/watch?v=4I7W5WUQQQI>

**UNIVERSIDAD SANTO TOMAS
DUAD
FACULTAD DE CYT
PROGRAMA INGENIERIA EN INFORMATICA**

PRACTICA ALGORITMO DE BELLMAN-FORD

**MARIO DUSTANO CONTRERAS CASTRO
DOCENTE TIEMPO COMPLETO**

Algoritmo de Bellman-Ford

Tomado de: <https://jariasf.wordpress.com/2013/01/01/camino-mas-corto-algoritmo-de-bellman-ford/>

Introducción

El algoritmo de Bellman-Ford (algoritmo de Bell-End-Ford) genera el camino más corto en un grafo dirigido ponderado (en el que el peso de alguna de las aristas puede ser negativo). El algoritmo de Dijkstra resuelve este mismo problema en un tiempo menor, pero requiere que los pesos de las aristas no sean negativos, salvo que el grafo sea dirigido y sin ciclos. Por lo que el Algoritmo Bellman-Ford normalmente se utiliza cuando hay aristas con peso negativo. Este algoritmo fue desarrollado por Richard Bellman, Samuel End y Lester Ford.

Según Robert Sedgewick, “Los pesos negativos no son simplemente una curiosidad matemática; [...] surgen de una forma natural en la reducción a problemas de caminos más cortos”, y son un ejemplo de una reducción del problema del camino hamiltoniano que es NP-completo hasta el problema de caminos más cortos con pesos generales. Si un grafo contiene un ciclo de coste total negativo entonces este grafo no tiene solución. El algoritmo es capaz de detectar este caso.

Si el grafo contiene un ciclo de coste negativo, el algoritmo lo detectará, pero no encontrará el camino más corto que no repite ningún vértice. La complejidad de este problema es al menos la del problema del camino más largo de complejidad NP-Completo.

El Algoritmo de Bellman-Ford es, en su estructura básica, simplemente relaja todas las aristas, y lo hace $|V|-1$ veces, siendo $|V|$ el número de vértices en el grafo. Las repeticiones permiten a las distancias mínimas recorrer el árbol, ya que en la ausencia de ciclos negativos, el camino más corto solo visita cada vértice una vez. A diferencia de la solución voraz, la cual depende de la suposición de que los pesos sean positivos, esta solución se aproxima más al caso general.

Descripción

El algoritmo de Bellman-Ford determina la ruta más corta desde un nodo origen hacia los demás nodos para ello es requerido como entrada un grafo cuyas aristas posean pesos. La diferencia de este algoritmo con los demás es que los pesos pueden tener valores negativos ya que Bellman-Ford me permite detectar la existencia de un ciclo negativo.

Como trabaja

El algoritmo parte de un vértice origen que será ingresado, a diferencia de Dijkstra que utiliza una técnica voraz para seleccionar vértices de menor peso y actualizar sus distancias mediante el paso de relajación, Bellman-Ford simplemente relaja todas las aristas y lo hace $|V| - 1$ veces, siendo $|V|$ el número de vértices del grafo.

Para la detección de ciclos negativos realizamos el paso de relajación una vez más y si se obtuvieron mejores resultados es porque existe un ciclo negativo, para verificar porque tenemos un ciclo podemos seguir relajando las veces que queramos y seguiremos obteniendo mejores resultados.

Algoritmo en pseudocódigo

Considerar distancia[i] como la distancia más corta del vértice origen ingresado al vértice i y |V| el número de vértices del grafo.

método BellmanFord(Grafo,origen):

inicializamos las distancias con un valor grande

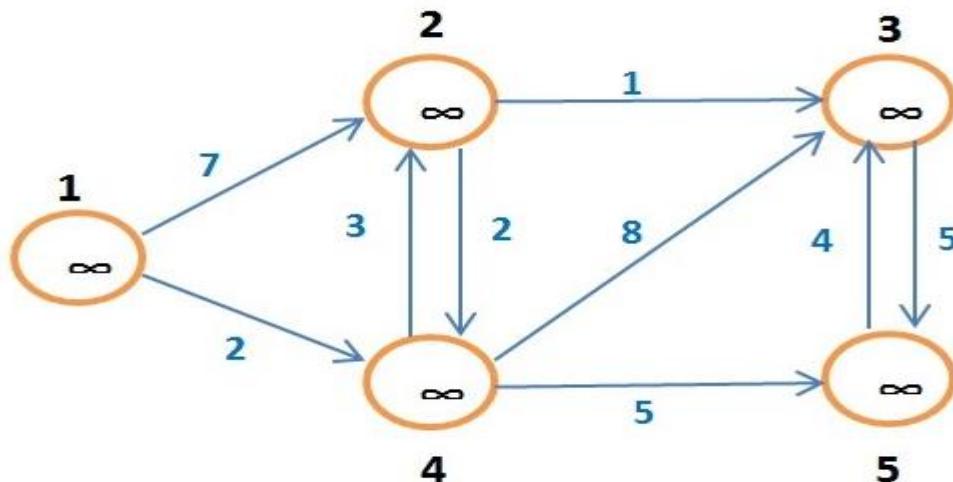
```
distancia[ origen ] = 0
para i = 1 hasta |V| - 1:
    para cada arista E del Grafo:
        sea ( u , v ) vértices que unen la arista E
        sea w el peso entre vértices ( u , v )
        Relajacion( u , v , w )
    para cada arista E del Grafo:
        sea ( u , v ) vértices que unen la arista E
        sea w el peso entre vértices ( u , v )
        si Relajacion( u , v , w )
            Imprimir "Existe ciclo negativo"
        Terminar Algoritmo
```

Relajacion(actual , adyacente , peso):

```
si distancia[ actual ] + peso < distancia[ adyacente ]
    distancia[ adyacente ] = distancia[ actual ] + peso
```

Ejemplo y Código pasó a paso

Tengamos el siguiente grafo, cuyos ID están en color negro encima de cada vértice, los pesos están en color azul y la distancia inicial en cada vértice es infinito



Algunas consideraciones para entender el código que se explicara junto con el funcionamiento del algoritmo.

```
#define MAX 105 //maximo numero de vértices
#define Node pair< int , int > //definimos el nodo como un par( first , second ) donde first es el vertice adyacente y second el peso de la arista
#define INF 1<<30 //definimos un valor grande que represente la distancia infinita inicial, basta con que sea superior al maximo valor del peso en alguna de las aristas
```

Inicializamos los valores de nuestros arreglos:

Vértices	1	2	3	4	5
Distancia[u]	∞	∞	∞	∞	∞
Previo[u]	-1	-1	-1	-1	-1

Dónde:

Vértices: ID de los vértices.

Distancia[u]: Distancia más corta desde vértice inicial a vértice con ID = u.

Previo[u]: Almacenara el ID del vértice anterior al vértice con ID = u, me servirá para impresión del camino más corto.

En cuanto al código los declararemos de la siguiente manera:

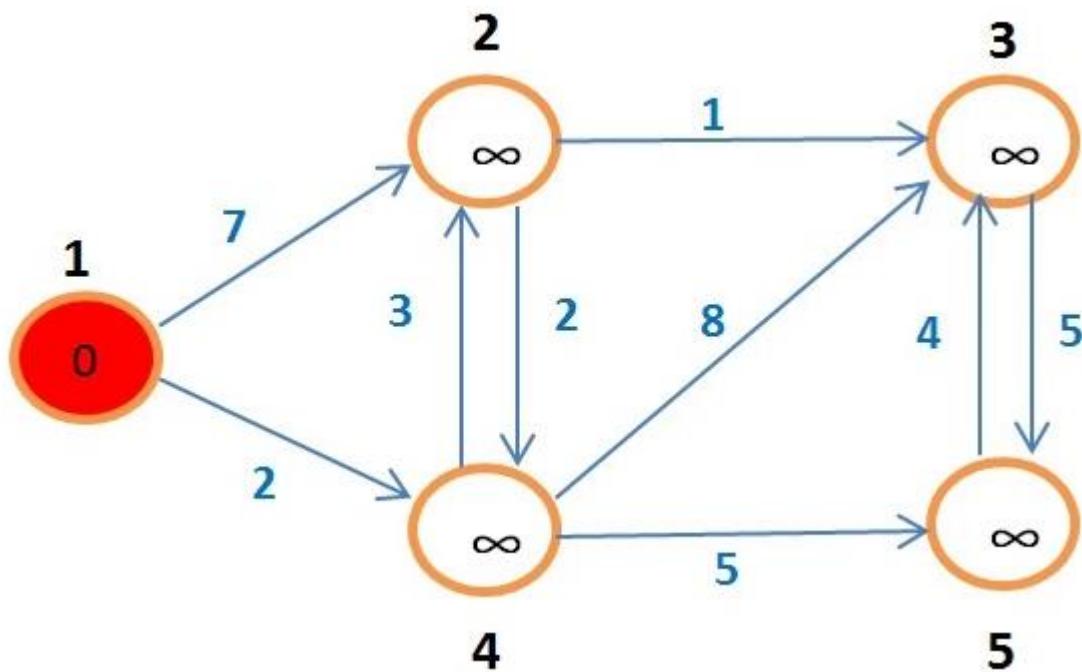
```
vector< Node > ady[ MAX ]; //lista de adyacencia
int distancia[ MAX ]; //distancia[ u ] distancia de vértice inicial a vértice con ID = u
int previo[ MAX ]; //para la impresión de caminos
int V; //número de vértices
```

Y la función de inicialización será simplemente lo siguiente:

//función de inicialización

```
void init(){
    for( int i = 0 ; i <= V ; ++i ){
        distancia[ i ] = INF; //inicializamos todas las distancias con valor infinito
        previo[ i ] = -1; //inicializamos el previo del vértice i con -1
    }
}
```

De acuerdo al vértice inicial que elijamos cambiara la distancia inicial, por ejemplo la ruta más corta partiendo del vértice 1 a todos los demás vértices.



Inicialmente la distancia de vértice 1 -> vértice 1 es 0 por estar en el mismo lugar.
`distancia[inicial] = 0;` //Este paso es importante, inicializamos la distancia del inicial como 0

Hasta este momento la tabla quedaría de la siguiente manera:

Vértices	1	2	3	4	5
Distancia[u]	0	∞	∞	∞	∞
Previo[u]	-1	-1	-1	-1	-1

Ahora según Bellman-Ford debemos realizar el paso de relajación $|V| - 1 = 5 - 1 = 4$ veces para cada arista:

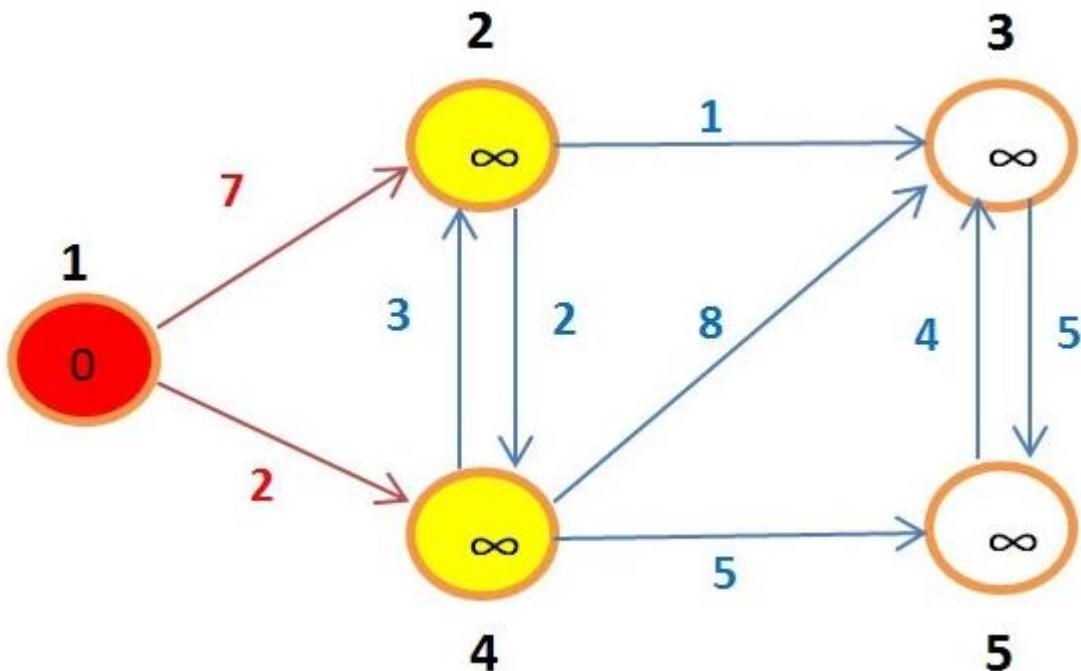
```
for( int i = 1 ; i <= V - 1 ; ++i ){ //Iteraremos |V| - 1 veces
```

...

}

Primera Iteración

Empezamos con las aristas que parten del vértice 1:

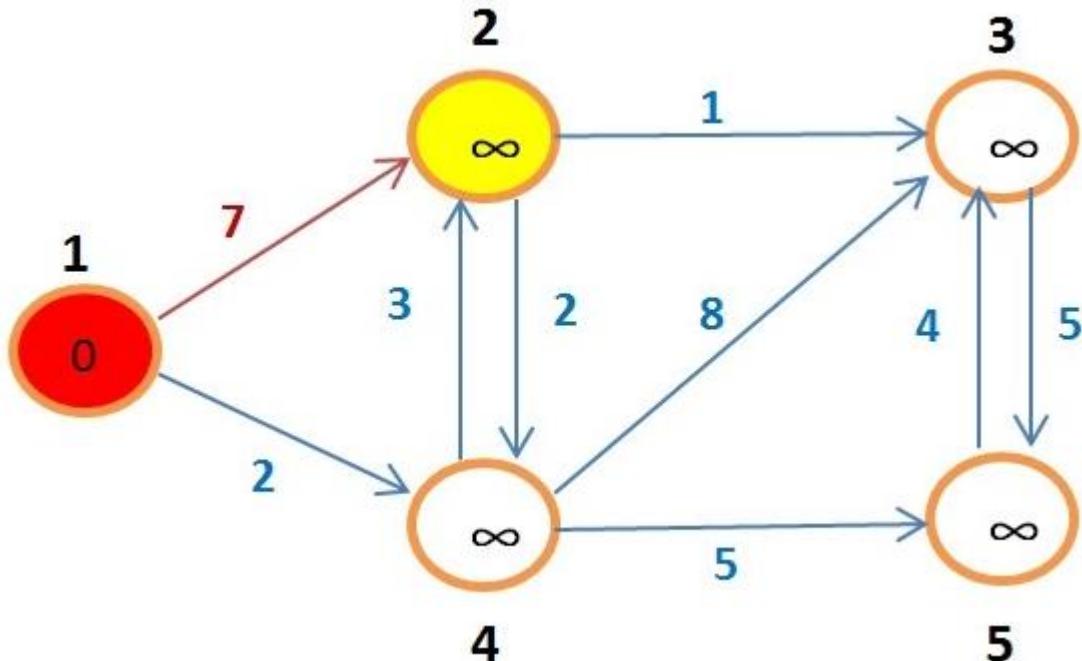


Como podemos observar tenemos 2 aristas, la que une vértice 1 con vértice 2 y vértice 1 con vértice 4. Ello en código:

```
for( int actual = 1 ; actual <= V ; ++actual ){ //Estos dos for = O(E)
    for( int j = 0 ; j < ady[ actual ].size() ; ++j ){ //reviso sus adyacentes del vertice actual
        int adyacente = ady[ actual ][ j ].v; //id del vertice adyacente
        int peso = ady[ actual ][ j ].w; //peso de la arista que une actual con adyacente (actual , adyacente )
        ...
    }
}
```

Las aristas de acuerdo al código serían de la forma $e = (\text{actual}, \text{adyacente}, \text{peso})$ donde actual es el vértice de donde partimos (en este caso sería 1) adyacente son los vértices que unen la arista e (en este caso serían los vértices 2 y 4) y peso son los pesos de cada arista (en este caso tendríamos 7 y 2).

Evaluamos primero para vértice 2:

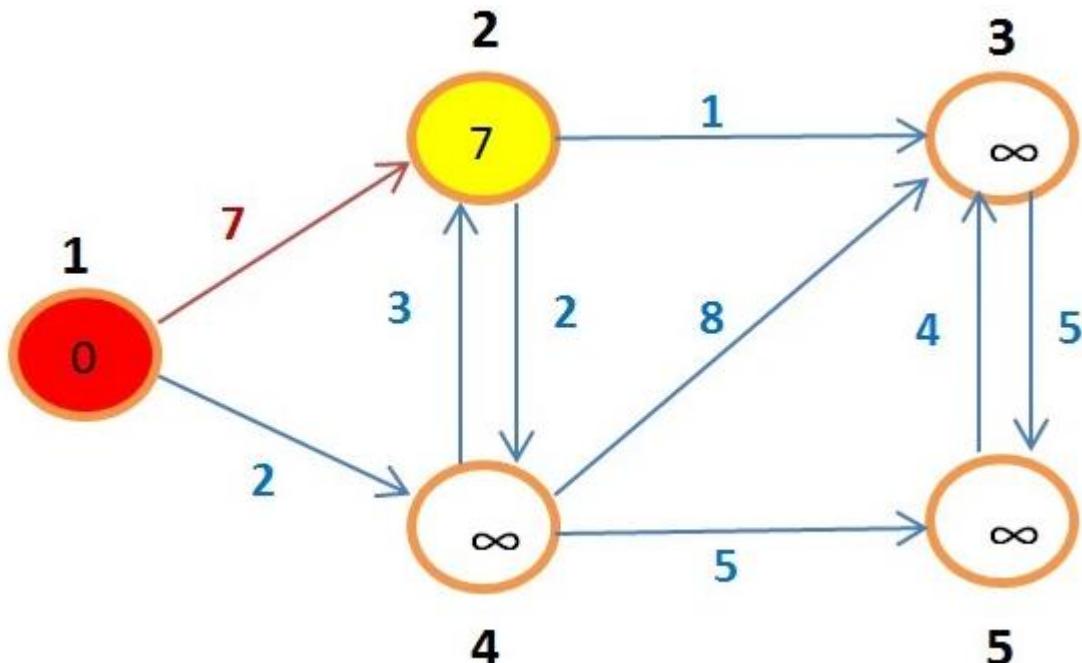


Vemos que la distancia actual desde el vértice inicial a 2 es ∞ , verifiquemos el paso de relajación:

$$\text{distancia}[1] + 7 < \text{distancia}[2] \rightarrow 0 + 7 < \infty \rightarrow 7 < \infty$$

El paso de relajación es posible realizarlo entonces actualizamos la distancia en el vértice 2 quedando:

Vértices	1	2	3	4	5
Distancia[u]	0	7	∞	∞	∞
Previo[u]	-1	1	-1	-1	-1



El paso de relajación se daría en la siguiente parte:

```

for( int actual = 1 ; actual <= V ; ++actual ){ //Estos dos for = O(E)
    for( int j = 0 ; j < ady[ actual ].size() ; ++j ){ //reviso sus adyacentes del vértice actual
        int adyacente = ady[ actual ][ j ].v; //id del vértice adyacente
        int peso = ady[ actual ][ j ].w; //peso de la arista que une actual con adyacente (
        actual , adyacente )
        //Realizamos paso de relajación para la arista actual
        relajacion( actual , adyacente , peso );
    }
}

```

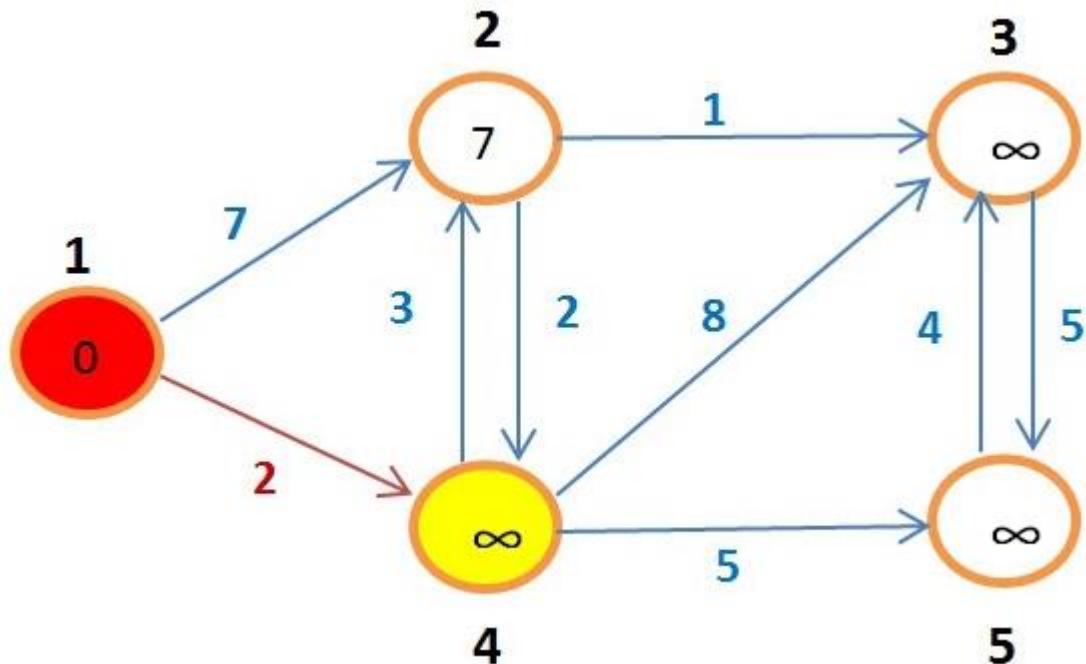
Donde la función de relajación seria

```

//Paso de relajacion
void relajacion( int actual , int adyacente , int peso ){
    //Si la distancia del origen al vértice actual + peso de su arista es menor a la distancia
    //del origen al vertice adyacente
    if( distancia[ actual ] + peso < distancia[ adyacente ] ){
        distancia[ adyacente ] = distancia[ actual ] + peso; //relajamos el vértice actualizando
        la distancia
        previo[ adyacente ] = actual; //a su vez actualizamos el vértice previo
        Q.push( Node( adyacente , distancia[ adyacente ] ) ); //agregamos adyacente a la
        cola de prioridad
    }
}

```

Ahora evaluamos al siguiente adyacente que es el vértice 4:

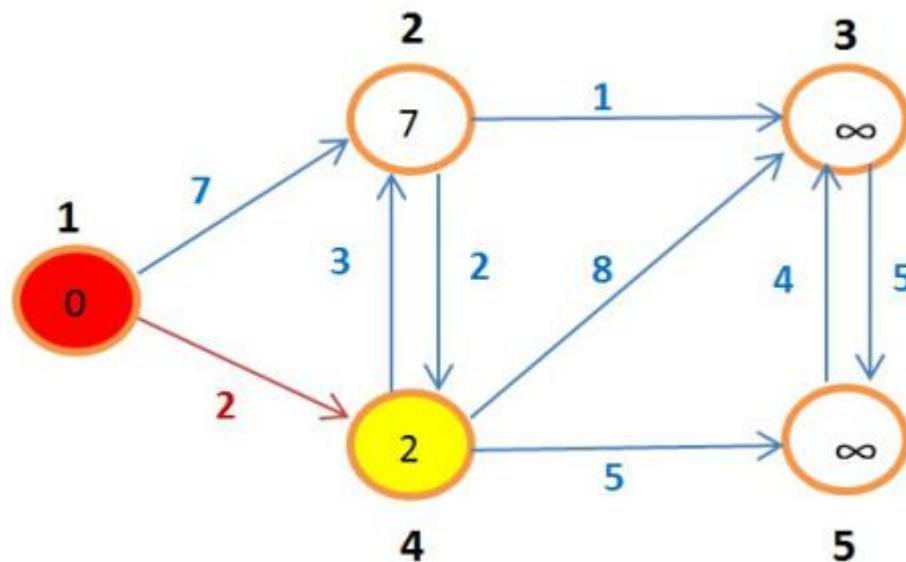


De manera similar al anterior vemos que la distancia actual desde el vértice inicial a 4 es ∞ , verifiquemos el paso de relajación:

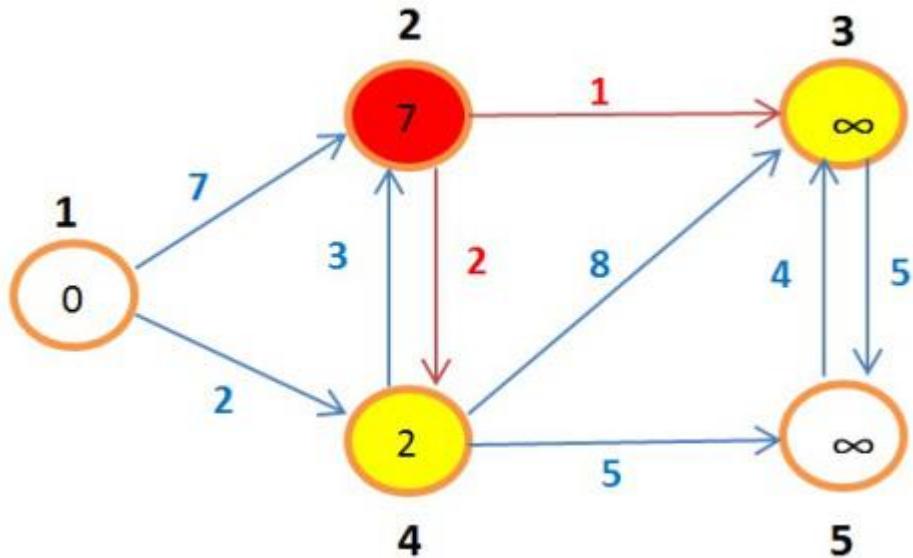
$$\text{distancia}[1] + 2 < \text{distancia}[4] \rightarrow 0 + 2 < \infty \rightarrow 2 < \infty$$

El paso de relajación es posible realizarlo entonces actualizamos la distancia en el vértice 4 quedando:

Vértices	1	2	3	4	5
Distancia[u]	0	7	∞	2	∞
Previo[u]	-1	1	-1	1	-1



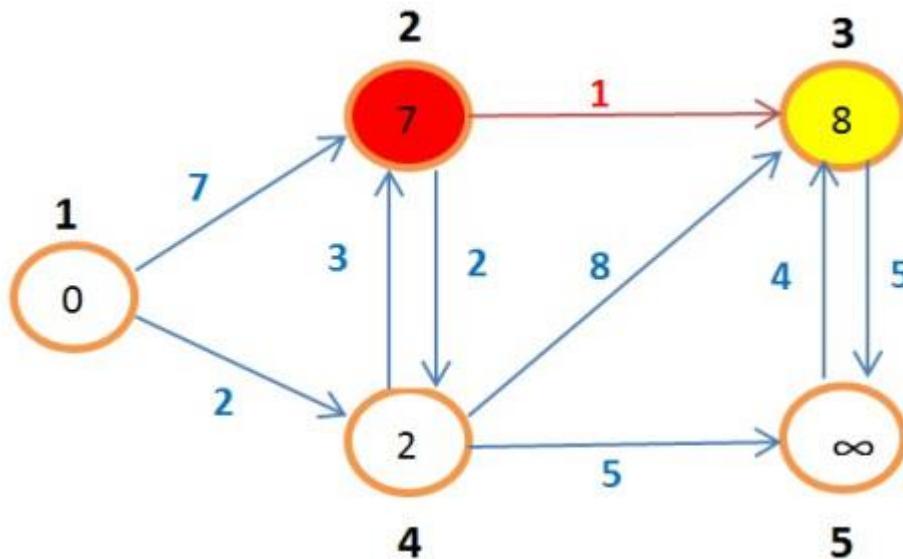
Hemos terminado las aristas que parten del vértice 1, continuamos con las aristas que parten del vértice 2:



Comenzamos por el vértice 3 y realizamos paso de relajación:

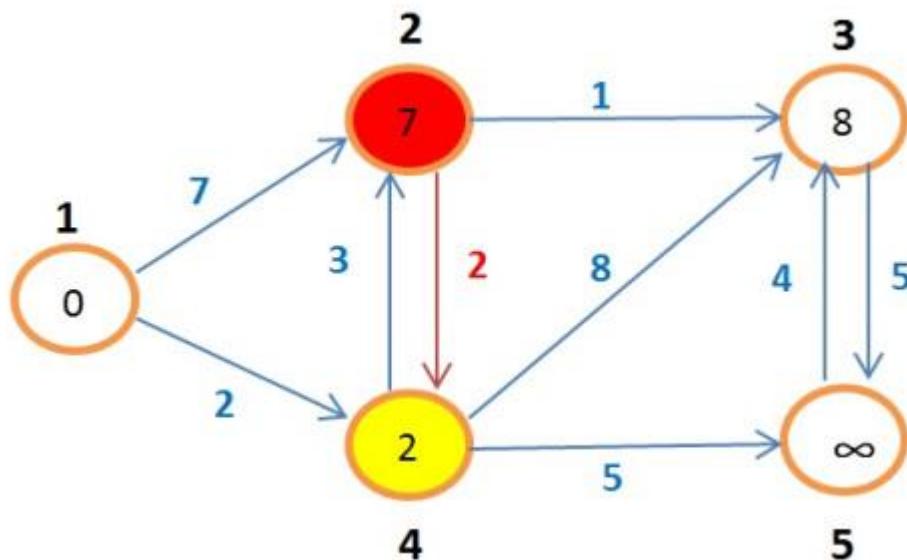
$$\text{distancia}[2] + 1 < \text{distancia}[3] \rightarrow 7 + 1 < \infty \rightarrow 8 < \infty$$

En esta oportunidad hemos encontrado una ruta más corta partiendo desde el vértice inicial al vértice 3, actualizamos la distancia en el vértice 3 y actualizamos el vértice previo al actual quedando:



Vértices	1	2	3	4	5
Distancia[u]	0	7	8	2	∞
Previo[u]	-1	1	2	1	-1

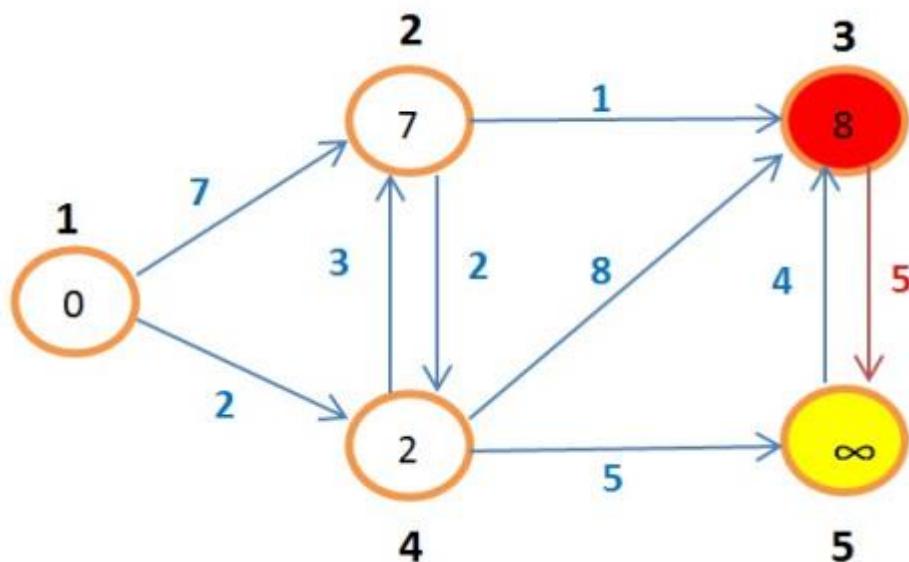
Ahora continuamos con la arista que une al vértice 2 con el vértice 4:



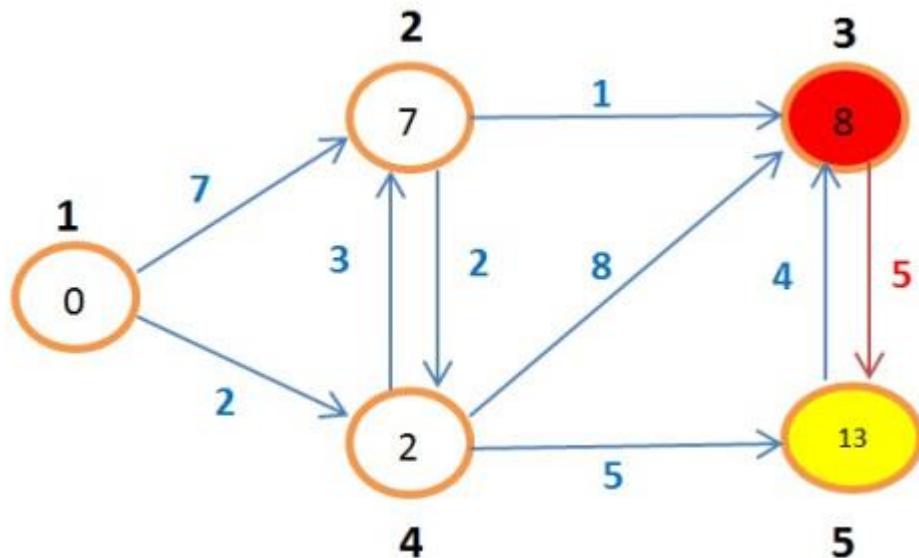
En este caso vemos que no se lleva acabo el paso de relajación:

$$\text{distancia}[2] + 2 < \text{distancia}[4] \rightarrow 7 + 2 < 2 \rightarrow 9 < 2$$

Por lo tanto no actualizamos valores en la tabla. Ahora el siguiente vértice a evaluar es el vértice 3 que posee una sola arista:

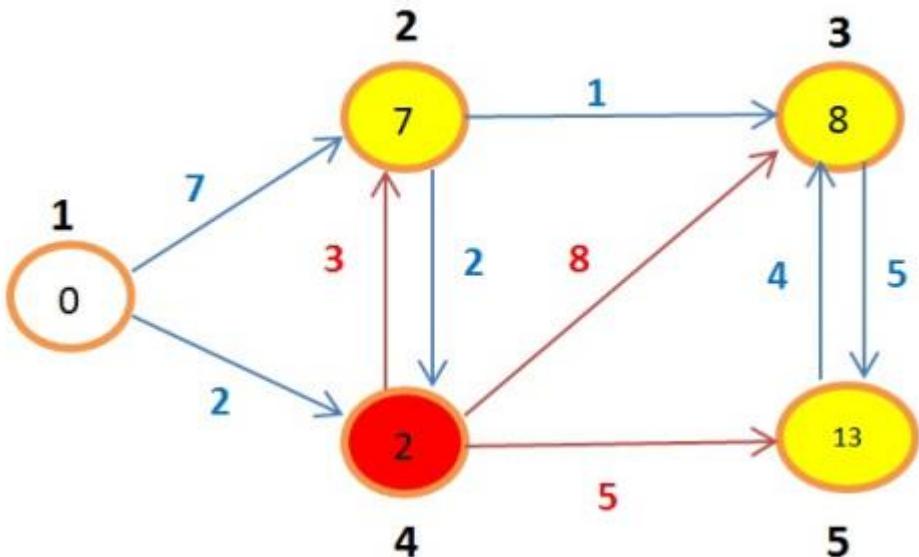


Como el peso de su vértice adyacente es infinito actualizamos la distancia:



Vértices	1	2	3	4	5
Distancia[u]	0	7	8	2	13
Previo[u]	-1	1	2	1	3

Ahora el siguiente vértice a evaluar es el vértice 4 que posee cuatro aristas:



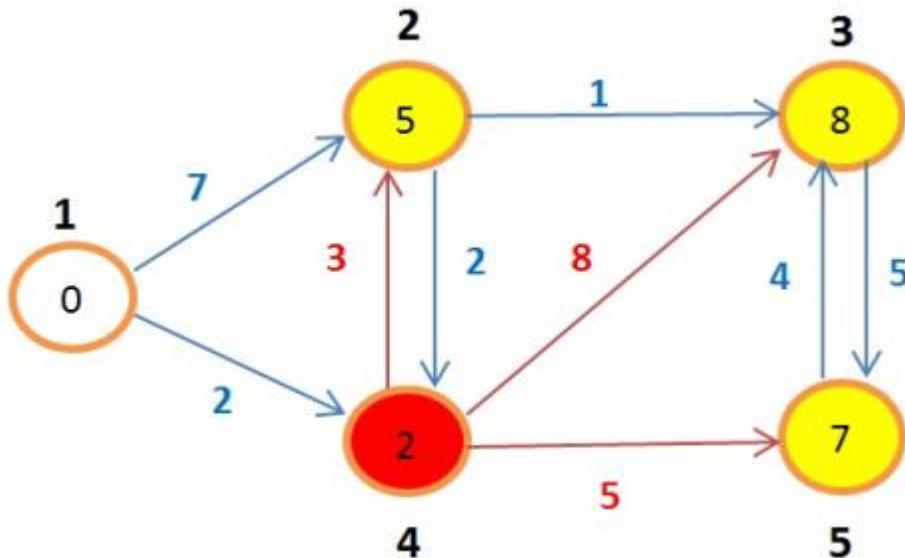
Realizamos paso de relajación para cada vértice adyacente:

Con vértice 2: $\text{distancia}[4] + 3 < \text{distancia}[2] \rightarrow 2 + 3 < 7 \rightarrow 5 < 7$

Con vértice 3: $\text{distancia}[4] + 8 < \text{distancia}[3] \rightarrow 2 + 8 < 8 \rightarrow 10 < 8$

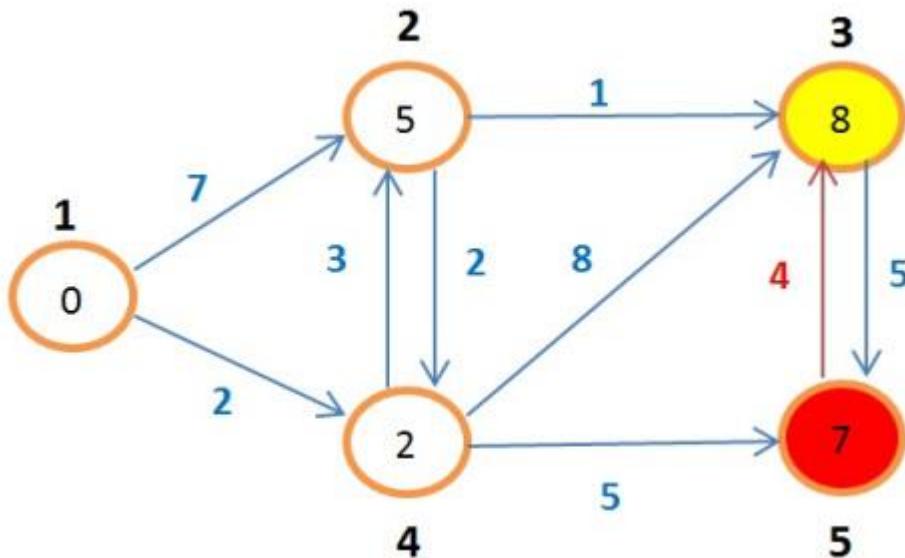
Con vértice 5: $\text{distancia}[4] + 5 < \text{distancia}[5] \rightarrow 2 + 5 < 13 \rightarrow 7 < 13$

Actualizamos distancias para los vértices 2 y 5:



Vértices	1	2	3	4	5
Distancia[u]	0	5	8	2	7
Previo[u]	-1	4	2	1	4

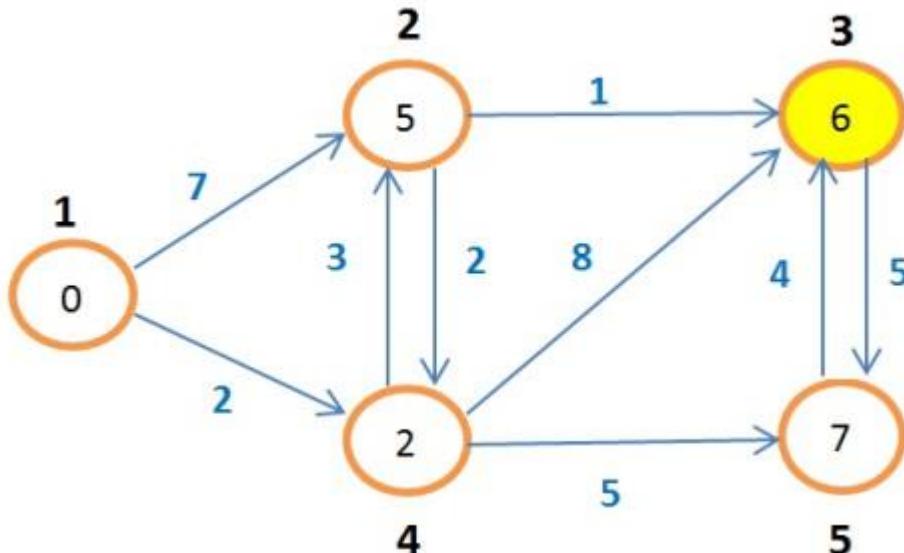
Ahora continuamos con vértice 5:



En este caso no actualizamos las distancias. Hemos terminado la primera iteración, continuemos:

Segunda Iteración:

Luego de la segunda iteración obtendremos lo siguiente:



Vértices	1	2	3	4	5
Distancia[u]	0	5	6	2	7
Previo[u]	-1	4	2	1	4

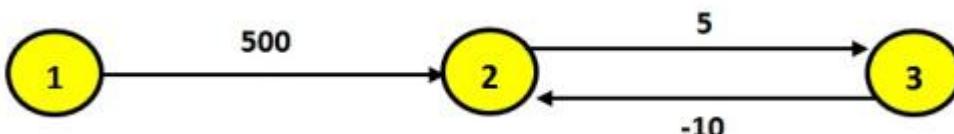
En esta iteración solamente se realizó el paso de relajación en la arista que une vértices 2 y 3. Para el grafo dado en la segunda iteración ya habremos obtenido la ruta más corta partiendo del vértice 1 a todos los demás vértices. Sin embargo no siempre obtendremos el óptimo en la 2da iteración, todo dependerá del grafo ingresado.

Impresión del Camino Más Corto

La impresión del camino más corto dado un vértice destino es de la misma forma como se explicó en el tutorial del [Algoritmo de Dijkstra](#).

Detección de Ciclo Negativo

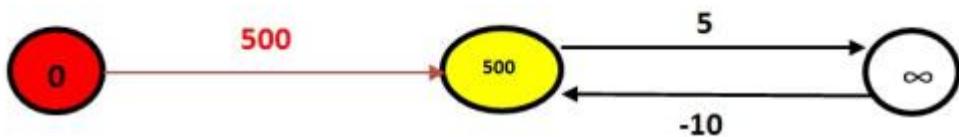
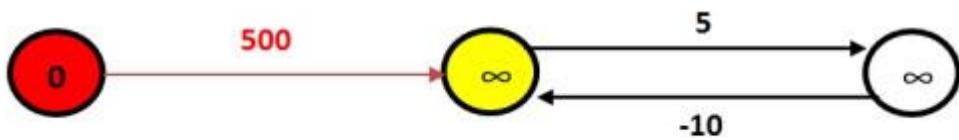
Para la detección de ciclo negativo tomaremos como ejemplo el grafo siguiente:



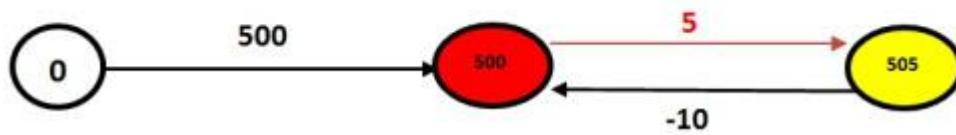
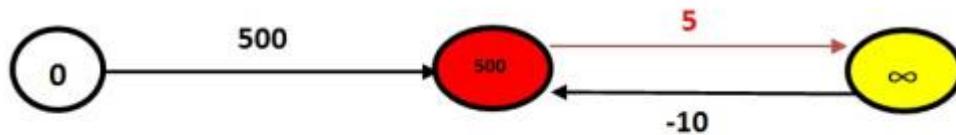
Asumimos que el vértice inicial es el vértice 1, tendremos que realizar 2 iteraciones.

Primera Iteración

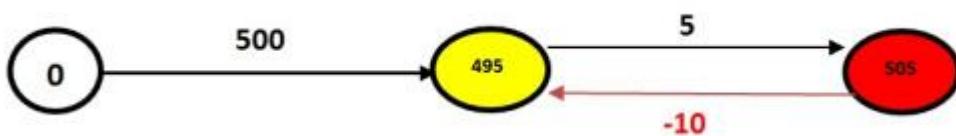
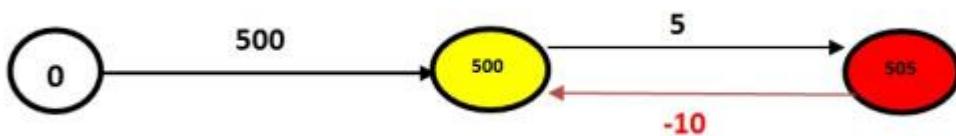
Empezamos por el vértice 1:



Continuamos con el vértice 2:

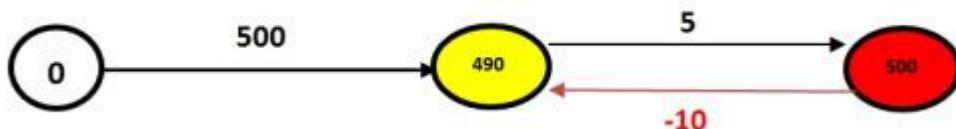
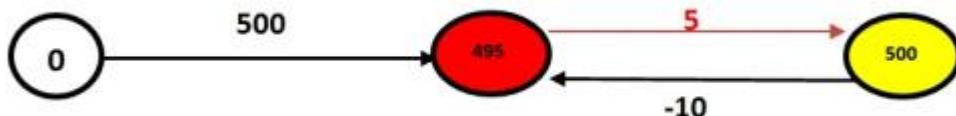


Continuamos con el vértice 3:

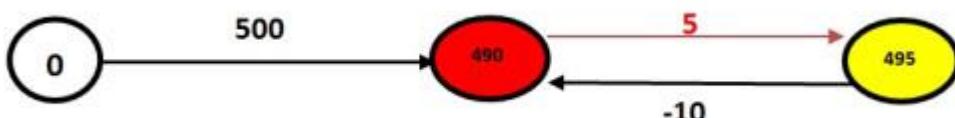


Segunda Iteración

En esta última iteración se relajan vértices 2 y 3:



Hemos terminado el número de iteraciones que teníamos que realizar. Para verificar la existencia de un ciclo negativo, según el algoritmo de Bellman-Ford, realizamos el paso de relajación sobre todas las aristas una vez más:



Como se pudo realizar el paso de relajación entonces existe un ciclo negativo:

```
//Comprobamos si hay ciclo negativo en el grafo ingresado
for( int actual = 1 ; actual <= V ; ++actual ){ //Estos dos for = O(E)
    for( int j = 0 ; j < ady[ actual ].size() ; ++j ){
        int adyacente = ady[ actual ][ j ].v; //id del vértice adyacente
        int peso = ady[ actual ][ j ].w; //peso de la arista que une actual con adyacente (origen , destino )
        //Si aún es posible relajar la arista actual entonces tenemos ciclo negativo
        if( relajacion( actual , adyacente , peso ) ){
            puts("Existe Ciclo Negativo");
            return;
        }
    }
}
```

PRIMER MOMENTO. INGRESO A NETBEANS

Practica desarrollada por los estudiantes:

Juan Pablo Estrada Ortiz

John Jairo Velásquez

Estructura de Datos

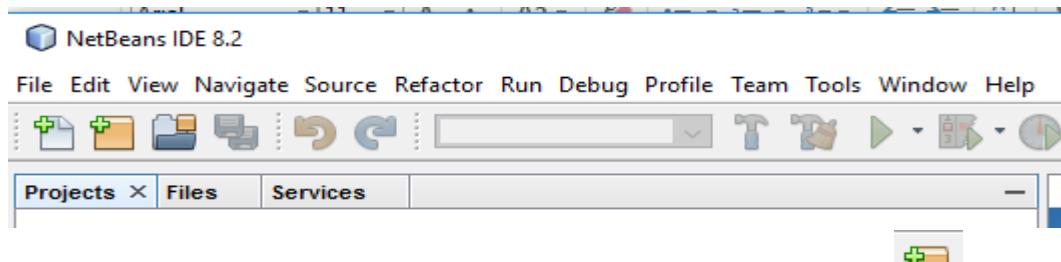
Programa Ingeniería de Sistemas

Universidad Autónoma de Colombia

2017/2

Menú Inicio>>Todas las Aplicaciones>>Netbeans>>Netbeans 8.xx

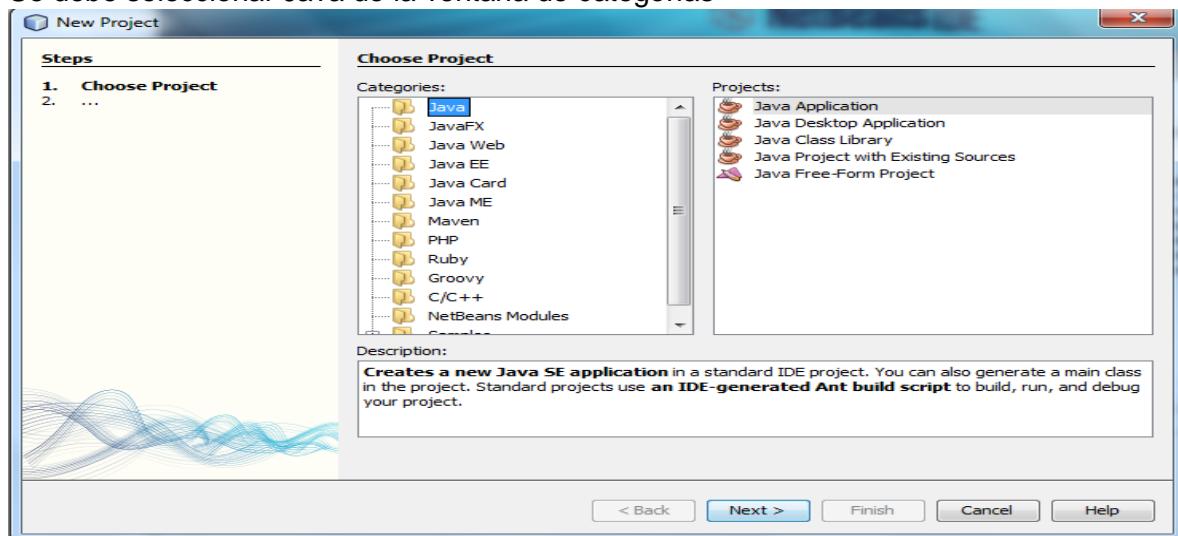
Ventana de Netbeans



Se va a crear un nuevo proyecto. Seleccionar y picar sobre el icono

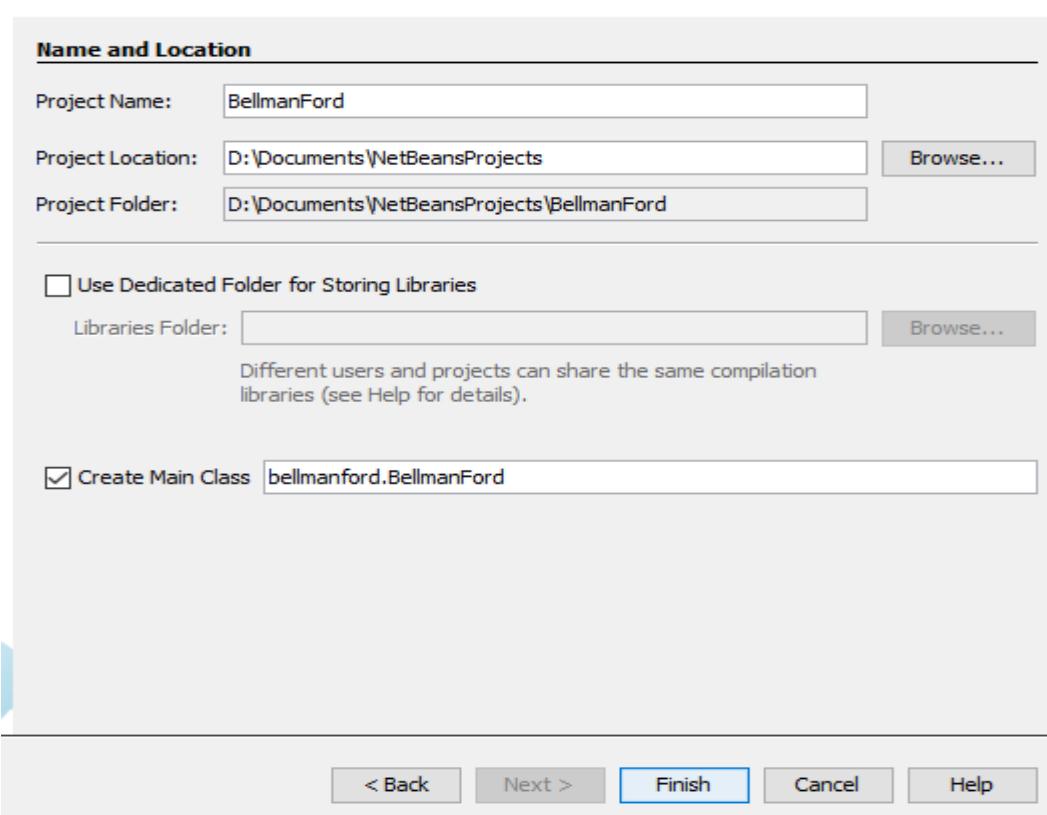


Se debe seleccionar Java de la ventana de categorías



Ahora, se selecciona Java Application de tipo de proyecto. Pulsar el Botón Next

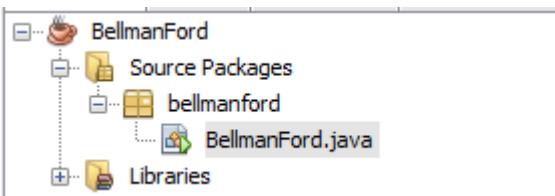
Nombre del Proyecto: BellmanFord



Pulsar el Botón Finish

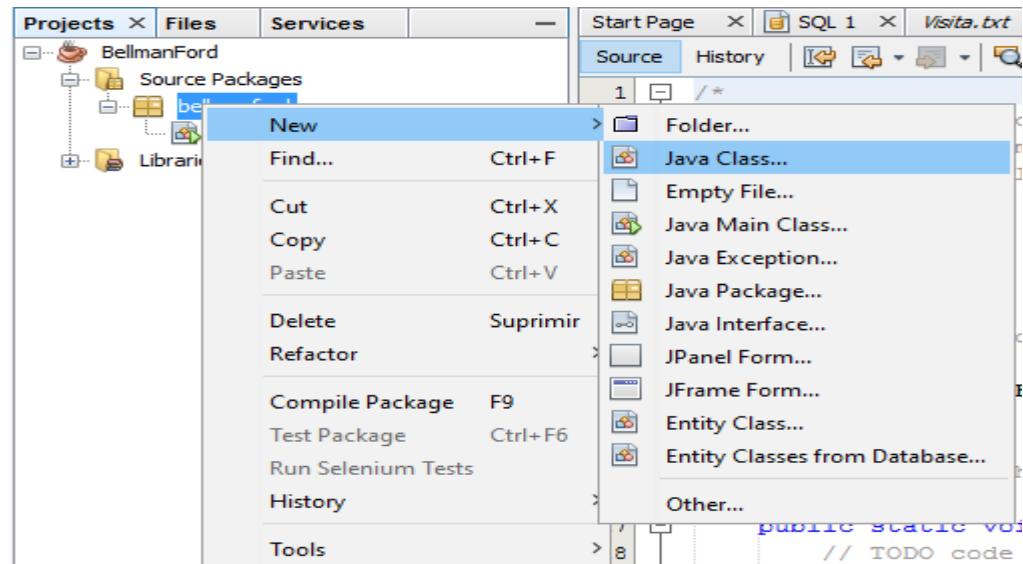
El Netbeans genera la estructura de un proyecto con:

- Paquete de Programas Fuentes(Source Packages):: Bellmanford
- Programa Principal:: BellamanFord.java



CREACION DE LA CLASE Admin

Se debe posiciona sobre el paquete bellmanford y pulsar el botón derecho del mouse>>opción new>>Java Class



Nombre de la Clase: **Admin**

Name and Location

Class Name:	Admin
Project:	BellmanFord
Location:	Source Packages
Package:	bellmanford
Created File:	D:\Documents\NetBeansProjects\BellmanFord\src\bellmanford\Admin.java

Pulsar el Botón Finish

Colocar en esta línea:

```
6 package bellmanford;  
7 |  
8  
9 public class Admin {
```



import java.util.*;

import java.io.*;

import javax.swing.*;

Quedando...

```
1 /*  
2  * To change this license header, choose License Headers in Project Properties.  
3  * To change this template file, choose Tools | Templates  
4  * and open the template in the editor.  
5  */  
6 package bellmanford;  
7 import java.util.*;  
8 import java.io.*;  
9 import javax.swing.*;  
10
```

Se debe reemplazar la class Admin

```
12 public class Admin {  
13 |  
14 }  
15
```



Por el siguiente contenido:

Quedando:

```
private final LinkedList<Aristas> aristas;  
final float etiquetas[];  
final int predecesor[];  
final int numeroVertices;  
private final int totalAristas;  
final int nodoOrigen;  
private final int INFINITY = 999;  
private static class Aristas {  
    int origen, destino;  
    float coste;  
  
    public Aristas(int a, int b, float c) {  
        origen = a;  
        destino = b;  
        coste = c;  
    }
```

@Override

```

        public String toString() {
            return "Aristas{" + "origen=" + origen + ", destino=" + destino + ", coste=" + coste +
        '}';
    }
}

public Admin() throws IOException {
    float item;
    aristas = new LinkedList<>();
    numeroVertices = Integer.parseInt(JOptionPane.showInputDialog(null, "Introduce
numero de vertices "));
    JOptionPane.showMessageDialog(null,"Matriz de costes");
    for (int i = 0; i < numeroVertices; i++) {
        for (int j = 0; j < numeroVertices; j++) {
            if (i != j) {
                item = Float.parseFloat(JOptionPane.showInputDialog(null, "Introduce coste
del nodo " + (i + 1) + " al nodo " + (j + 1)));
                if (item != 0) {
                    aristas.add(new Aristas(i, j, item));
                }
            }
        }
    }
    totalAristas = aristas.size();
    etiquetas = new float[numeroVertices];
    predecesor = new int[numeroVertices];
    nodoOrigen = Integer.parseInt(JOptionPane.showInputDialog(null, "Introduce el
vertice origen"))-1;
}
void relajoArista() {
    String T="";
    int i, j;
    for (i = 0; i < numeroVertices; ++i) {
        etiquetas[i] = INFINITY;
    }
    System.out.println(nodoOrigen);
    etiquetas[nodoOrigen] = 0;
    for (i = 0; i < numeroVertices - 1; ++i) {
        for (j = 0; j < totalAristas; ++j) {
            T=T+"\n";
            T=T+"aristas(:"+j+"):"+aristas.get(j);
            if (etiquetas[aristas.get(j).origen] + aristas.get(j).coste <
etiquetas[aristas.get(j).destino]) {
                etiquetas[aristas.get(j).destino] = etiquetas[aristas.get(j).origen] +
aristas.get(j).coste;
                predecesor[aristas.get(j).destino] = aristas.get(j).origen;
            }
        }
        for (int p = 0; etiquetas.length < p; p++) {
            T=T+"\t" + etiquetas[p];
        }
    }
    JOptionPane.showMessageDialog(null,T);
}

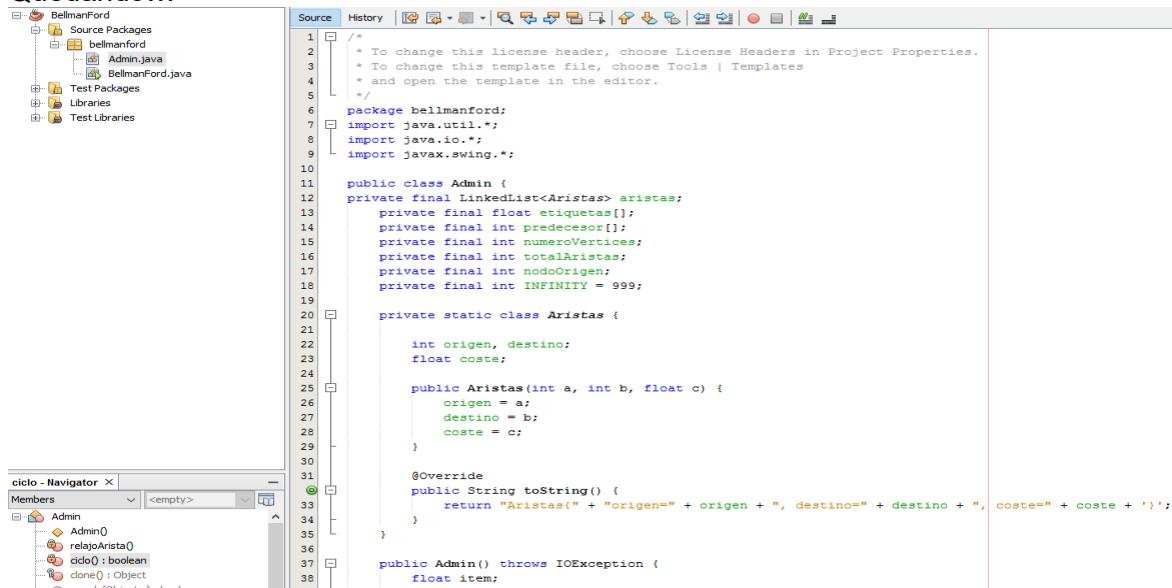
```

```

    }
    boolean ciclo() {
        int j;
        for (j = 0; j < totalAristas; ++j) {
            if (etiquetas[aristas.get(j).origen] + aristas.get(j).coste <
etiquetas[aristas.get(j).destino]) {
                return false;
            }
        }
        return true;
    }
}

```

Quedando...



The screenshot shows an IDE interface with the following details:

- Project Explorer:** Shows the project structure under "BellmanFord".
- Code Editor:** Displays the `Admin.java` file with the following code:

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates.
 * and open the template in the editor.
 */
package bellmanford;
import java.util.*;
import java.io.*;
import javax.swing.*;

public class Admin {
    private final LinkedList<Aristas> aristas;
    private final float etiquetas[];
    private final int predecesor[];
    private final int numerVertices;
    private final int totalAristas;
    private final int nodoOrigen;
    private final int INFINITY = 999;

    private static class Aristas {
        int origen, destino;
        float coste;

        public Aristas(int a, int b, float c) {
            origen = a;
            destino = b;
            coste = c;
        }

        @Override
        public String toString() {
            return "Aristas(" + "origen=" + origen + ", destino=" + destino + ", coste=" + coste + ')';
        }
    }

    public Admin() throws IOException {
        float item;
    }
}

```

- Navigator:** Shows the members of the `Admin` class, including `ciclo()`, `relajaArista()`, and `clone()`.

ACTUALIZACION DE LA CLASE BellmanFord.java

Hacer doble click en BellmanFord.java...

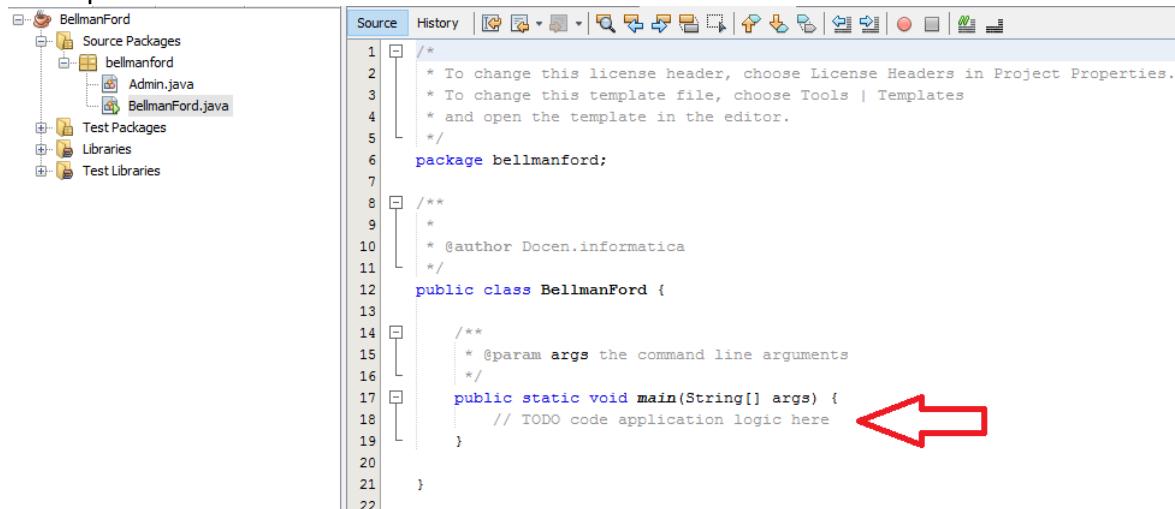
Colocar en esta línea:

```
6 package bellmanford;
7
8 /**
9 *
10 * @author Docen.informatica
11 */
12 public class BellmanFord {
import java.io.IOException;
import javax.swing.*;
```

Quedando:

```
1 package bellmanford;
2 import java.io.IOException;
3 import javax.swing.*;
4 public class BellmanFord {
```

Reemplazar en:



Por las siguientes Líneas de Código:

```
Admin bellman = new Admin();
bellman.relajoArista();
if (bellman.ciclo()) {
for (int i = 0; i < bellman.numeroVertices; i++)
{
System.out.println("Coste desde " + bellman.nodoOrigen + " a " + (i + 1) + " =>" +
bellman.etiquetas[i]);
}
for (int i = 0; i < bellman.numeroVertices; i++)
{
    System.out.println("El predecesor de " + (i + 1) + " es " + (bellman.predecesor[i]
+ 1));
}
}
```

```

else
{
    System.out.println("Hay un ciclo negativo");
}
}

```

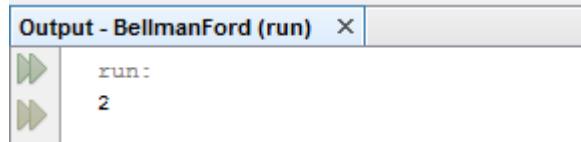
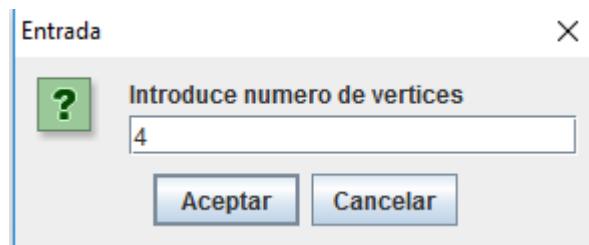
Quedando:

```

1 package bellmanford;
2 import java.io.IOException;
3 import javax.swing.*;
4 public class BellmanFord {
5     public static void main(String[] args) throws IOException {
6         Admin bellman = new Admin();
7         bellman.relaajoArista();
8         if (bellman.ciclo()) {
9             for (int i = 0; i < bellman.numeroVertices; i++)
10            {
11                JOptionPane.showMessageDialog(null,"Coste desde " + bellman.nodoOrigen + " a " + (i + 1) + " =>" + bellman.etiquetas[i]);
12            }
13            for (int i = 0; i < bellman.numeroVertices; i++)
14            {
15                JOptionPane.showMessageDialog(null,"El predecesor de " + (i + 1) + " es " + (bellman.predecesor[i] + 1));
16            }
17        }
18        else
19        {
20            JOptionPane.showMessageDialog(null,"Hay un ciclo negativo");
21        }
22    }
23 }
24

```

Pulsar F5 o 



Mensaje

X



```
aristas(:0):Aristas{origen=0, destino=1, coste=2.0}
aristas(:1):Aristas{origen=0, destino=2, coste=3.0}
aristas(:2):Aristas{origen=0, destino=3, coste=1.0}
aristas(:3):Aristas{origen=1, destino=0, coste=2.0}
aristas(:4):Aristas{origen=1, destino=2, coste=4.0}
aristas(:5):Aristas{origen=1, destino=3, coste=5.0}
aristas(:6):Aristas{origen=2, destino=0, coste=3.0}
aristas(:7):Aristas{origen=2, destino=1, coste=4.0}
aristas(:8):Aristas{origen=2, destino=3, coste=5.0}
aristas(:9):Aristas{origen=3, destino=0, coste=6.0}
aristas(:10):Aristas{origen=3, destino=1, coste=2.0}
aristas(:11):Aristas{origen=3, destino=2, coste=3.0}
aristas(:0):Aristas{origen=0, destino=1, coste=2.0}
aristas(:1):Aristas{origen=0, destino=2, coste=3.0}
aristas(:2):Aristas{origen=0, destino=3, coste=1.0}
aristas(:3):Aristas{origen=1, destino=0, coste=2.0}
aristas(:4):Aristas{origen=1, destino=2, coste=4.0}
aristas(:5):Aristas{origen=1, destino=3, coste=5.0}
aristas(:6):Aristas{origen=2, destino=0, coste=3.0}
aristas(:7):Aristas{origen=2, destino=1, coste=4.0}
aristas(:8):Aristas{origen=2, destino=3, coste=5.0}
aristas(:9):Aristas{origen=3, destino=0, coste=6.0}
aristas(:10):Aristas{origen=3, destino=1, coste=2.0}
aristas(:11):Aristas{origen=3, destino=2, coste=3.0}
aristas(:0):Aristas{origen=0, destino=1, coste=2.0}
aristas(:1):Aristas{origen=0, destino=2, coste=3.0}
aristas(:2):Aristas{origen=0, destino=3, coste=1.0}
aristas(:3):Aristas{origen=1, destino=0, coste=2.0}
aristas(:4):Aristas{origen=1, destino=2, coste=4.0}
aristas(:5):Aristas{origen=1, destino=3, coste=5.0}
aristas(:6):Aristas{origen=2, destino=0, coste=3.0}
aristas(:7):Aristas{origen=2, destino=1, coste=4.0}
aristas(:8):Aristas{origen=2, destino=3, coste=5.0}
aristas(:9):Aristas{origen=3, destino=0, coste=6.0}
aristas(:10):Aristas{origen=3, destino=1, coste=2.0}
aristas(:11):Aristas{origen=3, destino=2, coste=3.0}
```

Aceptar

**UNIVERSIDAD SANTO TOMAS
DUAD
FACULTAD DE CYT
PROGRAMA INGENIERIA EN INFORMATICA**

PRACTICA ALGORITMO DE FLOYD-WARSHALL

**MARIO DUSTANO CONTRERAS CASTRO
DOCENTE TIEMPO COMPLETO**

Algoritmo de Floyd-Warshall

<http://dalila.sip.ucm.es/~manuel/Informatica/FloydWarshall.pdf>

Grafo Ponderado o Etiquetado. Es preciso atribuir a cada arista un número específico, llamado valuación, ponderación o coste según el contexto, y se obtiene así un grafo valuado o ponderado.

Algoritmo de Floyd-Warshall compara todos los posibles caminos a través del grafo entre cada par de vértices. El algoritmo es capaz de hacer esto con sólo V^3 comparaciones

(esto es notable considerando que puede haber hasta V^2 aristas en el grafo, y que cada combinación de aristas se prueba). Lo hace mejorando paulatinamente una estimación del camino más corto entre dos vértices, hasta que se sabe que la estimación es óptima.

Sea un grafo con conjunto de vértices V , numerados de 1 a N . Sea además una función $\text{caminoMinimo}(i,j,k)$ que devuelve el camino mínimo de i a j usando únicamente los vértices de 1 a k como puntos intermedios en el camino. Ahora, dada esta función, nuestro objetivo es encontrar el camino mínimo desde cada i a cada j usando únicamente los vértices de 1 hasta k .

Hay dos candidatos para este camino: un camino mínimo, que utiliza únicamente los vértices del conjunto $(1\dots k)$; o bien existe un camino que va desde i hasta $k+1$, y de $k+1$ hasta j , que es mejor. Sabemos que el camino óptimo de i a j que únicamente utiliza los vértices de 1 hasta k está definido por $\text{caminoMinimo}(i,j,k)$, y está claro que si hubiera un camino mejor de i a $k+1$ a j , la longitud de este camino sería la concatenación del camino mínimo de i a $k+1$ (utilizando vértices de $(1\dots k)$) y el camino mínimo de $k+1$ a j (que también utiliza los vértices en $(1\dots k)$).

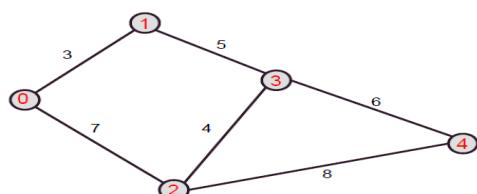
Por lo tanto, podemos definir $\text{caminoMinimo}(i,j,k)$ de forma recursiva:

$$\begin{aligned}\text{caminoMinimo}(i, j, k) &= \min(\text{caminoMinimo}(i, j, k - 1), \text{caminoMinimo}(i, k, k - 1) + \text{caminoMinimo}(k, j, k - 1)); \\ \text{caminoMinimo}(i, j, 0) &= \text{pesoArista}(i, j);\end{aligned}$$

Esta fórmula es la base del algoritmo Floyd-Warshall. Funciona ejecutando primero $\text{caminoMinimo}(i, j, 1)$ para todos los pares (i, j) , usándolos para después hallar $\text{caminoMinimo}(i, j, 2)$ para todos los pares (i, j) ... Este proceso continúa hasta que $k=n$, y habremos encontrado el camino más corto para todos los pares de vértices i, j usando algún vértice intermedio.

Objetivo

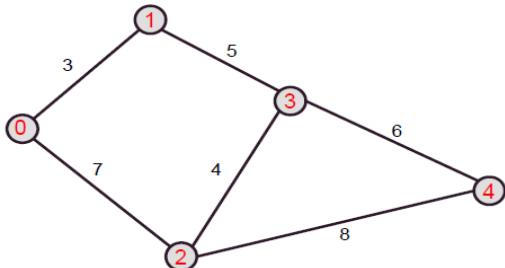
Dado un grafo ponderado, queremos obtener el camino de distancia mínima entre dos vértices cualesquiera.



0	0	3	7	8	14
1	3	0	9	5	11
2	7	9	0	4	8
3	8	5	4	0	6
4	14	11	8	6	0

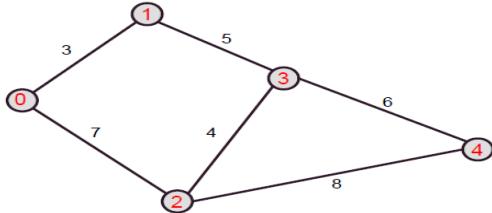
Representación

Matriz de adyacencia, con los pesos de cada arista. Si no hay arista entre dos vértices determinados, se considera $+\infty$



0	1	2	3	4
0	0	3	7	$+\infty$
1	3	0	$+\infty$	5
2	7	$+\infty$	0	4
3	$+\infty$	5	4	0
4	$+\infty$	$+\infty$	8	6

Matriz de adyacencia, con los pesos de cada arista. Si no hay arista entre dos vértices determinados, se considera $+\infty$



0	1	2	3	4
0	0	3	7	$+\infty$
1	3	0	$+\infty$	5
2	7	$+\infty$	0	4
3	$+\infty$	5	4	0
4	$+\infty$	$+\infty$	8	6

Simétrica, con ceros en la diagonal

Sólo almacenamos los elementos por debajo de la diagonal principal.

0	1	2	3	4
0	0	3	7	$+\infty$
1	3	0	$+\infty$	5
2	7	$+\infty$	0	4
3	$+\infty$	5	4	0
4	$+\infty$	$+\infty$	8	6

`[[3], [7, inf], [inf, 5, 4], [inf, inf, 8, 6]]`

① ② ③ ④

Cálculo del Camino Mínimo

- Supongamos que tenemos n ciudades, numeradas desde 0 hasta $n-1$.
- Construimos una sucesión de matrices:

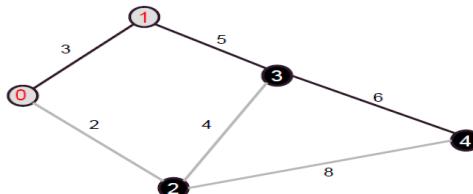
$$A_0 \rightarrow A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_n$$

El significado de $A_k(i,j)$ es el siguiente:

Longitud del camino mínimo que hay desde la ciudad i hasta la ciudad j , suponiendo que sólo podemos pasar por las ciudades comprendidas entre 0 y $k-1$.

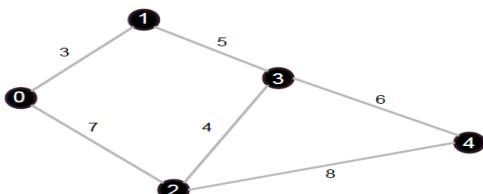
Ejemplo: A_2

$A_2(i, j) =$ Longitud del camino mínimo que hay desde la ciudad i hasta la ciudad j , suponiendo que sólo podemos pasar por las ciudades 0 y 1 .



$$\begin{aligned} A_2(0, 3) &= 8 \\ A_2(0, 4) &= +\infty \end{aligned}$$

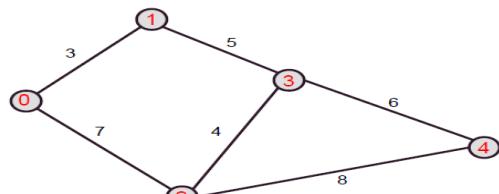
$A_0(i, j) =$ Longitud del camino mínimo que hay desde la ciudad i hasta la ciudad j , suponiendo no podemos pasar por ninguna ciudad intermedia.



	0	1	2	3	4
0	0	3	7	$+\infty$	$+\infty$
1	3	0	$+\infty$	5	$+\infty$
2	7	$+\infty$	0	4	8
3	$+\infty$	5	4	0	6
4	$+\infty$	$+\infty$	8	6	0

A_0 es la matriz de adyacencia del grafo

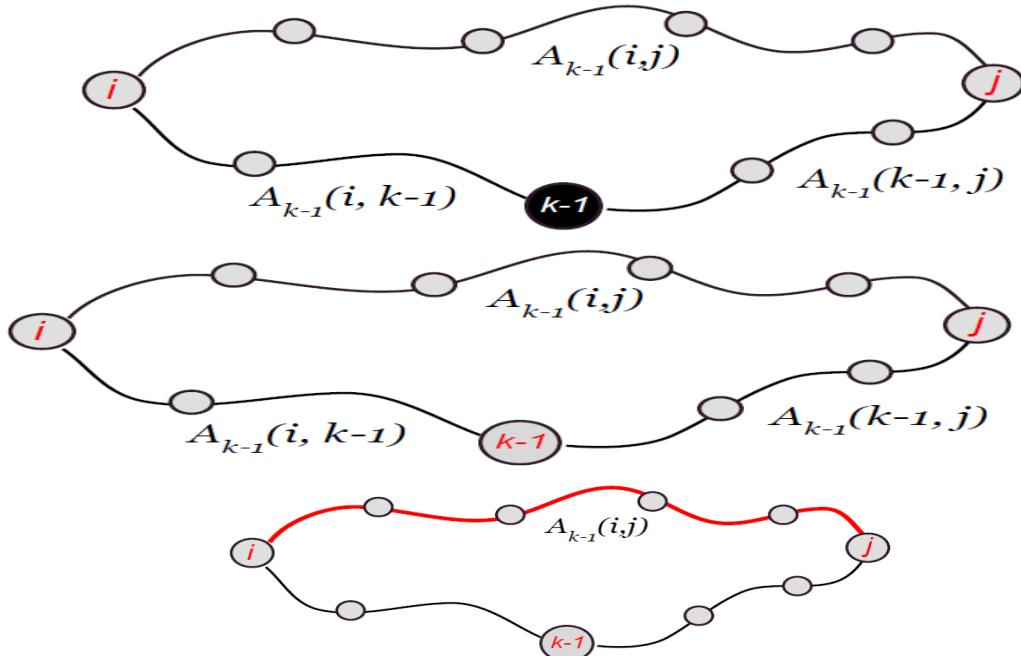
$A_n(i, j) =$ Longitud del camino mínimo que hay desde la ciudad i hasta la ciudad j , suponiendo que podemos pasar por cualquier ciudad intermedia.



A_n es lo que buscamos !

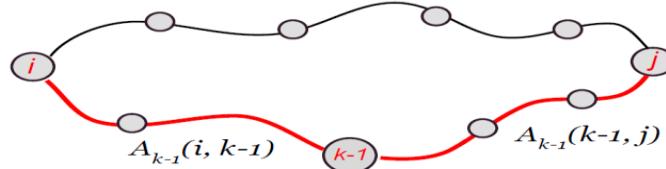
	0	1	2	3	4
0	0	3	7	8	14
1	3	0	9	5	11
2	7	9	0	4	8
3	8	5	4	0	6
4	14	11	8	6	0

¿Cómo calcular A_k a partir de A_{k-1} ?



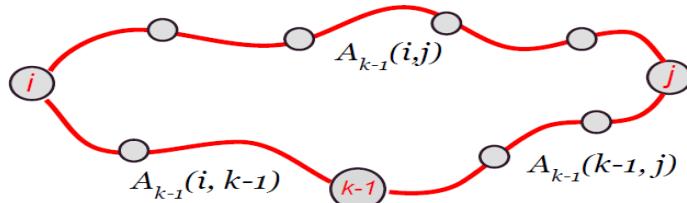
Si el nuevo camino mínimo **no** pasa por el nodo $k-1$

$$A_k(i,j) = A_{k-1}(i,j)$$



Si el nuevo camino mínimo pasa por el nodo $k-1$

$$A_k(i,j) = A_{k-1}(i,k-1) + A_{k-1}(k-1,j)$$



La opción que más nos conviene:

$$A_k(i,j) = \min \{ A_{k-1}(i,j), A_{k-1}(i,k-1) + A_{k-1}(k-1,j) \}$$

PRIMER MOMENTO. INGRESO A NETBEANS

Practica desarrollada por los estudiantes:

Brayan Andrés Vargas

Iván Darío Campos

Estructura de Datos

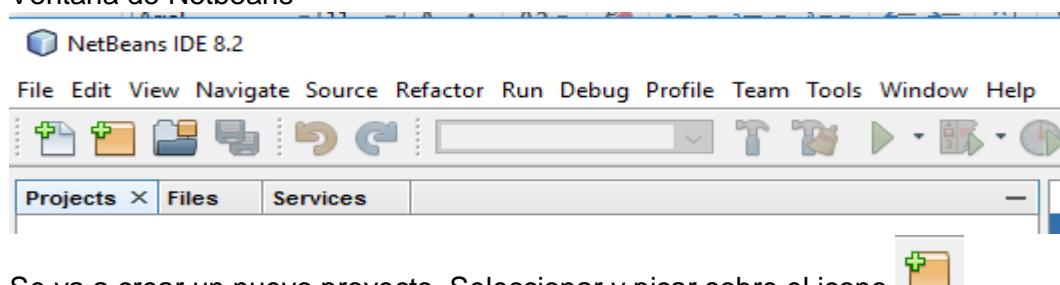
Programa Ingeniería de Sistemas

Universidad Autónoma de Colombia

2017/2

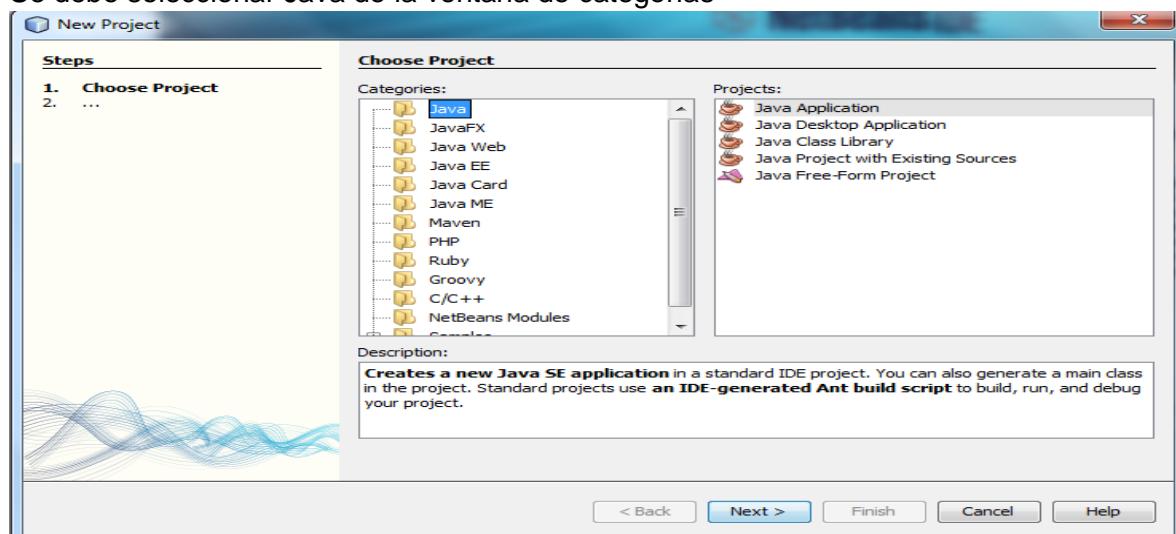
Menú Inicio>>Todas las Aplicaciones>>Netbeans>>Netbeans 8.xx

Ventana de Netbeans



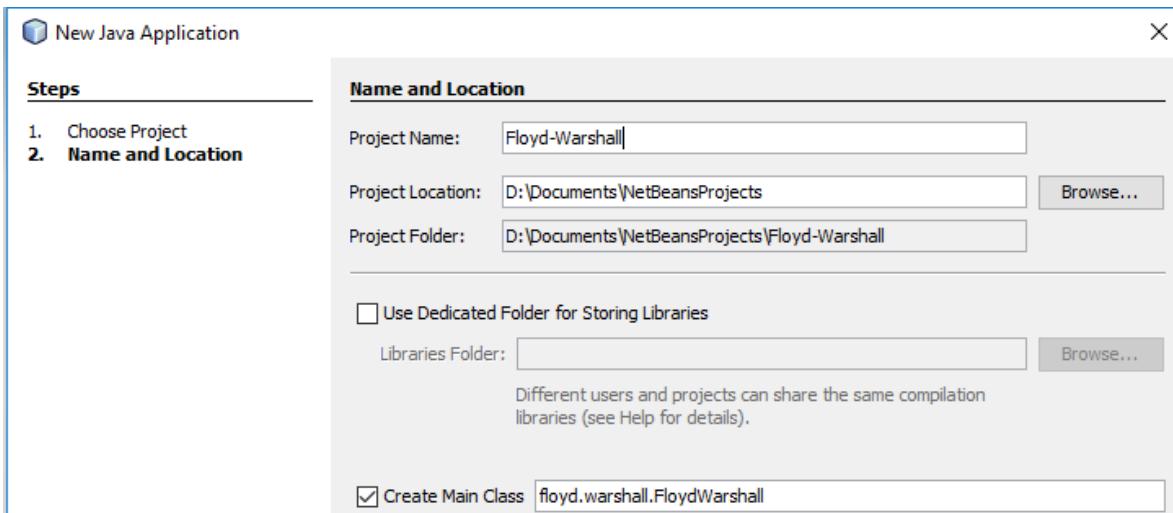
Se va a crear un nuevo proyecto. Seleccionar y picar sobre el icono

Se debe seleccionar Java de la ventana de categorías



Ahora, se selecciona Java Application de tipo de proyecto. Pulsar el Botón Next

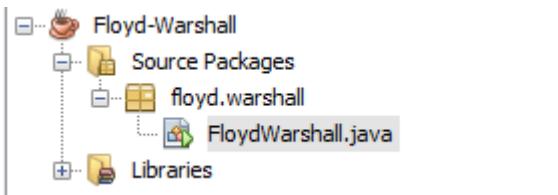
Nombre del Proyecto: **Floyd-Warshall**



Pulsar el Botón Finish

El Netbeans genera la estructura de un proyecto con:

- Paquete de Programas Fuentes(Source Packages):: floyd.warshall
- Programa Principal:: FloydWarshall.java



ACTUALIZACION DE LA CLASE Floyd-Warshall

Colocar en esta línea:

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package floyd.warshall;
7
8  /**
9   *
10  * @author Docen.informatica
11  */
12 public class FloydWarshall {
13
14     /**
15      * @param args the command line arguments
16     */
17     public static void main(String[] args) {
18         // TODO code application logic here
19     }
20
21 }
```



import java.util.Scanner;

Quedando:

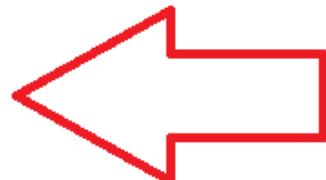
```
6  package floyd.warshall;
7  import java.util.Scanner;
8
9  /**
10  *
11  * @author Docen.informatica
12  */
13 public class FloydWarshall {
```

Reemplazar todo el contenido por:

```
public class FloydWarshall {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }

}
```



Por las siguientes Líneas de Código:

```
public static int[][] shortestpath(int[][] adj, int[][] path) {  
  
    int n = adj.length;  
    int[][] ans = new int[n][n];  
  
    copy(ans, adj);  
  
    for (int k = 0; k < n; k++) {  
  
        // Es así que entre cada par de puntos posibles.  
        for (int i = 0; i < n; i++) {  
            for (int j = 0; j < n; j++) {  
  
                if (ans[i][k] + ans[k][j] < ans[i][j]) {  
                    ans[i][j] = ans[i][k] + ans[k][j];  
                    path[i][j] = path[k][j];  
                }  
            }  
        }  
    }  
  
    // Devuelva la matriz camino más corto.  
    return ans;  
}  
public static void copy(int[][] a, int[][] b) {  
  
    for (int i = 0; i < a.length; i++) {  
        for (int j = 0; j < a[0].length; j++) {  
            a[i][j] = b[i][j];  
        }  
    }  
}  
public static int min(int a, int b) {  
    if (a < b) {  
        return a;  
    } else {  
        return b;  
    }  
}  
public static void main(String[] args) {  
    Scanner stdin = new Scanner(System.in);  
  
    int[][] m = new int[5][5];  
    m[0][0] = 0;
```

```

m[0][1] = 3;
m[0][2] = 8;
m[0][3] = 10000;
m[0][4] = -4;
m[1][0] = 10000;
m[1][1] = 0;
m[1][2] = 10000;
m[1][3] = 1;
m[1][4] = 7;
m[2][0] = 10000;
m[2][1] = 4;
m[2][2] = 0;
m[2][3] = 10000;
m[2][4] = 10000;
m[3][0] = 2;
m[3][1] = 10000;
m[3][2] = -5;
m[3][3] = 0;
m[3][4] = 10000;
m[4][0] = 10000;
m[4][1] = 10000;
m[4][2] = 10000;
m[4][3] = 6;
m[4][4] = 0;

int[][] shortpath;
int[][] path = new int[5][5];

for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; j++) {
        if (m[i][j] == 10000) {
            path[i][j] = -1;
        } else {
            path[i][j] = i;
        }
    }
}

for (int i = 0; i < 5; i++) {
    path[i][i] = i;
}
shortpath = shortestpath(m, path);

for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; j++) {
        System.out.print(shortpath[i][j] + " ");
    }
}

```

```

        }
        System.out.println();
    }

    System.out.println("De dónde a dónde usted quiere encontrar el camino más corto?(0
to 4)");
    int start = stdin.nextInt();
    int end = stdin.nextInt();

    String myPath = end + "";

    while (path[start][end] != start) {
        myPath = path[start][end] + " -> " + myPath;
        end = path[start][end];
    }

    myPath = start + " -> " + myPath;
    System.out.println("Este es el camino " + myPath);
}

```

Pulsar F5 o 

Output - Floyd-Warshall (run) X	
     	run: 0 1 -3 2 -4 3 0 -4 1 -1 7 4 0 5 3 2 -1 -5 0 -2 8 5 1 6 0 De dónde a dónde usted quiere encontrar el camino más corto?(0 to 4) 1 0 Este es el camino 1 -> 3 -> 0 BUILD SUCCESSFUL (total time: 22 seconds)

**UNIVERSIDAD SANTO TOMAS
DUAD
FACULTAD DE CYT
PROGRAMA INGENIERIA EN INFORMATICA**

PRACTICA ALGORITMO DE KRUSKAL

**MARIO DUSTANO CONTRERAS CASTRO
DOCENTE TIEMPO COMPLETO**

Bogotá, Junio 2018

Algoritmo de Kruskal. Árbol de coste total mínimo/máximo

Tomado de: <https://estructurasite.wordpress.com/algoritmo-de-kruskal/>

El **objetivo** del algoritmo de Kruskal es construir un árbol (subgrafo sin ciclos) formado por arcos sucesivamente seleccionados de mínimo peso a partir de un grafo con pesos en los arcos.

Un árbol (**spanning tree**) de un grafo es un subgrafo que contiene todos sus vértices o nodos. Un grafo puede tener múltiples árboles. Por ejemplo, un grafo completo de cuatro nodos (todos relacionados con todos) tendría 16 árboles.

La aplicación **típica** de este problema es el diseño de redes telefónicas. Una empresa con diferentes oficinas, trata de trazar líneas de teléfono para conectarlas unas con otras. La compañía telefónica le ofrece esta **interconexión**, pero ofrece tarifas diferentes o costes por conectar cada par de oficinas. Cómo conectar entonces las oficinas al mínimo coste total.

La formulación del Árbol de coste total mínimo (minimum spanning tree – MST) también ha sido aplicada para hallar soluciones en diversas áreas (diseño de redes de transporte, diseño de redes de telecomunicaciones – TV por cable, sistemas distribuidos, interpretación de datos climatológicos, visión artificial – análisis de imágenes – extracción de rasgos de parentesco, análisis de clusters y búsqueda de **superestructuras** de quasar, plegamiento de proteínas, reconocimiento de células cancerosas, y otros).

Otra aplicación menos **obvia** es que el árbol de coste total mínimo puede ser usado como solución aproximada al problema del viajante de comercio (traveling salesman problem), recuerde que encontrar la solución óptima a este problema es NP-Hard. La manera formal de definir este problema es encontrar la trayectoria más corta para visitar cada punto al menos una vez. Nótese que si se visitan todos los puntos exactamente una vez, lo que se tiene es un tipo especial de árbol. En el ejemplo anterior, 12 de los 16 árboles son trayectorias de este tipo. Si se tiene una trayectoria que visita algunos vértices o nodos más de una vez, siempre se puede soltar algunos nodos del árbol. En general el peso del árbol total mínimo es menor que el del viajante de comercio, debido a que su minimización se realiza sobre un conjunto estrictamente mayor. Existen diferentes algoritmos y maneras de usar el árbol de coste total mínimo para encontrar la solución al problema del viajante de comercio (con resultados cercanos al óptimo).

Algoritmo de Kruskal (árbol de coste total mínimo)

Dado un **grafo G** con nodos conectados por arcos con peso (coste o longitud): el peso o coste total de un árbol será la suma de pesos de sus arcos. Obviamente, árboles diferentes tendrán un coste diferente. El problema es entonces ¿cómo encontrar el árbol de coste total **mínimo**?

Una manera de encontrar la solución al problema del árbol de coste total **mínimo**, es la enumeración completa. Aunque esta forma de resolución es eficaz, no se puede considerar un algoritmo, y además no es nada **eficiente**.

Este problema fue resuelto independientemente por **Dijkstra** (1959), **Kruskal**(1956) y **Prim** (1957) y la existencia de un algoritmo polinomial (que todos ellos demostraron) es una grata sorpresa, debido a que un grafo con N vértices puede llegar a contener **N^N-2** subárboles. A lo largo de la historia se ha hecho un gran esfuerzo para encontrar un algoritmo rápido para este problema. El algoritmo de Kruskal es uno de los más fáciles de entender y probablemente el mejor para resolver problemas a mano.

El **algoritmo** se basa en una propiedad clave de los árboles que permite estar seguros de si un arco debe pertenecer al árbol o no, y usar esta propiedad para seleccionar cada **arco**. Nótese en el algoritmo, que siempre que se añade un arco (u,v), éste será siempre la conexión más corta (menor coste) alcanzable desde el nodo u al resto del grafo G. Así que por definición éste deberá ser parte del árbol.

Este algoritmo es de tipo **greedy**, ya que a cada paso, éste selecciona el arco más barato y lo añade al subgrafo. Este tipo de algoritmos pueden no funcionar para **resolver** otro tipo de problemas, por ejemplo para encontrar la ruta más corta entre los **nodos a y b**.

Para **simplificar**, se asumirán que existe un único árbol de coste total mínimo, aunque en muchos problemas puede existir más de una solución óptima de igual valor total mínimo.

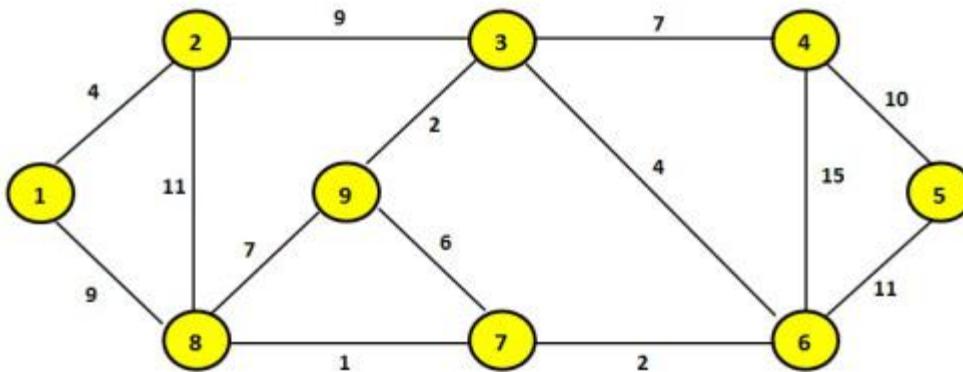
A continuación se muestra el pseudocódigo del algoritmo:

```

Kruskal (G)
E(1)=0, E(2)= todos los Arcos del grafo G
    Mientras E(1) contenga menos de n-1 arcos y E(2)=0 do
        De los arcos de E(2) seleccionar el de menor coste -->e(ij)
        E(2)= E(2) - {e(ij)}
        Si V(i), V(j) no están en el mismo árbol entonces
            juntar los árboles de V(i) y de V(j) en uno sólo
            end Si
    end do
Fin del algoritmo
```

Este problema puede ser resuelto por diferentes algoritmos, e incluso en la actualidad sigue siendo tema de interés en investigaciones.

Ejemplo de árbol de coste total mínimo

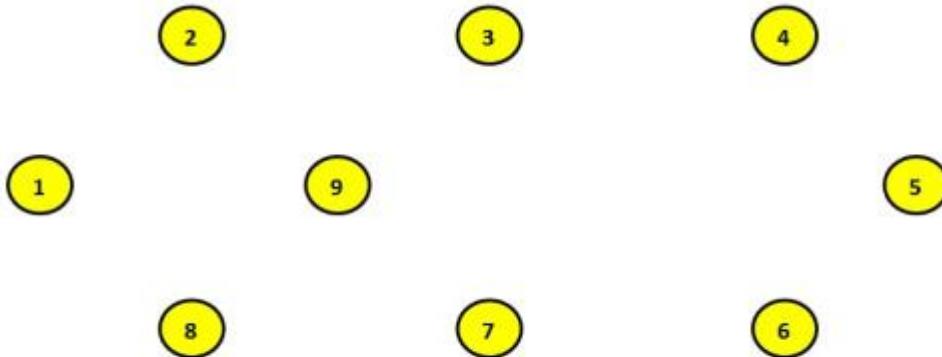


Como podemos ver en la imagen anterior la definición de nuestro grafo en código sería:

```

1 struct Edge{
2     int origen; //Vértice origen
3     int destino; //Vértice destino
4     int peso; //Peso entre el vértice origen y destino
5     Edge(){}
6 ...
7 }arista[ MAX ]; //Arreglo de aristas para el uso en kruskal
```

Primeramente usaremos el método MakeSet de unión-find para inicializar cada componente, obteniendo las siguientes componentes conexas iniciales:



Vértices	1	2	3	4	5	6	7	8	9
Raíz	1	2	3	4	5	6	7	8	9

Ahora el siguiente paso es ordenar las aristas del grafo en orden ascendente:

Vértices de las Aristas	Peso de la Arista
8 – 7	1
3 – 9	2
6 – 7	2
1 – 2	4
3 – 6	4
7 – 9	6
3 – 4	7
8 – 9	7
1 – 8	9
2 – 3	9
4 – 5	10
2 – 8	11
5 – 6	11
4 – 6	15

Para el ordenamiento podemos usar las librerías predefinidas de Java y C++ como estamos ordenando estructuras necesitamos un comparador, en este caso estamos ordenando por peso por lo tanto dentro de la estructura antes definida agregamos:

```

1 struct Edge{
2     ...
3     //Comparador por peso, me servira al momento de ordenar lo realizara en orden ascendente

```

```

4 //Cambiar signo a > para obtener el arbol de expansion maxima
5 bool operator<( const Edge &e ) const {
6     return peso < e.peso;
7 }
8 }arista[ MAX ]; //Arreglo de aristas para el uso en kruskal

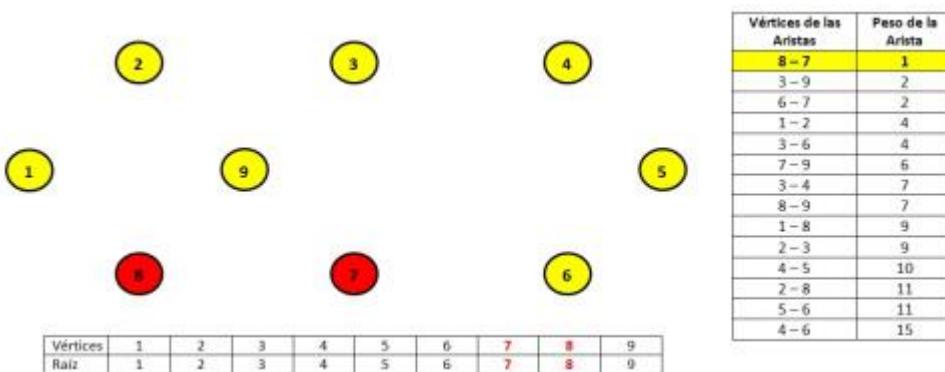
```

Ordenamos el arreglo de aristas mediante lo siguiente:

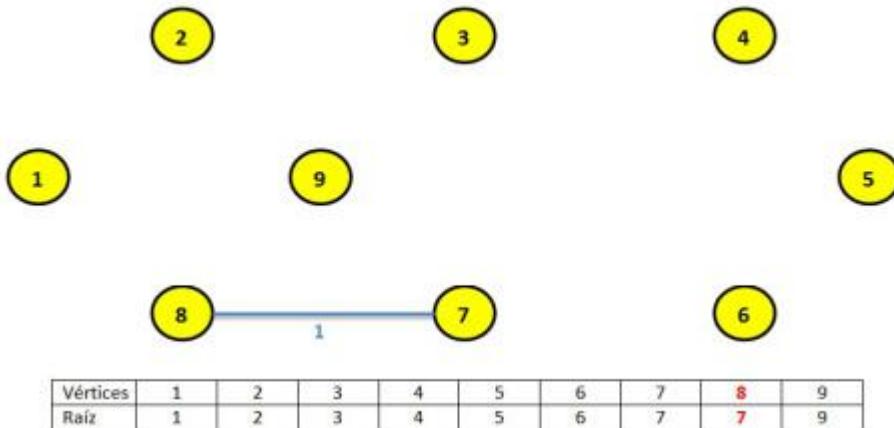
```
1 std::sort( arista , arista + E ); //Ordenamos las aristas por su comparador
```

Lo siguiente será recorrer todas las aristas ya ordenadas y verificar si sus vértices están o no en la misma componente.

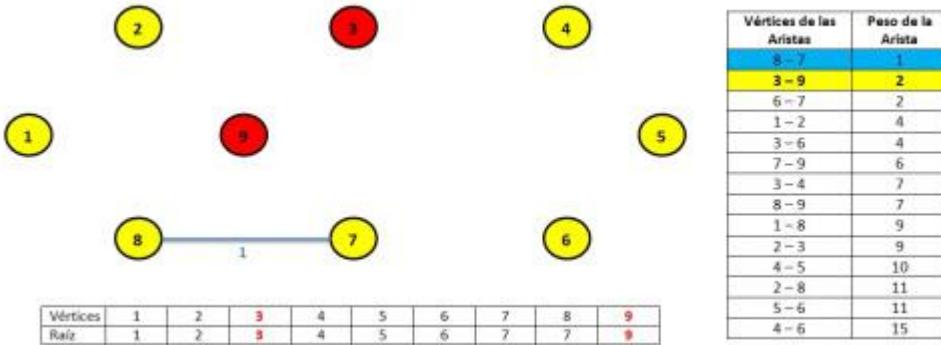
La primera arista a verificar es la que une a los vértices 8 y 7, verificamos si están en la misma componente, para ello hacemos Find(8) , Find(7):



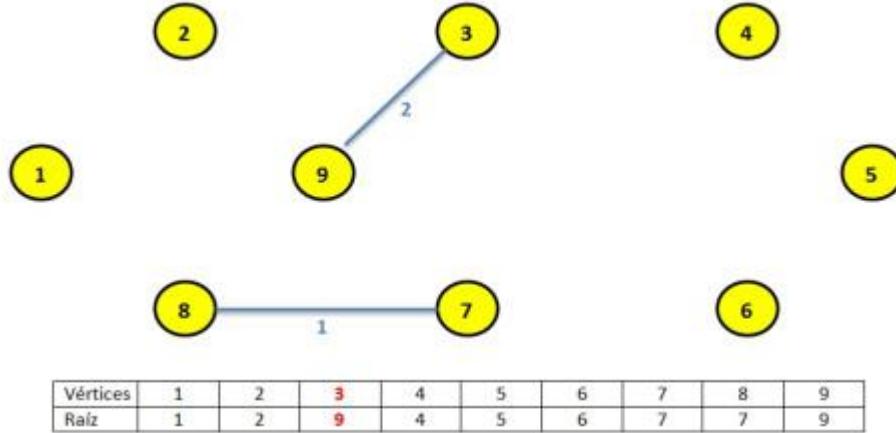
Como podemos observar en la tabla y en la misma imagen no están en la misma componente conexa, por tanto esta arista es valida para el MST así que unimos los vértices por el método de Union(8 , 7).



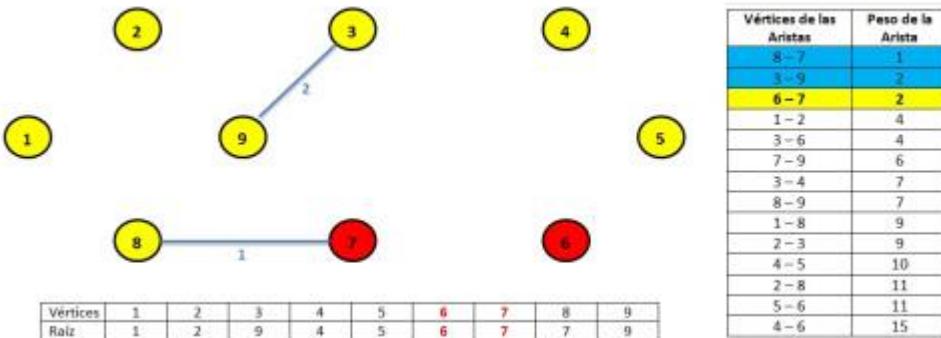
Continuamos con la siguiente arista:



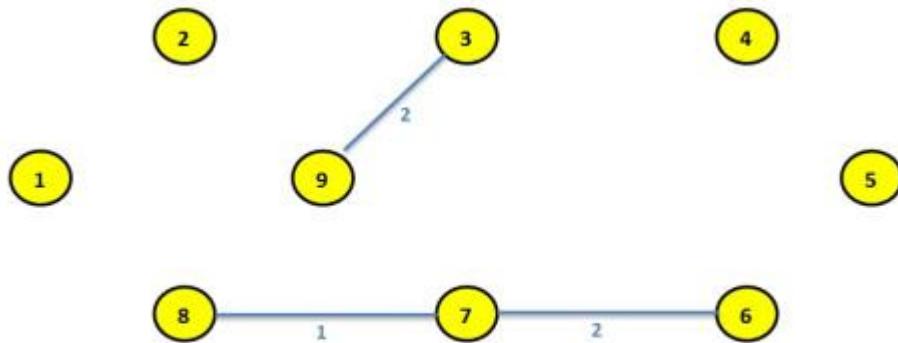
Observamos la tabla de Union-Find y vemos que $\text{Find}(3) \neq \text{Find}(9)$. Entonces es posible realizar la unión de ambas componentes:



Continuamos con la siguiente arista:

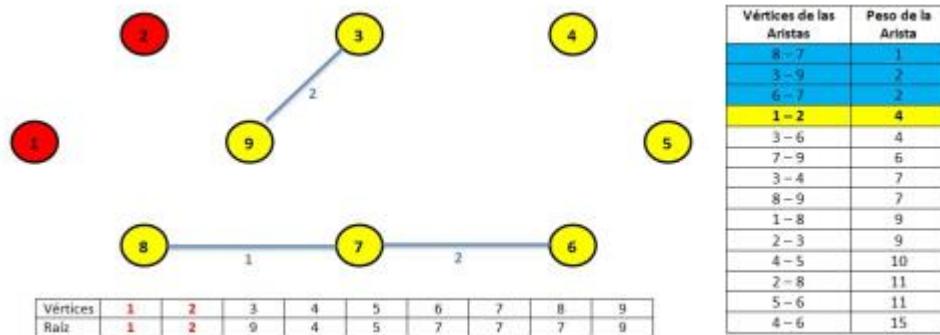


En la imagen podemos observar que ambos vértices no están en la misma componente, por tanto realizamos la Union(6 , 7):

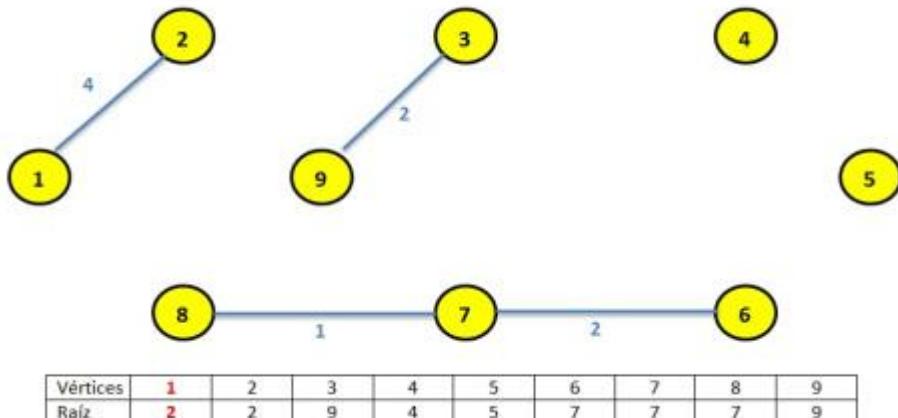


Vértices	1	2	3	4	5	6	7	8	9
Raíz	1	2	9	4	5	7	7	7	9

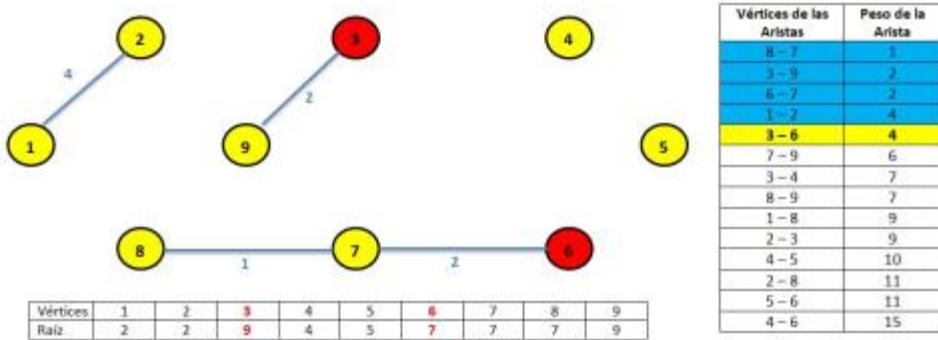
Continuamos con la siguiente arista, los vértices 1 y 2 no están en la misma componente conexa:



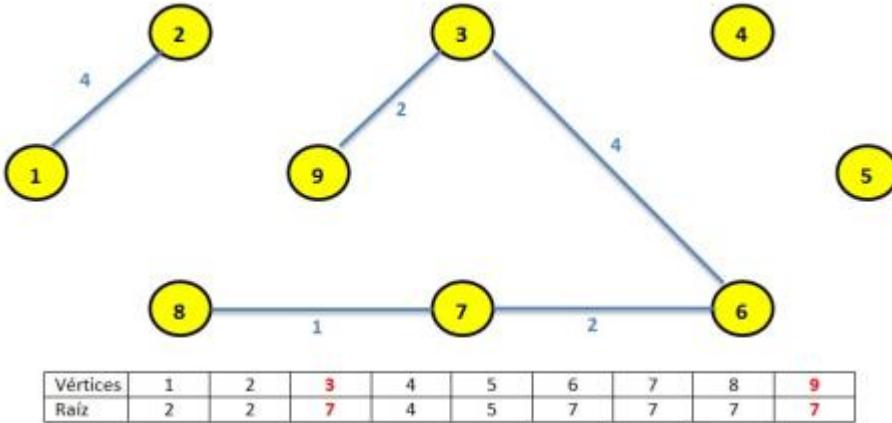
Realizamos la Union(1,2):



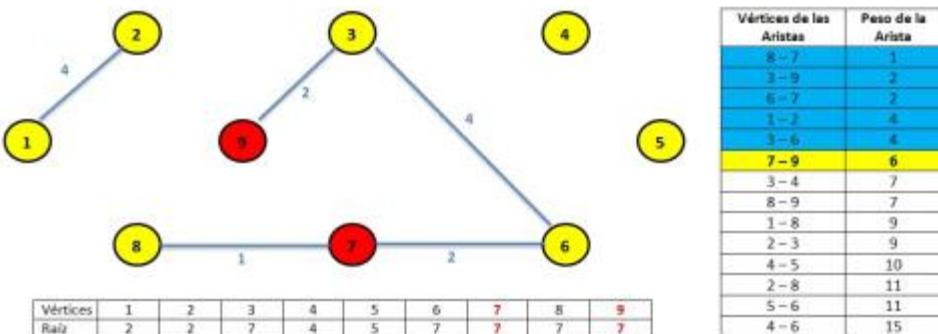
Continuamos con la siguiente arista:



Al observar la imagen los vértices 3 y 6 están en distinta componentes conexas. Entonces realizamos la Union(3,6) y actualizamos la tabla de Union-Find.

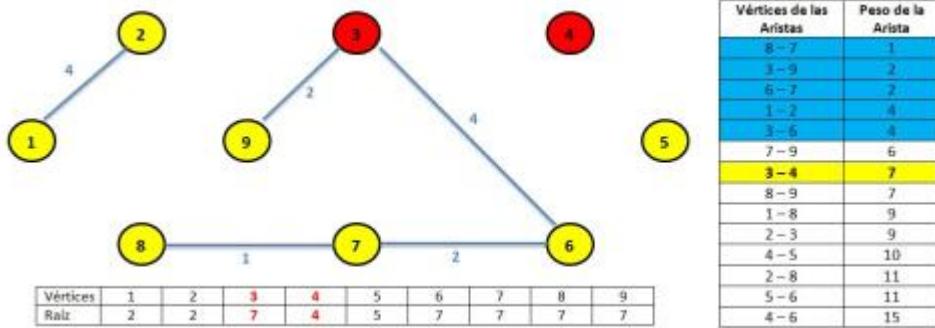


Continuamos con la siguiente arista:

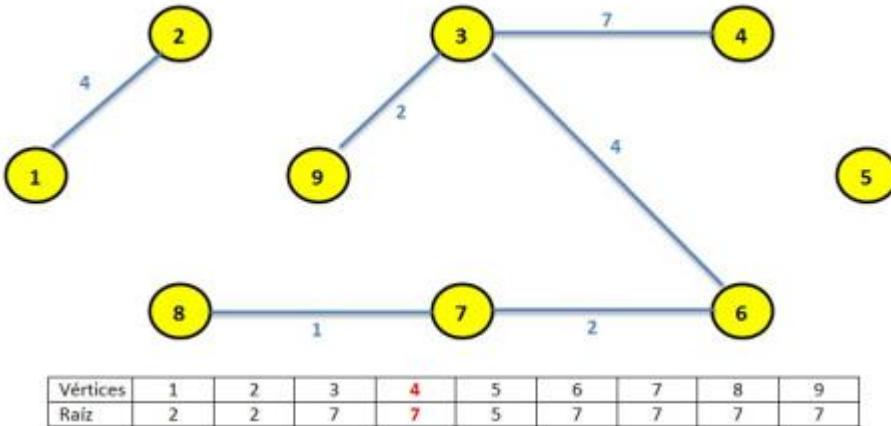


En este caso si observamos la imagen los vértices 7 y 9 están en la misma componente conexa; asimismo en la tabla de Union-Find el elemento raíz del vértice 7 es el mismo que el del vértice 9 por ello afirmamos que están en la misma componente conexa, por lo tanto no habrá que realizar la unión de ambos vértices. Con esto evitamos tener ciclos en el árbol de expansión mínima.

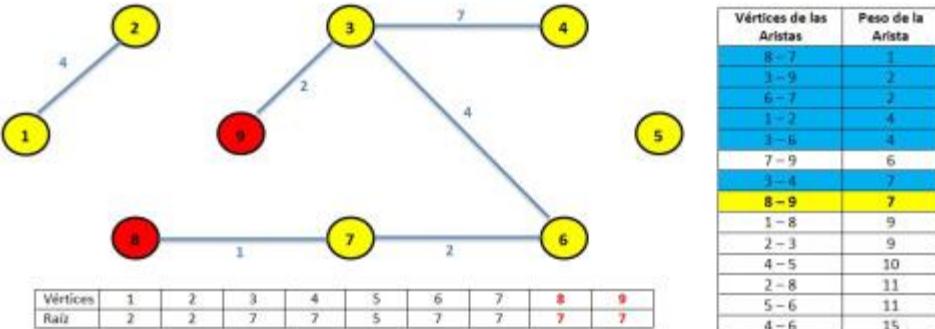
Continuamos con la siguiente arista:



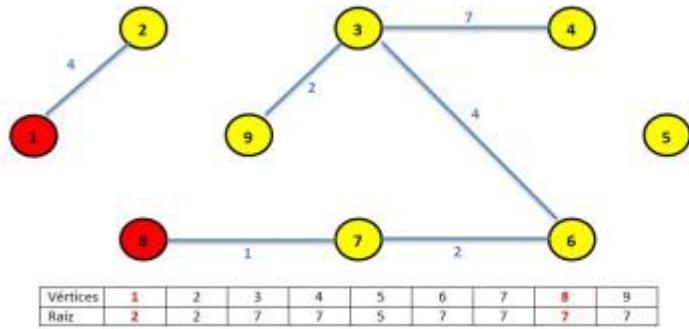
Al observar la imagen los vértices 3 y 4 no están en la misma componente conexa por lo tanto realizamos la Union(3,4) en el grafo:



Continuamos con la siguiente arista:



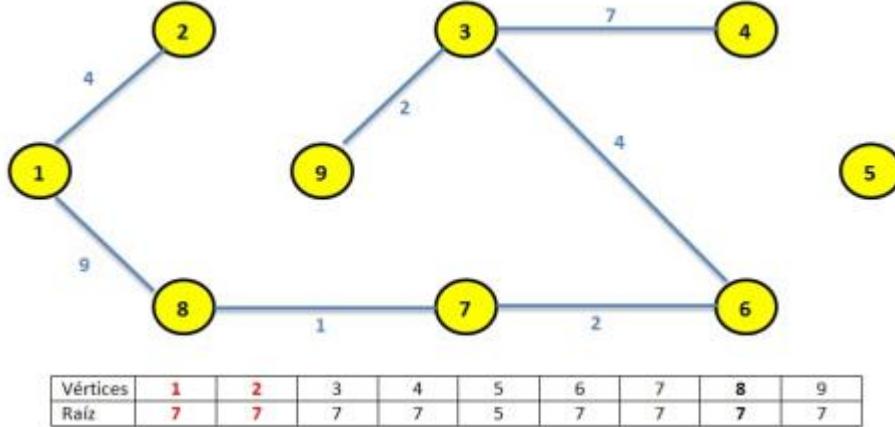
Los vértices 8 y 9 están en la misma componente conexa por lo tanto no realizamos Unión de vértices. Continuemos con la siguiente arista:



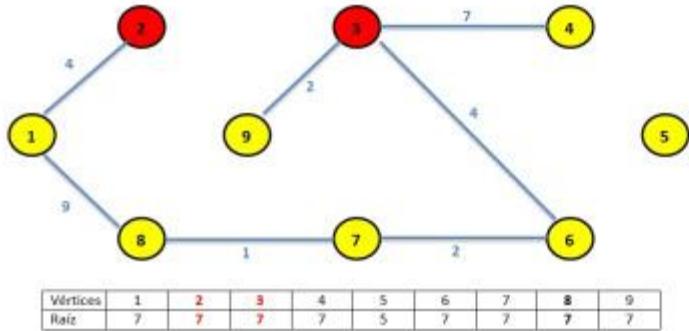
Vértices de las Aristas	Peso de la Arista
8 - 7	1
3 - 9	2
6 - 7	2
1 - 2	4
3 - 6	4
7 - 9	6
3 - 4	7
8 - 9	7
1 - 8	9
2 - 3	9
4 - 5	10
2 - 8	11
5 - 6	11
4 - 6	15

Los vértices

1 y 8 están diferentes componentes. Realizamos la Union(1,8) en el grafo:



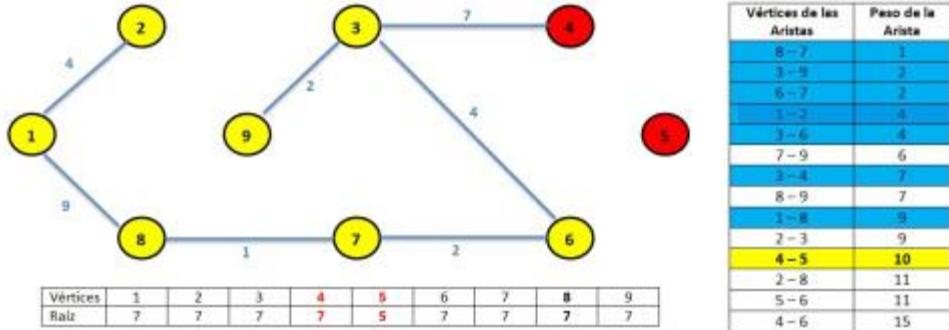
Continuamos con la siguiente arista:



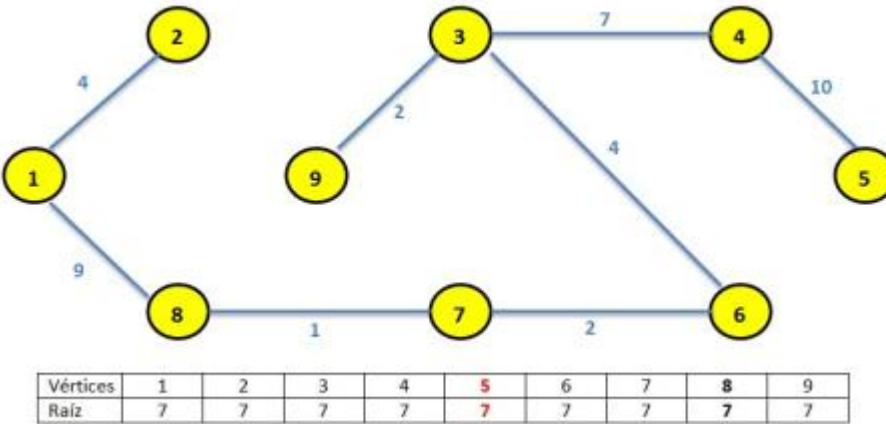
Vértices de las Aristas	Peso de la Arista
8 - 7	1
3 - 9	2
6 - 7	2
1 - 2	4
3 - 6	4
7 - 9	6
3 - 4	7
8 - 9	7
1 - 8	9
2 - 3	9
4 - 5	10
2 - 8	11
5 - 6	11
4 - 6	15

Los vértices

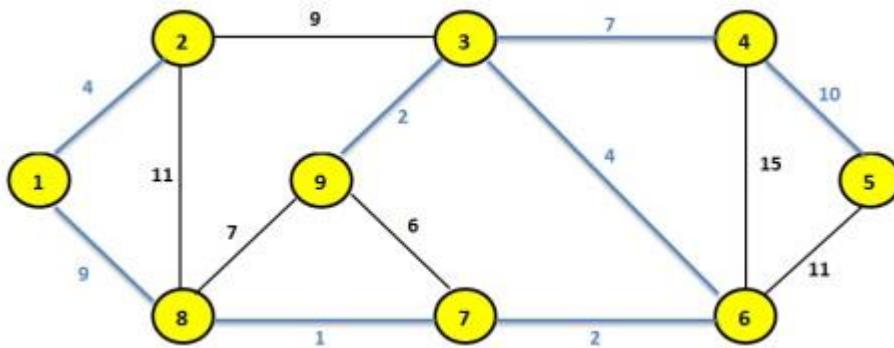
2 y 3 están en la misma componente conexa por lo tanto no realizamos Union de componentes. Continuamos con la siguiente arista:



Los vértices 4 y 7 no están en la misma componente conexa, realizamos Union(4,5) en el grafo:



Como podemos observar ya están todos los vértices del grafo conectados así que al momento de continuar viendo las demás aristas ordenadas siempre tendremos el caso de que ya están en la misma componente conexa por lo tanto el Árbol de Expansión Mínima para el grafo es el siguiente:



El peso total del árbol de expansión mínima para el grafo mostrado es 39.

En código simplemente es iterar sobre el arreglo de aristas ingresado y ordenado obteniendo sus respectivos datos. Para verificar si están o no en la misma componente usamos el método sameComponent explicado en el tutorial de Union-Find:

```

1   for( int i = 0 ; i < E ; ++i ){ //Recorremos las aristas ya ordenadas por peso
2       origen = arista[ i ].origen; //Vértice origen de la arista actual
3       destino = arista[ i ].destino; //Vértice destino de la arista actual
4       peso = arista[ i ].peso; //Peso de la arista actual
5       //Verificamos si estan o no en la misma componente conexa
6       if( !sameComponent( origen , destino ) ){ //Evito ciclos
7           total += peso; //Incremento el peso total del MST
8           MST[ numAristas++ ] = arista[ i ]; //Agrego al MST la arista actual
9           Union( origen , destino ); //Union de ambas componentes en una sola
10      }
11  }

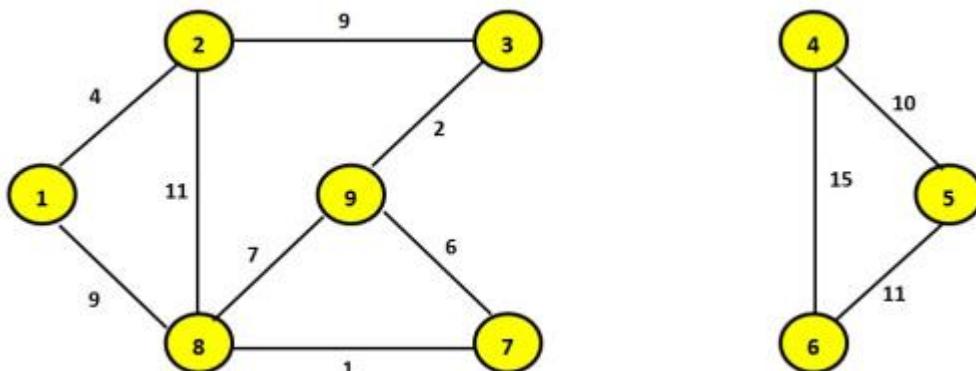
```

Verificación de MST

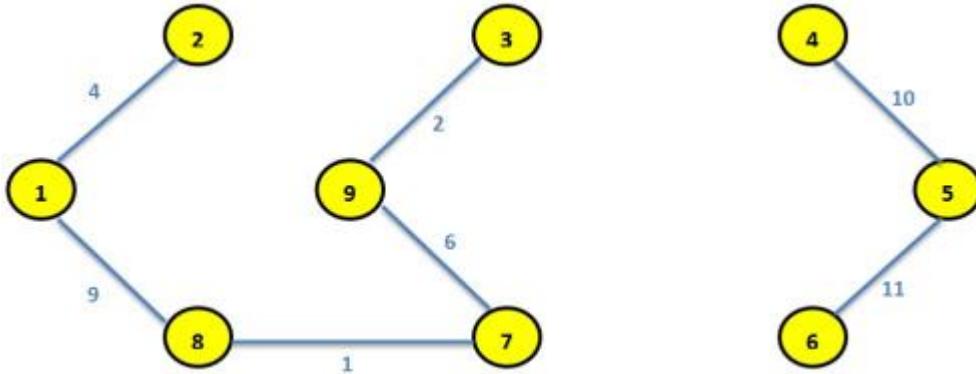
Para que sea un MST válido el número de aristas debe ser igual al número de vértices – 1. Esto se cumple debido a que el MST debe poseer todos los vértices del grafo ingresado y además no deben existir ciclos. Si vemos el ejemplo antes explicado tenemos en el MST:

- Número de Aristas = 8
- Número de Vértices = 9

Cumple con lo dicho $\rightarrow 9 - 1 = 8$ por tanto tenemos un MST válido. Veamos otro ejemplo teniendo como grafo ingresado lo siguiente:



Como podemos observar el grafo ingresado posee 2 componentes conexas, al aplicar kruskal obtendremos los siguientes MST:



En la imagen podemos observar el MST luego de aplicar kruskal, sin embargo no es un MST válido porque no tiene 1 componente conexa que posea todos los vértices, comprobemos:

- Número de Aristas = 7
- Número de Vértices = 9

No cumple lo dicho inicialmente $9 - 1 \neq 7$ por lo tanto tenemos un MST invalido. En código basta con un if:

```

1 //Si el MST encontrado no posee todos los vértices mostramos mensaje de error
2 //Para saber si contiene o no todos los vértices basta con que el numero
3 //de aristas sea igual al número de vertices - 1
4 if( V - 1 != numAristas ){
5     puts("No existe MST valido para el grafo ingresado, el grafo debe ser conexo.");
6     return;
7 }
```

Videos:

<https://www.youtube.com/watch?v=OZKuWP1KxdY>

<https://www.youtube.com/watch?v=EqODdfiqtow>

https://www.youtube.com/watch?v=JyFG_z7iMyo

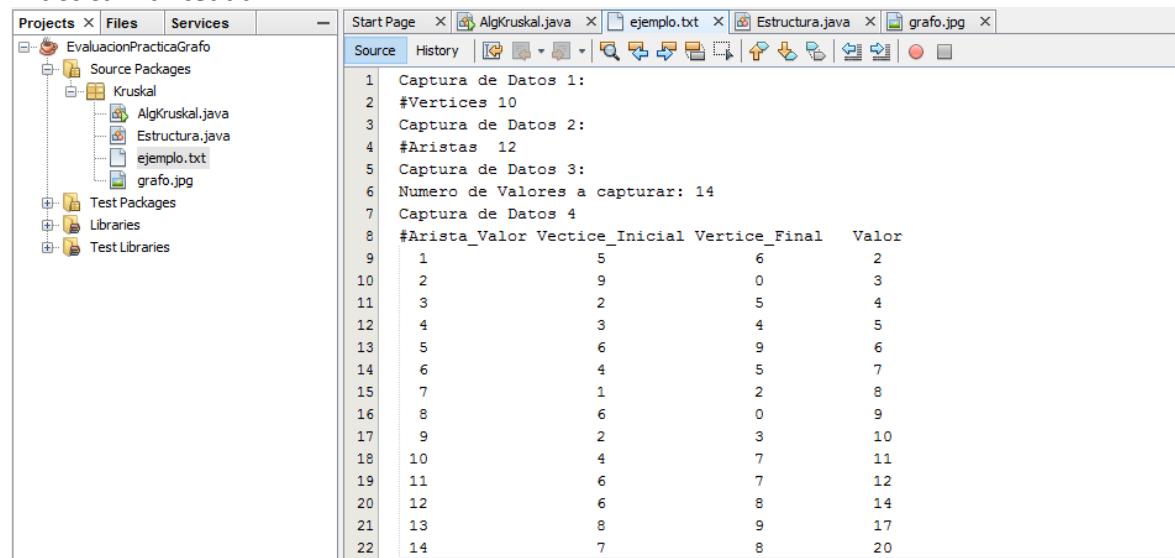
PRACTICA A REALIZAR

Estudiante: Johnathan David Pacheco Pulido

Programa: Ingeniería en Informatica

Espacio Académico: Teoría de Grafos

Práctica Planteada.



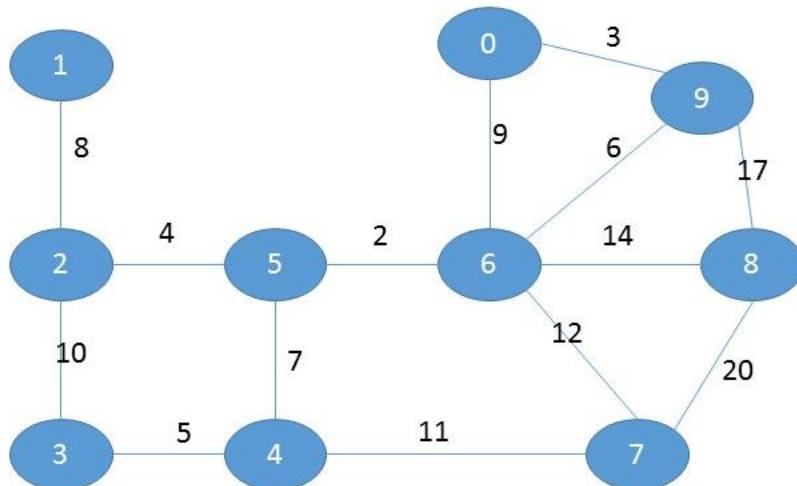
The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "EvaluacionPracticaGrafo". It contains a "Source Packages" folder with a "Kruskal" package containing "AlgKruskal.java", "Estructura.java", and "ejemplo.txt". There are also "Test Packages", "Libraries", and "Test Libraries" sections.
- Code Editor:** The "AlgKruskal.java" file is open. The code reads data from "ejemplo.txt" and prints it to the console. The data is as follows:

	Captura de Datos 1:	Captura de Datos 2:	Captura de Datos 3:	Captura de Datos 4:
	#Vertices 10	#Aristas 12	Numero de Valores a capturar: 14	#Arista_Valor Vertice_Inicial Vertice_Final Valor
1				1 5 6 2
2				2 9 0 3
3				3 2 5 4
4				4 3 4 5
5				5 6 9 6
6				6 4 5 7
7				7 1 2 8
8				8 6 0 9
9				9 2 3 10
10				10 4 7 11
11				11 6 7 12
12				12 6 8 14
13				13 8 9 17
14				14 7 8 20

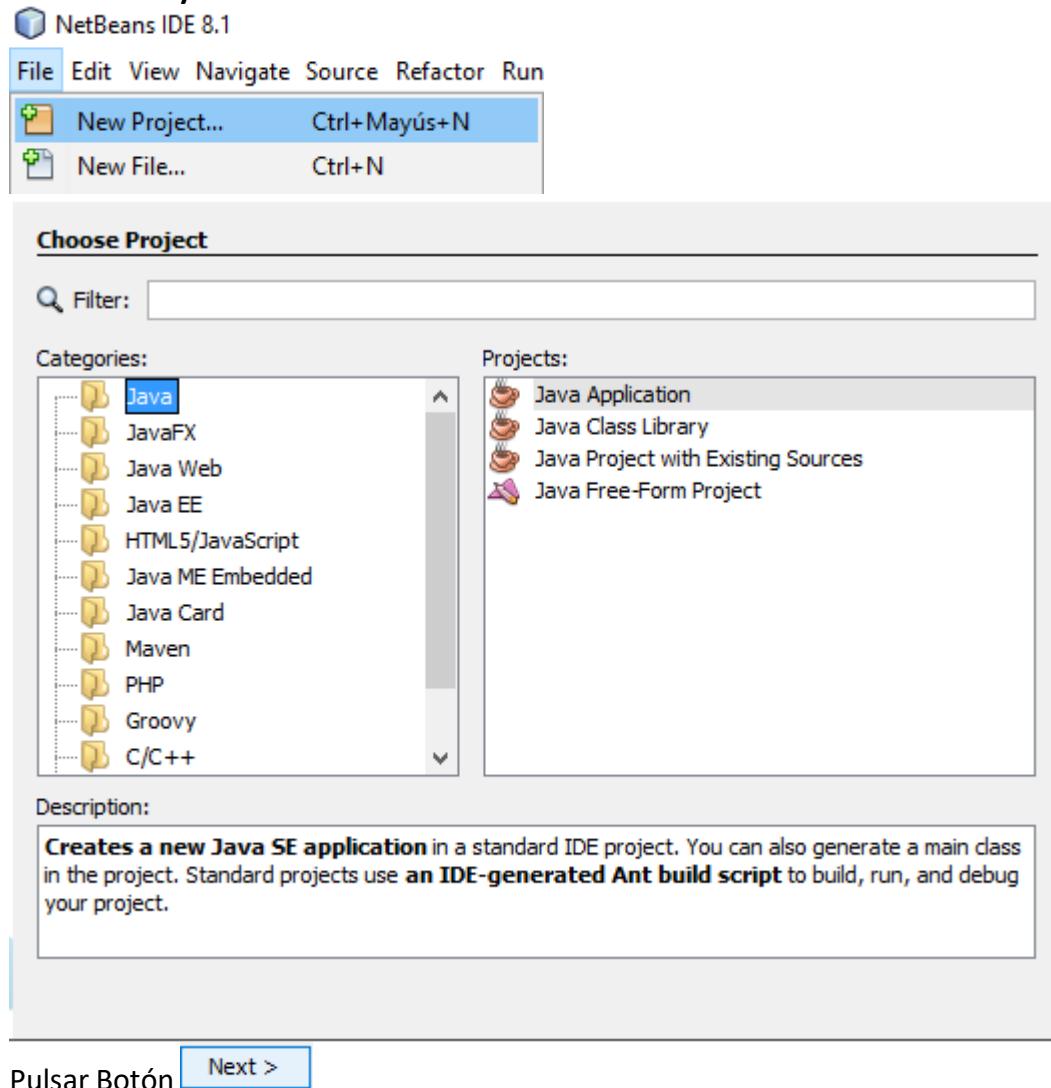
Grafo.jpg

GRAFO PARA IMPLEMENTAR ALGORITMO DE KRUSKAL



PROYECTO NETBEANS

Creación Proyecto



Pulsar Botón Next >

Name and Location

Project Name:

Project Location:

Project Folder:

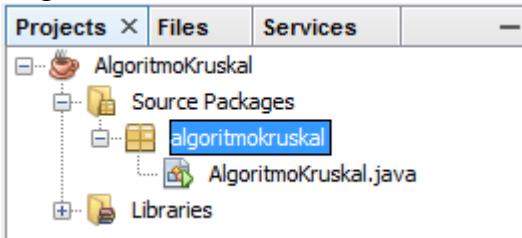
Use Dedicated Folder for Storing Libraries
Libraries Folder:
Different users and projects can share the same compilation libraries (see Help for details).

Create Main Class

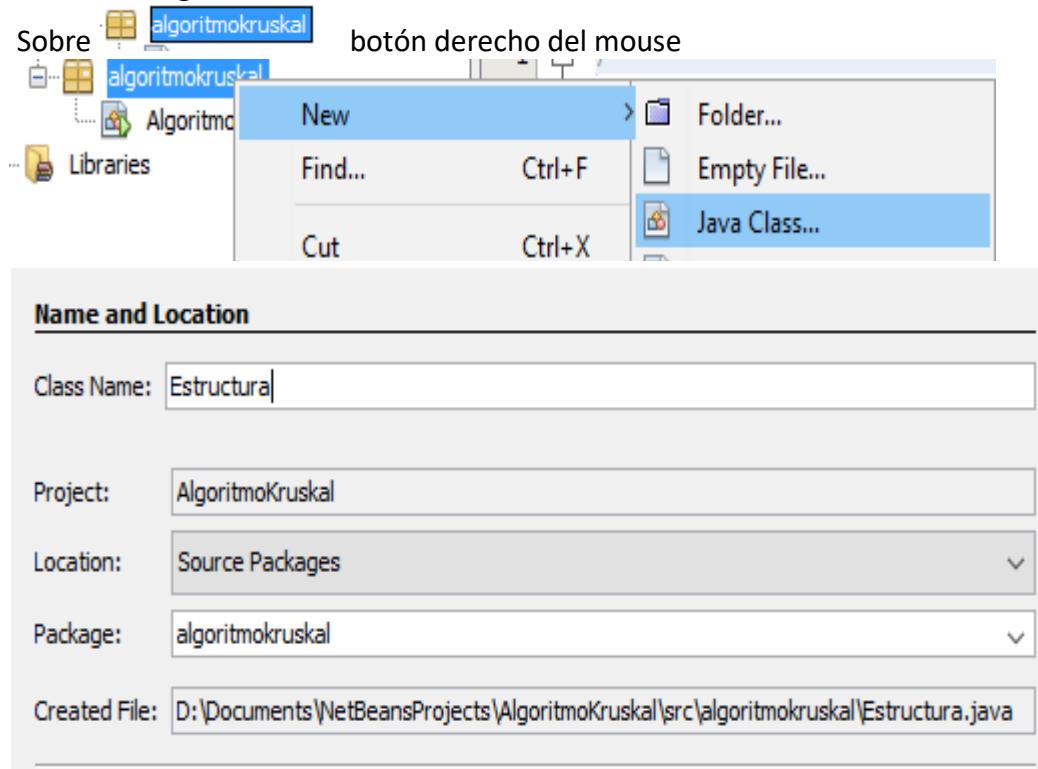
Project Name: AlgoritmoKruskal

Pulsar Botón

Se genera...



Creación Programa Java – Estructura -



Class Name: Estructura

Finish

Pulsar Botón

Reemplazar contenido de Estructura.java por:

The screenshot shows the NetBeans IDE with the 'Estructura.java' file open in the code editor. The code is as follows:

```
1 package algoritmokruskal;
2 public class Estructura
3 {
4     public int arco,a,b;
5     public Estructura(int arc,int a, int b)
6     {
7         this.a =a;
8         this.b =b;
9         this.arco =arc;
10    }
11    public String toString()
12    {
13        return "" + this.arco;
14    }
15 }
```

```

package algoritmokruskal;
public class Estructura
{
    public int arco,a,b;
    public Estructura(int arc,int a, int b)
    {
        this.a =a;
        this.b =b;
        this.arco =arc;
    }
    public String toString()
    {
        return "" + this.arco;
    }
}

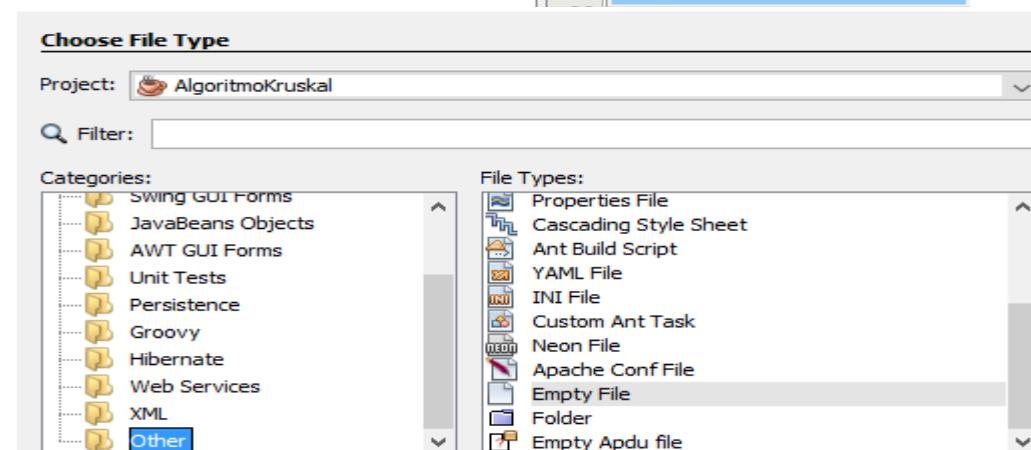
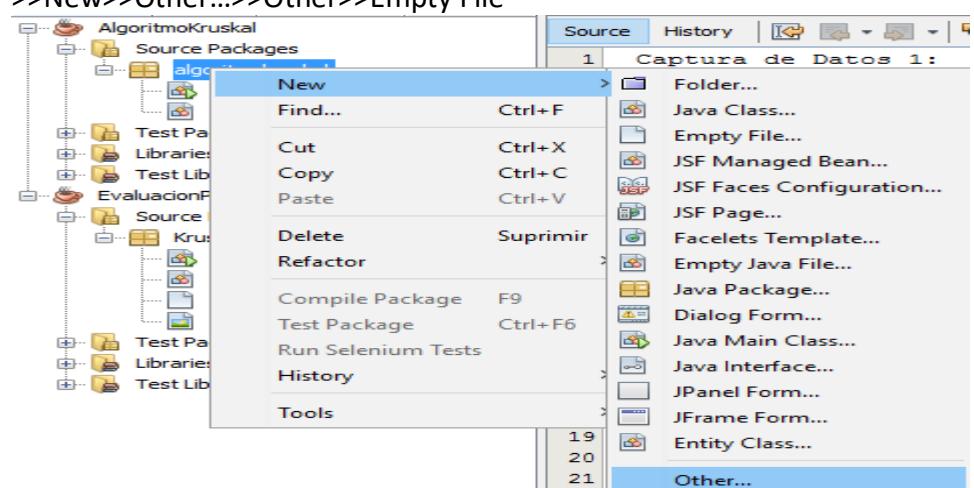
```

Dar click en



Adición Archivo Texto

Sobre **algoritmokruskal** botón derecho del mouse
 >>New>>Other...>>Other>>Empty File



Pulsar botón

File Name: valores.txt

Name and Location

File Name:

Project:

Folder:

Created File:

Pulsar Botón

Reemplazar contenido con:

Captura de Datos 1:

#Vertices 10

Captura de Datos 2:

#Aristas 12

Captura de Datos 3:

Número de Valores a capturar: 14

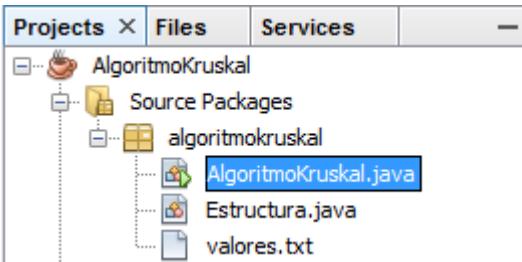
Captura de Datos 4

#Arista_Valor Vertice_Inicial Vertice_Final Valor

1	5	6	2
2	9	0	3
3	2	5	4
4	3	4	5
5	6	9	6
6	4	5	7
7	1	2	8
8	6	0	9
9	2	3	10
10	4	7	11
11	6	7	12
12	6	8	14
13	8	9	17
14	7	8	20

Edición Programa Java – AlgoritmoKruskal –

Seleccionar dando un click....



Reemplazar contenido con:

```
package algoritmokruskal;
import java.util.*;
import javax.swing.*;
public class AlgoritmoKruskal
{
    // Conjuntos de datos de la estructura
    int Nodopadre[] = new int[1000];
    int HallarNodo(int x)
    {
        if(Nodopadre[x] == x)
        {
            return x;
        }
        return HallarNodo(Nodopadre[x]);
    }
    void unir(int x, int y)
    {
        int fx = HallarNodo(x);
        int fy = HallarNodo(y);
        Nodopadre[fx] = fy;
    }
    public static void main(String args[])
    {
        Scanner lector = new Scanner(System.in);
        AlgoritmoKruskal kruskal = new AlgoritmoKruskal();
        // inicializar padres para los conjuntos de datos
        for(int i=0;i<100;i++)
        {
            kruskal.Nodopadre[i]=i;
        }
        // declarando las variables para cargar entrada
        int i,j,n,m,k;
        int a,b,arco;
        ArrayList<Estructura> borde = new ArrayList<Estructura>();
```

```

// cargando la entrada
do
{
n = Integer.parseInt(JOptionPane.showInputDialog(null,"Numero
de Vertices"));
}
while(n<5);
do
{
m = Integer.parseInt(JOptionPane.showInputDialog(null,"Numero
de Aristas"));
}
while(m<n);
int B[][]=new int[n][m];
for(i=0;i<n;i++)
for(j=0;j<m;j++)
B[i][j]=0;
do
{
k = Integer.parseInt(JOptionPane.showInputDialog(null,"Numero de
Valores a Capturar:[ "+n+" ...]"));
}
while(k<=n);
for(i=1;i<=k;i++)
{
do
{
do
{
a = Integer.parseInt(JOptionPane.showInputDialog(null,"Numero
de Vertice Inicial...Captura de Valor Arista:"+i));
}
while(a<0 || a>=n);
do
{
b = Integer.parseInt(JOptionPane.showInputDialog(null,"Numero
de Vertice Final...Captura de Valor Arista:"+i));
}
while(b<0 || b>=n);
}
while(b==a || B[a][b]!=0);
do
{

```

```

arco = Integer.parseInt(JOptionPane.showInputDialog(null,"Valor
Arco...B["+a+"]["+b+"]"));
}
while(arco<=0);
B[a][b]=arco;
borde.add(new Estructura(arco,a,b));
}
// imprimimos una linea para separar la entrada de la salida
System.out.println("\nEl recorrido del grafo es:");
// EMPEZAMOS EL ALGORITMO DE KRUSKAL
// Primero declaramos las variables para el MST(arbol de expansion
minimo)
int mst_peso = 0, mst_borde = 0;
int mst_ni = 0;
// PASO 1: ordenar la lista
// comparador es una interfaz que utiliza para comparar 2
elementos de una colección
Collections.sort(borde, new Comparator<Estructura>()
{
    public int compare(Estructura p1, Estructura p2)
    {
        return p1.arco - p2.arco;
    }
});
// PASO 2-3:
while( ( mst_borde < n-1 ) || ( mst_ni < m ) )
{
    // frenamos el borde en los tres enteros que lo describen
    a = borde.get(mst_ni).a;
    b = borde.get(mst_ni).b;
    arco = borde.get(mst_ni).arco;
    // comprobamos si el borde está bien para ser incluido en el
    arbol de expansion minima
    // si a y b están en árboles diferentes (si están en el mismo
    crearemos un ciclo)
    if( kruskal.HallarNodo(a) != kruskal.HallarNodo(b) )
    {
        // unimos los dos árboles que el borde conecta
        kruskal.unir(a,b);
        // agregamos el peso del borde
        mst_peso += arco;
        // imprimimos el borde y lo contamos
        System.out.println(a + " " + b + " " + arco);
    }
}

```

```

        mst_borde++;
    }
    // aumentar el índice del borde
    mst_ni++;
}
// Presentando el PESO
System.out.println( "\nEl peso del arbol de expansion minima es " +
mst_peso);
}
}

```

```

1 package algoritmokruskal;
2 import java.util.*;
3 import javax.swing.*;
4 public class AlgoritmoKruskal
5 {
6     // Conjuntos de datos de la estructura
7     int Nodopadre[] = new int[1000];
8     int HallarNodo(int x)
9     {
10         if(Nodopadre[x] == x)
11         {
12             return x;
13         }
14         return HallarNodo(Nodopadre[x]);
15     }
16     void unir(int x, int y)
17     {
18         int fx = HallarNodo(x);
19         int fy = HallarNodo(y);
20         Nodopadre[fx] = fy;
21     }
22     public static void main(String args[])

```

Dar click en

Sobre el archivo valores.txt dar un click

#Arista	Valor	Vectice_Inicial	Vertice_Final	Valor
1	5	1	5	2
2	9	2	9	3
3	2	3	2	4
4	3	4	3	5
5	6	5	6	6
6	4	6	4	7
7	1	7	1	8
8	6	8	6	9
9	2	9	2	10
10	4	10	4	11
11	6	11	6	12
12	6	12	6	14
13	8	13	8	17
14	7	14	7	20

Ejecución



Pulsar F5 o Botón

Entrada X

?

Numero de Vertices

Aceptar Cancelar

Entrada X

?

Numero de Aristas

Aceptar Cancelar

Entrada X

?

Numero de Valores a Capturar:[10...]

Aceptar Cancelar

Captura de Valores en este orden:

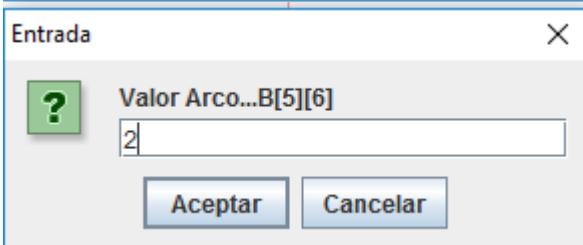
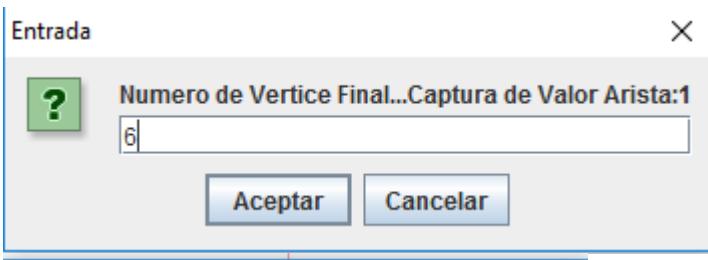
#Arista	Valor	Vectice_Inicial	Vectice_Final	Valor
1	5		6	2
2	9		0	3
3	2		5	4
4	3		4	5
5	6		9	6
6	4		5	7
7	1		2	8
8	6		0	9
9	2		3	10
10	4		7	11
11	6		7	12
12	6		8	14
13	8		9	17
14	7		8	20

Entrada X

?

Numero de Vertice Inicial...Captura de Valor Arista:1

Aceptar Cancelar



.....

Dando como respuesta:

Output

AlgoritmoKruskal (run) #2

```
run:  
  
El recorrido del grafo es:  
5 6 2  
9 0 3  
2 5 4  
3 4 5  
6 9 6  
4 5 7  
1 2 8  
4 7 11  
6 8 14  
  
El peso del arbol de expansion minima es 60  
BUILD SUCCESSFUL (total time: 1 minute 48 seconds)
```

**UNIVERSIDAD SANTO TOMAS
DUAD
FACULTAD DE CYT
PROGRAMA INGENIERIA EN INFORMATICA**

PRACTICA ALGORITMO DE PRIM

**MARIO DUSTANO CONTRERAS CASTRO
DOCENTE TIEMPO COMPLETO**

Algoritmo de Prim

<http://ramonlopez.hol.es/Algoritmo%20de%20Prim.pdf>

Aclaración 1. Un **grafo** es **conexo** si cada par de vértices está conectado por un camino; es decir, si para cualquier par de vértices (a, b), existe al menos un camino posible desde a hacia b.

Aclaración 2. Un **grafo no dirigido** es un **grafo** en el cuál sus aristas son no dirigidas, es decir, no están orientadas en ningún sentido

El algoritmo de Prim es un algoritmo perteneciente a la teoría de los grafos para encontrar un árbol recubridor mínimo en un grafo conexo, no dirigido y cuyas aristas están etiquetadas.

En otras palabras, el algoritmo encuentra un subconjunto de aristas que forman un árbol con todos los vértices, donde el peso total de todas las aristas en el árbol es el mínimo posible. Si el grafo no es conexo, entonces el algoritmo encontrará el árbol recubridor mínimo para uno de los componentes conexos que forman dicho grafo no conexo (sin crear ciclo).

Descripción conceptual

El algoritmo incrementa continuamente el tamaño de un árbol, comenzando por un vértice inicial al que se le van agregando sucesivamente vértices cuya distancia a los anteriores es mínima. Esto significa que en cada paso, las aristas a considerar son aquellas que inciden en vértices que ya pertenecen al árbol.

El árbol recubridor mínimo está completamente construido cuando no quedan más vértices por agregar.

Pseudocódigo del algoritmo

Estructura de datos auxiliar: Cola = Estructura de datos Cola de prioridad

Prim (Grafo G)

// Inicializamos todos los nodos del grafo. La distancia la ponemos a infinito y el padre de cada nodo a NULL

// Se adiciona, en una cola de prioridad donde la prioridad es la distancia, todas las parejas <nodo,distancia> del grafo por cada u en V[G] hacer

distancia[u] = INFINITO

padre[u] = NULL

Añadir(cola,<u,distancia[u]>)

distancia[u]=0

mientras !esta_vacia(cola) hacer

// OJO: Se entiende por mayor prioridad aquel nodo cuya distancia[u] es menor.

u = extraer_minimo(cola) //devuelve el mínimo y lo elimina de la cola.

por cada v adyacente a 'u' hacer

si ((v ∈ cola) && (distancia[v] > peso(u, v))) entonces

padre[v] = u

distancia[v] = peso(u, v)

Actualizar(cola,<v,distancia[v]>)

Otra fuente:

<https://www.youtube.com/watch?v=RXL8Z-HfdHQ>

<https://www.youtube.com/watch?v=SEsyfJHJOFw>

<https://www.youtube.com/watch?v=O8XEOz8FCDQ>

PRIMER MOMENTO. INGRESO A NETBEANS

Practica desarrollada por los estudiantes:

Jerson Rodríguez

Yilver Vargas

Estructura de Datos

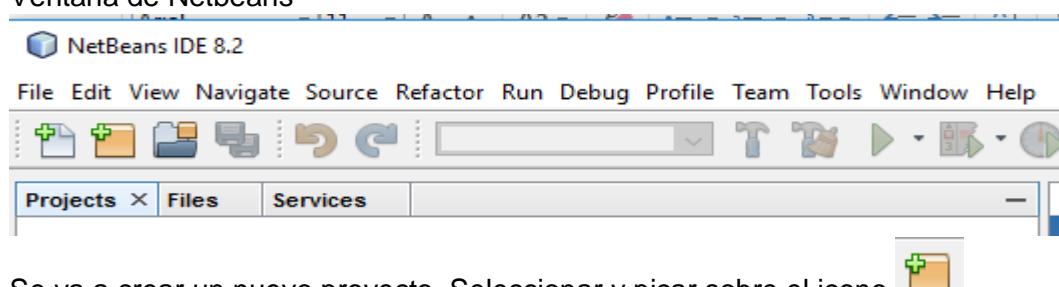
Programa Ingeniería de Sistemas

Universidad Autónoma de Colombia

2017/2

Menú Inicio>>Todas las Aplicaciones>>Netbeans>>Netbeans 8.xx

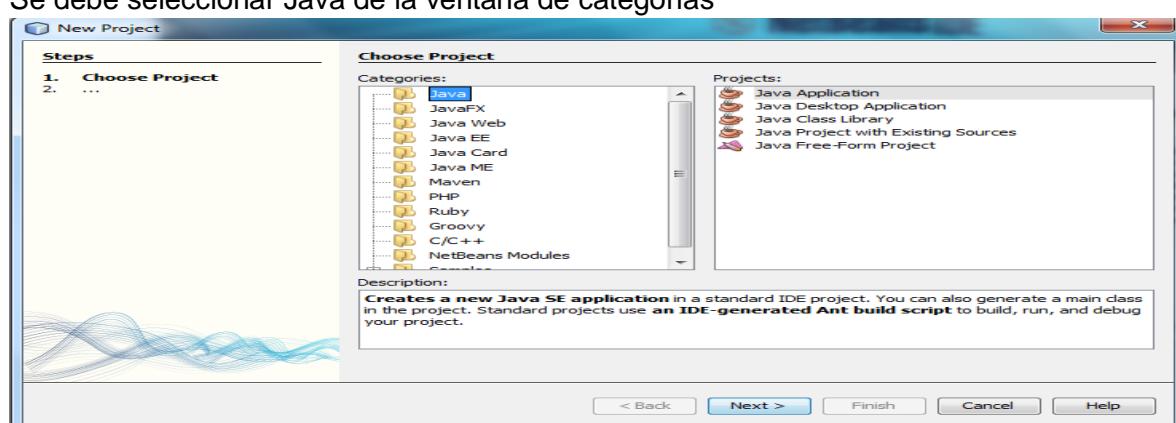
Ventana de Netbeans



Se va a crear un nuevo proyecto. Seleccionar y picar sobre el icono

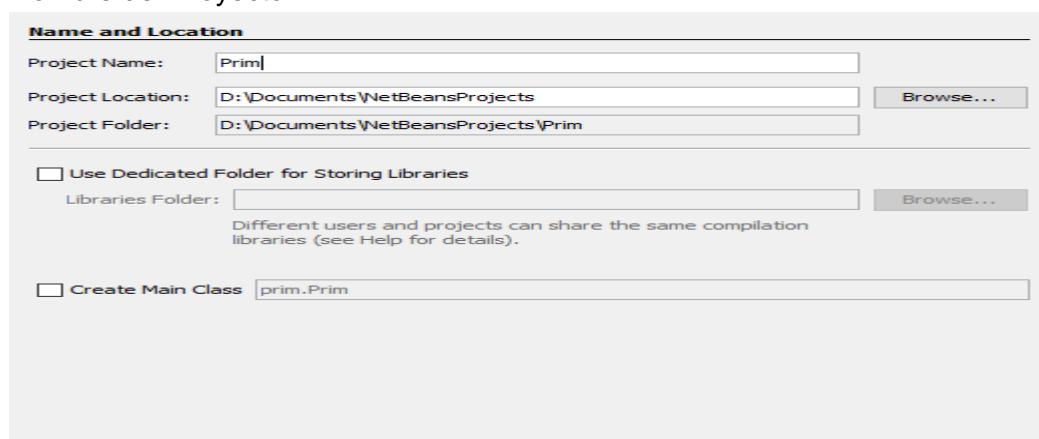


Se debe seleccionar Java de la ventana de categorías

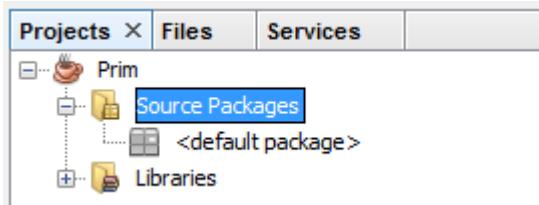


Ahora, se selecciona Java Application de tipo de proyecto. Pulsar el Botón Next

Nombre del Proyecto: Prim



Pulsar el Botón Finish



CREACION DEL PAQUETE Prim

The screenshot shows the NetBeans IDE's Project Explorer with the "Prim" project selected. The "Source Packages" node is expanded, and a context menu is open over it. The "New" option is selected, and a submenu is displayed with "Java Package..." highlighted. Below the submenu, the "Name and Location" dialog is shown, containing fields for "Package Name" (set to "Prim"), "Project" (set to "Prim"), "Location" (set to "Source Packages"), and "Created Folder" (set to "D:\Documents\NetBeansProjects\Prim\src\Prim").

Name and Location

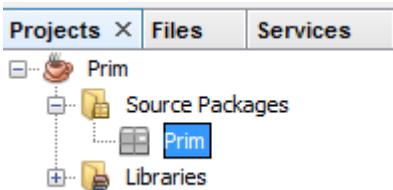
Package Name: Prim

Project: Prim

Location: Source Packages

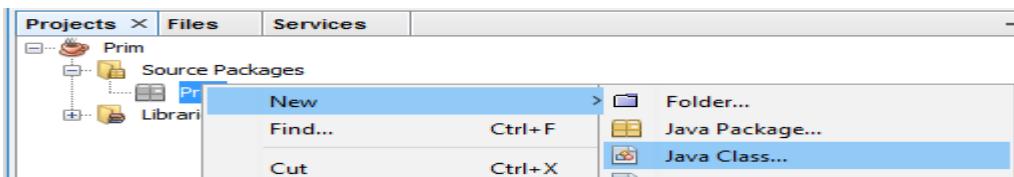
Created Folder: D:\Documents\NetBeansProjects\Prim\src\Prim

Pulsar el Botón Finish

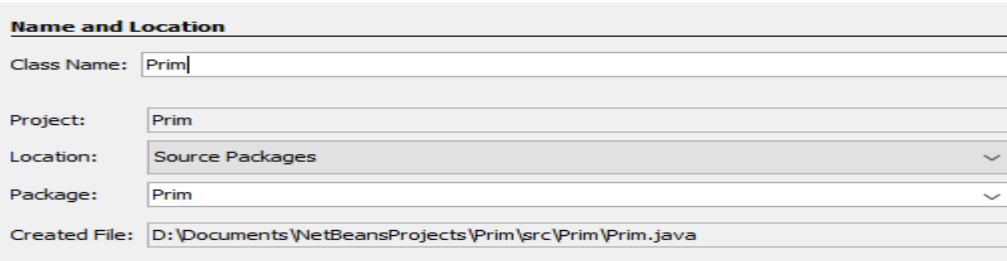


CREACION DE LA CLASE Prim

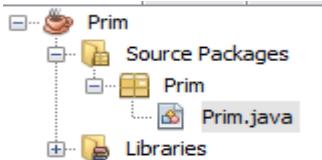
Se debe posicionar sobre el paquete Source Package>>Prim y pulsar el botón derecho del mouse>>opción new>>Java Class



Nombre de la Clase: **Prim**



Pulsar el Botón Finish



Reemplazar el contenido por:

```
package prim;
import java.util.ArrayList;
import java.util.List;
public class Prim{
    public int[][] aplicaPrim(int[][] grafo, int nodos){
        int[][] grafoPrim;
        int tam_grafoPrim=0;
        List<Integer> conjunto = new ArrayList<Integer>();
        conjunto.add(1);
        mergeSort(1,grafo.length,0,grafo);

        do{
            for(int x=0;x<grafo.length;x++){
                if(grafo[x][0]>0){
                    if( (conjunto.indexOf(grafo[x][1])>-1 && conjunto.indexOf(grafo[x][2])<0) ||
                        (conjunto.indexOf(grafo[x][2])>-1 && conjunto.indexOf(grafo[x][1])<0 ) ){
                        conjunto.add(conjunto.indexOf(grafo[x][1])>-1?grafo[x][2]:grafo[x][1]);

                        grafo[x][0]*=-1;
                        tam_grafoPrim++;
                        mergeSort(1,grafo.length,0,grafo);
                        x=grafo.length+1;
                    }
                }
            }
        }
```

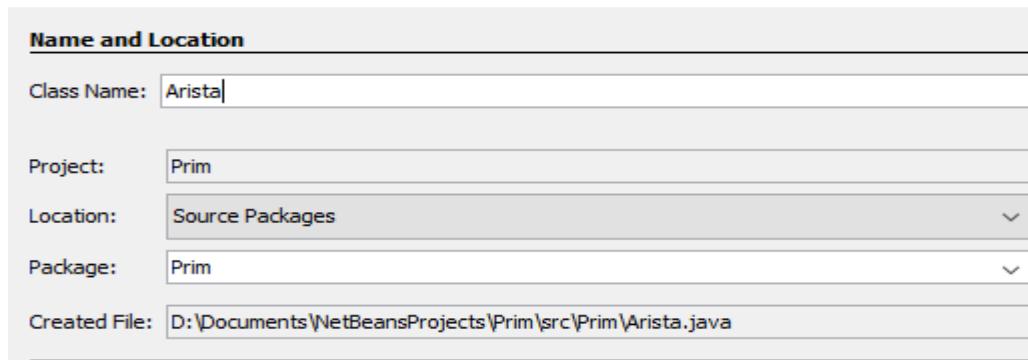
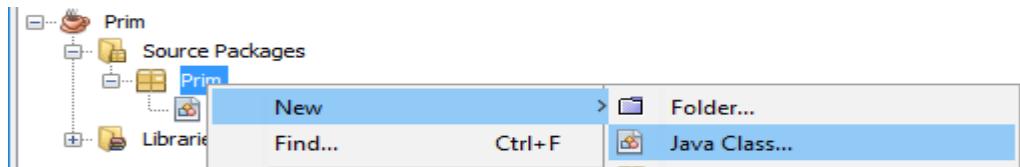
```

    }
} while(conjunto.size()<nodos);
grafoPrim = new int[tam_grafoPrim][3];
for(int x=0;x<tam_grafoPrim;x++){
    grafoPrim[x][0] = grafo[x][0]*-1;
    grafoPrim[x][1] = grafo[x][1];
    grafoPrim[x][2] = grafo[x][2];
}
return grafoPrim;
}
public void mergeSort(int l, int r, int col, int M[][]){
int q;
if(l<r){
    q=(int)(Math.floor((l+r)/2));
    mergeSort(l,q,col,M);
    mergeSort(q+1,r,col,M);
    merge(l,q,r,col,M);
}
}
public void merge(int l, int q, int r, int col, int M[][]){
int i;
int j;
int k = 0;
int n = r-l+1;
int[][] B = new int[n][3];
for(i=l;i<=q;i++){
    B[k][0] = M[i-1][0];
    B[k][1] = M[i-1][1];
    B[k++][2] = M[i-1][2];
}
for(j=r;j>=q+1;j--){
    B[k][0]=M[j-1][0];
    B[k][1]=M[j-1][1];
    B[k++][2]=M[j-1][2];
}
i = 1; j = n; k = l;
while (i<=j)
    if (B[i-1][col]<=B[j-1][col]){
        M[(k)-1][0] = B[(i)-1][0];
        M[(k)-1][1] = B[(i)-1][1];
        M[(k++)-1][2] = B[(i++)-1][2];
    }
    else{
        M[(k)-1][0] = B[(j)-1][0];
        M[(k)-1][1] = B[(j)-1][1];
        M[(k++)-1][2] = B[(j--)1][2];
    }
}
}

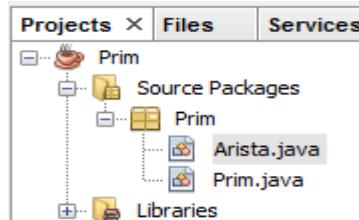
```

CREACION DE LA CLASE Arista

Se debe posiciona sobre el paquete Source Package>>Prim y pulsar el botón derecho del mouse>>opción new>>Java Class



Pulsar el Botón Finish



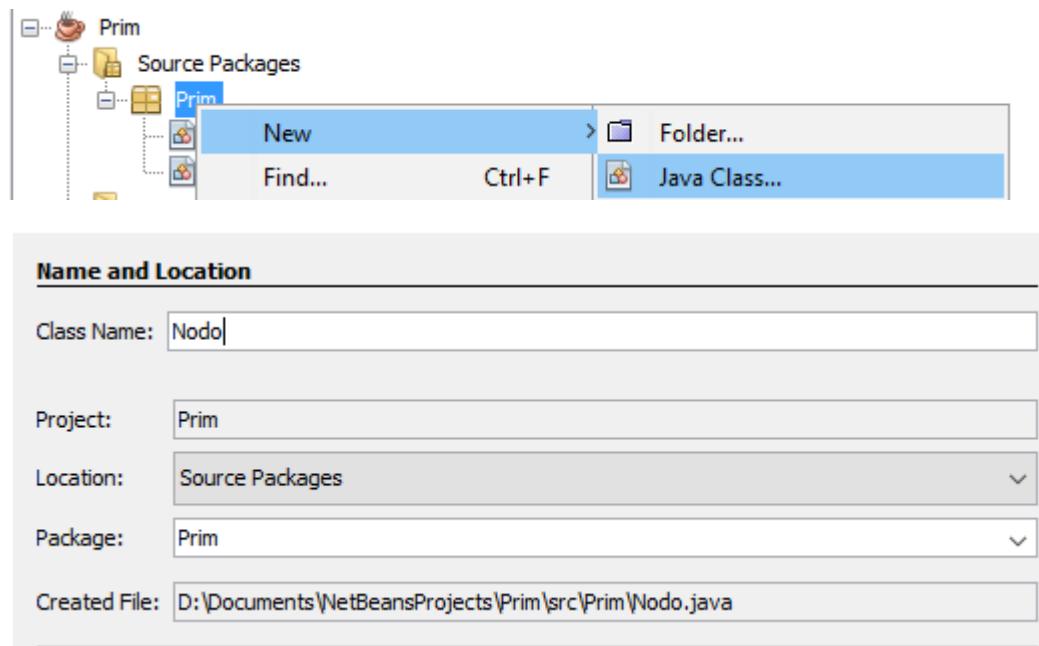
Reemplazar el contenido por:

```
package prim;
public class Arista {
    public char nodo1;
    public char nodo2;
    public int peso;
    public int x1, y1;
    public int x2, y2;
    public boolean verde;
}
```

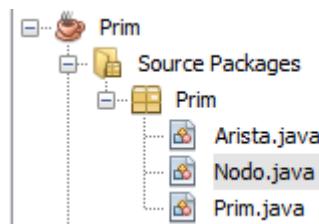


CREACION DE LA CLASE Nodo

Se debe posicionar sobre el paquete Source Package>>Prim y pulsar el botón derecho del mouse>>opción new>>Java Class



Pulsar el Botón Finish



Reemplazar el contenido por:

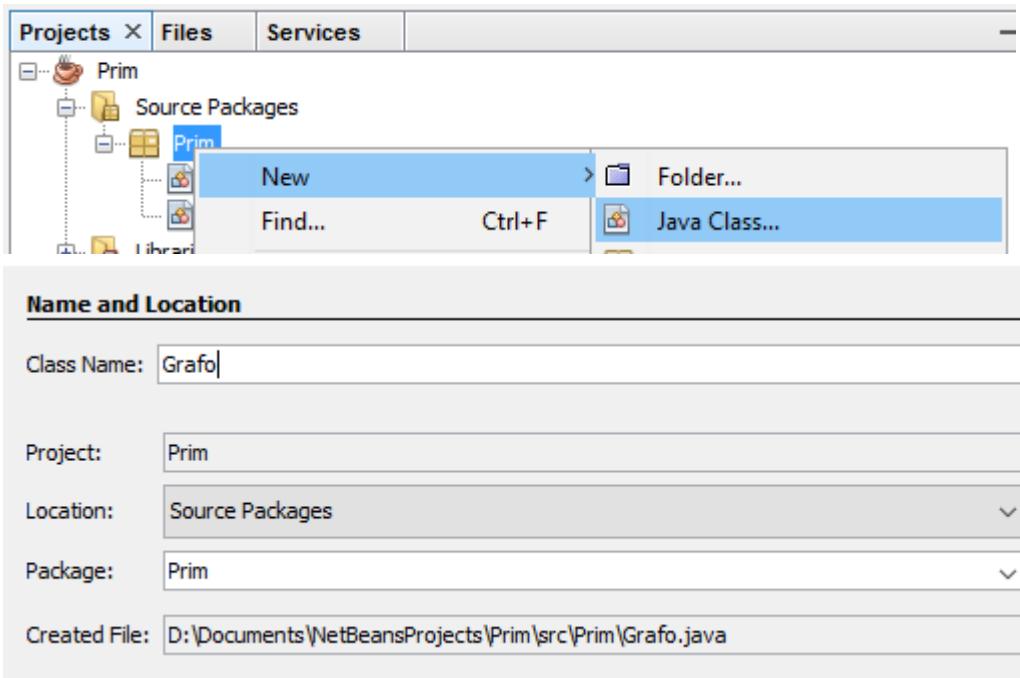
```
package prim;
public class Nodo{
    public int x;
    public int y;
    public int numero;
}
```

The screenshot shows the NetBeans editor with the 'Nodo.java' file open. The code is identical to the one provided in the previous step. The editor interface includes tabs for 'Start Page', 'Prim.java', and 'Arista.java'. The 'Source' tab is selected, showing the code. The code is as follows:

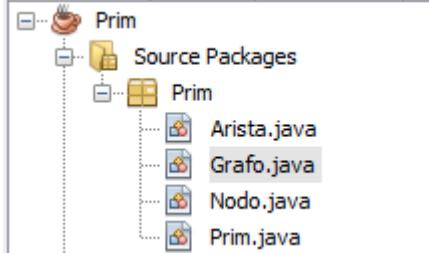
```
1 package prim;
2 public class Nodo{
3     public int x;
4     public int y;
5     public int numero;
6 }
```

CREACION DE LA CLASE Grafo

Se debe posiciona sobre el paquete Source Package>>Prim y pulsar el botón derecho del mouse>>opción new>>Java Class



Pulsar el Botón Finish



Reemplazar el contenido por:

```
package prim;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
public class Grafo extends Panel implements MouseListener, MouseMotionListener{
    private ArrayList<Nodo> lista;
    private ArrayList<Arista> aristas;
    private boolean bandera;
    private int contador;
    private Programa programa;
    public Grafo(Programa programa){
        this.programa=programa;
        lista= new ArrayList<Nodo>();
        aristas= new ArrayList<Arista>();
        contador=65;
        bandera=false;
```

```

        addMouseListener(this);
        addMouseMotionListener(this);
    }
    public void paint(Graphics g){
        g.setColor(Color.WHITE);
        g.fillRect(0,0,getSize().width,getSize().height);
        for(int i=0;i<aristas.size();i++){
            Arista arista = aristas.get(i);
            if(arista.verde) g.setColor(Color.GREEN);
            else g.setColor(Color.BLACK);
            g.drawLine(arista.x1,arista.y1,arista.x2,arista.y2);
            g.setColor(Color.RED);
            g.drawString(""+arista.peso,(arista.x1+arista.x2)/2,(arista.y1+arista.y2)/2);
        }
        for(int i=0;i<lista.size();i++){
            g.setColor(Color.BLACK);
            Nodo nodot=lista.get(i);
            g.fillOval(nodot.x-15,nodot.y-15,30,30);
            g.setColor(Color.WHITE);
            g.drawString(""+(char)nodot.numero,nodot.x-7,nodot.y+5);
        }
    }
    public void nuevo(){
        Nodo nodo=new Nodo();
        nodo.numero=contador;
        contador++;
        bandera=true;
        lista.add(nodo);
    }
    public void limpia(){
        lista.clear();
        aristas.clear();
        contador=65;
        bandera=false;
        repaint();
    }
    public ArrayList<Nodo> nodos(){
        return lista;
    }
    public void nuevaarista(char n1, char n2, int peso){
        Arista arista;
        for(int i=0;i<aristas.size();i++){
            arista = aristas.get(i);
            if((arista.nodo1==n1 && arista.nodo2==n2)|| (arista.nodo2==n1 && arista.nodo1==n2)){ 
                programa.muestramensaje("Ya existe la arista");
                return;
            }
        }
        arista = new Arista();
        arista.nodo1=n1;
        arista.nodo2=n2;
    }
}

```

```

arista.peso=peso;
arista.verde=false;
for(int i=0;i<lista.size();i++){
    Nodo nodot=lista.get(i);
    if((char)nodot.numero==n1){
        arista.x1=nodot.x;
        arista.y1=nodot.y;
    }
    else if((char)nodot.numero==n2){
        arista.x2=nodot.x;
        arista.y2=nodot.y;
    }
}
aristas.add(arista);
repaint();
}

public void prim(){
    Prim prim = new Prim();
    int aux=0, nodOrig=0, veces, veces2=0, nodDest;
    int[][] colaristas;
    int[][] recubridor;
    Arista ari;
    if(aristas.size()<1){
        programa.muestramensaje("No hay aristas");
        return;
    }
    for(int j=1;j<=lista.size();j++) aux+=lista.size()-j;
    colaristas=new int[aux][3];
    veces=lista.size()-1;
    for(int k=1;k<=aux;k++){
        if(veces2++==veces){
            nodOrig++;
            veces--;
            veces2=1;
        }
        nodDest = nodOrig+veces2;
        colaristas[k-1][0]=0;
        for(int y=0;y<aristas.size();y++){
            ari=aristas.get(y);
            if(nodOrig+65==ari.nodo1&&nodDest+65==ari.nodo2){
                colaristas[k-1][0]=ari.peso;
            }
            else if(nodOrig+65==ari.nodo2&&nodDest+65==ari.nodo1){
                colaristas[k-1][0]=ari.peso;
            }
        }
        colaristas[k-1][1]=nodOrig;
        colaristas[k-1][2]=nodDest;
    }
    recubridor=prim.aplicaPrim(colaristas,lista.size());
    veces=0;
}

```

```

        for(int y=0;y<aristas.size();y++) aristas.get(y).verde=false;
        for(int l=0;l<recubridor.length;l++){
            for(int y=0;y<aristas.size();y++){
                ari=aristas.get(y);
                if(ari.verde) continue;
                if(recubridor[l][1]+65==ari.nodo1&&recubridor[l][2]+65==ari.nodo2) ari.verde=true;
                if(recubridor[l][1]+65==ari.nodo2&&recubridor[l][2]+65==ari.nodo1) ari.verde=true;
            }
            veces+=recubridor[l][0];
        }
        repaint();
        programa.muestramensaje("Costo total: "+veces);
    }

    public void mouseClicked(MouseEvent e){
        if(bandera){
            bandera=false;
            programa.botones(contador<91,true,true);
            repaint();
        }
    }
    public void mouseEntered(MouseEvent e){}
    public void mouseExited(MouseEvent e){}
    public void mousePressed(MouseEvent e){}
    public void mouseReleased(MouseEvent e){}

    public void mouseDragged(MouseEvent e){}
    public void mouseMoved(MouseEvent e){
        if(bandera){
            Nodo nodo=list.list.size()-1;
            nodo.x=e.getX();
            nodo.y=e.getY();
            repaint();
        }
    }
}

```

Quedando:

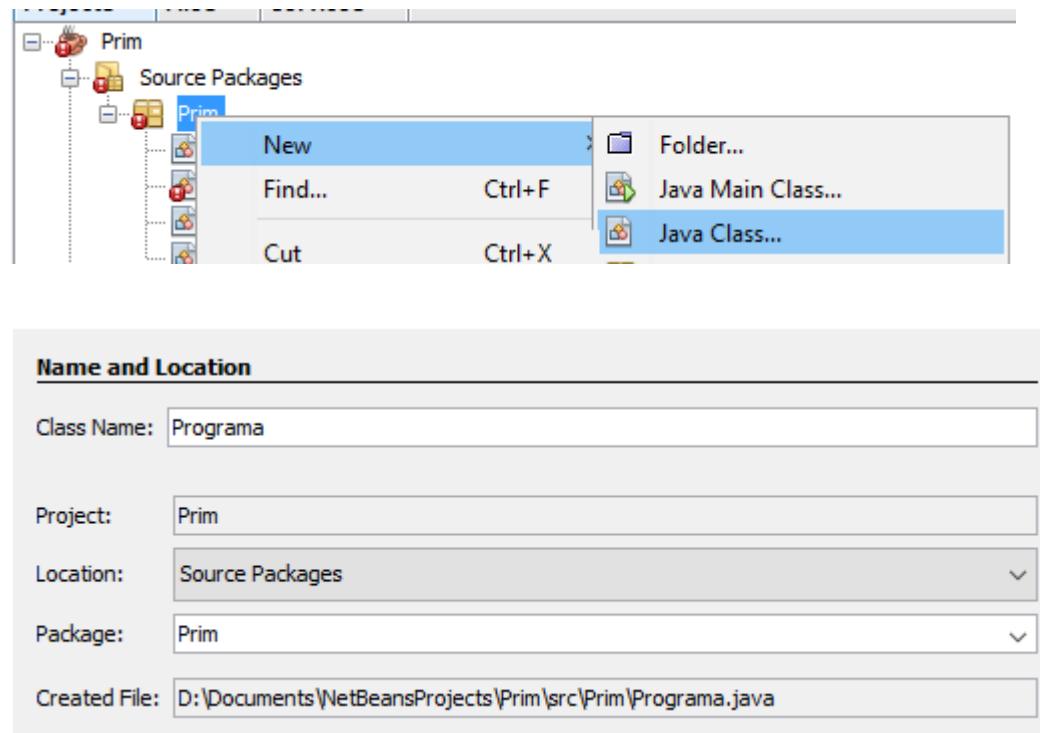
```

Source History 
1 package prim;
2 import java.awt.*;
3 import java.awt.event.*;
4 import java.util.ArrayList;
5 public class Grafo extends Panel implements MouseListener, MouseMotionListener{
6     private ArrayList<Nodo> lista;
7     private ArrayList<Arista> aristas;
8     private boolean bandera;
9     private int contador;
10    private Programa programa;

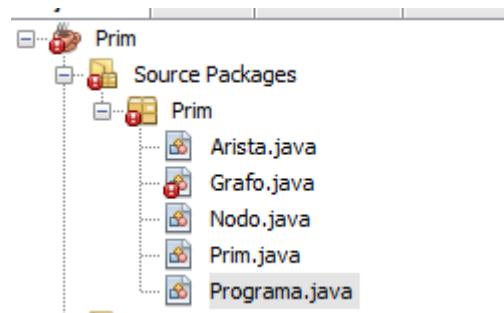
```

CREACION DE LA CLASE Programa

Se debe posicionar sobre el paquete Source Package>>Prim y pulsar el botón derecho del mouse>>opción new>>Java Class...



Pulsar el Botón Finish



Reemplazar el contenido por:

```
package prim;
import java.awt.*;
import java.awt.event.*;
import java.awt.Dialog;
import java.util.ArrayList;
public class Programa extends Frame implements ActionListener{
    private Button nuevo;
    private Button arista;
    private Button limpia;
    private Button prim;
    private Grafo grafo;
    private Dialog aristan;
    private List nodos1;
    private List nodos2;
    private TextField peso;
    private Button aristac;
    private Dialog mensaje;
    private Label etiqueta;
    public Programa(){
        Panel panel=new Panel(new GridLayout(4,0));
        grafo = new Grafo(this);
        nuevo = new Button("Nuevo");
        nuevo.addActionListener(this);
        nuevo.setActionCommand("nuevo");
        arista = new Button("Arista");
        arista.addActionListener(this);
        arista.setActionCommand("arista");
        prim = new Button("Prim");
        prim.addActionListener(this);
        prim.setActionCommand("prim");
        limpia = new Button("Limpiar");
        limpia.addActionListener(this);
        limpia.setActionCommand("limpia");
        panel.add(nuevo);
        panel.add(arista);
        panel.add(limpia);
        panel.add(prim);
        add(panel,BorderLayout.WEST);
        add(grafico,BorderLayout.CENTER);
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
        aristan=new Dialog(this,"Aregar arista",true);
        aristan.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                aristan.setVisible(false);
            }
        });
    }
}
```

```

aristan.setLayout(new GridLayout(2,4));
nodos1=new List();
nodos2=new List();
peso=new TextField();
aristac = new Button("Aregar");
aristac.addActionListener(this);
aristac.setActionCommand("aristac");
aristan.add(new Label("Desde nodo:"));
aristan.add(nodos1);
aristan.add(new Label("Hasta nodo:"));
aristan.add(nodos2);
aristan.add(new Label("Peso:"));
aristan.add(peso);
aristan.add(aristac);
mensaje=new Dialog(this,"Mensaje",true);
mensaje.addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent e){
        mensaje.setVisible(false);
    }
});
etiqueta = new Label("");
mensaje.add(etiqueta);
setTitle("Prim");
setSize(600,450);
setVisible(true);
 setLocationRelativeTo(null);
}

public void actionPerformed(ActionEvent e){
String accion=e.getActionCommand();
if(accion.equals("nuevo")){
    grafo.nuevo();
    botones(false,false,false);
}
else if(accion.equals("arista")){
    ArrayList<Nodo> lista=grafo.nodos();
    nodos1.removeAll();
    nodos2.removeAll();
    for(int i=0;i<lista.size();i++){
        Nodo nodot=lista.get(i);
        nodos1.add(""+(char)nodot.numero);
        nodos2.add(""+(char)nodot.numero);
    }
    aristan.pack();
    peso.setText("");
    aristan.setLocationRelativeTo(this);
    aristan.setVisible(true);
}
else if(accion.equals("aristac")){
    creaarista();
    aristan.setVisible(false);
}
}

```

```

        }
        else if(accion.equals("prim")) grafo.prim();
        else if(accion.equals("limpia")) grafo.limpia();
    }

private void creaarista(){
    String nodo1=nodos1.getSelectedItem();
    String nodo2=nodos2.getSelectedItem();
    int valpeso=0;
    if(nodo1==null||nodo2==null){
        muestramensaje("Nodos seleccionados invalidos");
        return;
    }
    else if(nodo1.equals(nodo2)){
        muestramensaje("Nodos seleccionados invalidos");
        return;
    }
    try{
        valpeso=Integer.parseInt(peso.getText());
    }
    catch(NumberFormatException e){
        muestramensaje("Peso invalido");
        return;
    }
    if(valpeso<=0){
        muestramensaje("Peso invalido");
        return;
    }
    grafo.nuevaarista(nodo1.charAt(0),nodo2.charAt(0),valpeso);
}

public void muestramensaje(String mens){
    etiqueta.setText(mens);
    mensaje.pack();
    mensaje.setLocationRelativeTo(this);
    mensaje.setVisible(true);
}

public void botones(boolean b1,boolean b2,boolean b3){
    nuevo.setEnabled(b1);
    arista.setEnabled(b2);
    limpia.setEnabled(b3);
}
public static void main(String[] args)
{
    new Programa();
}
}

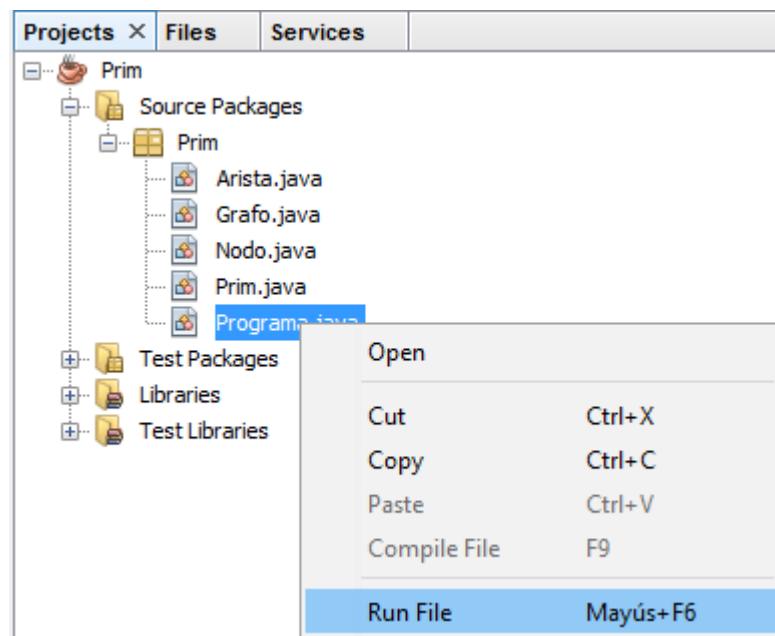
```

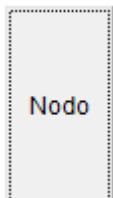
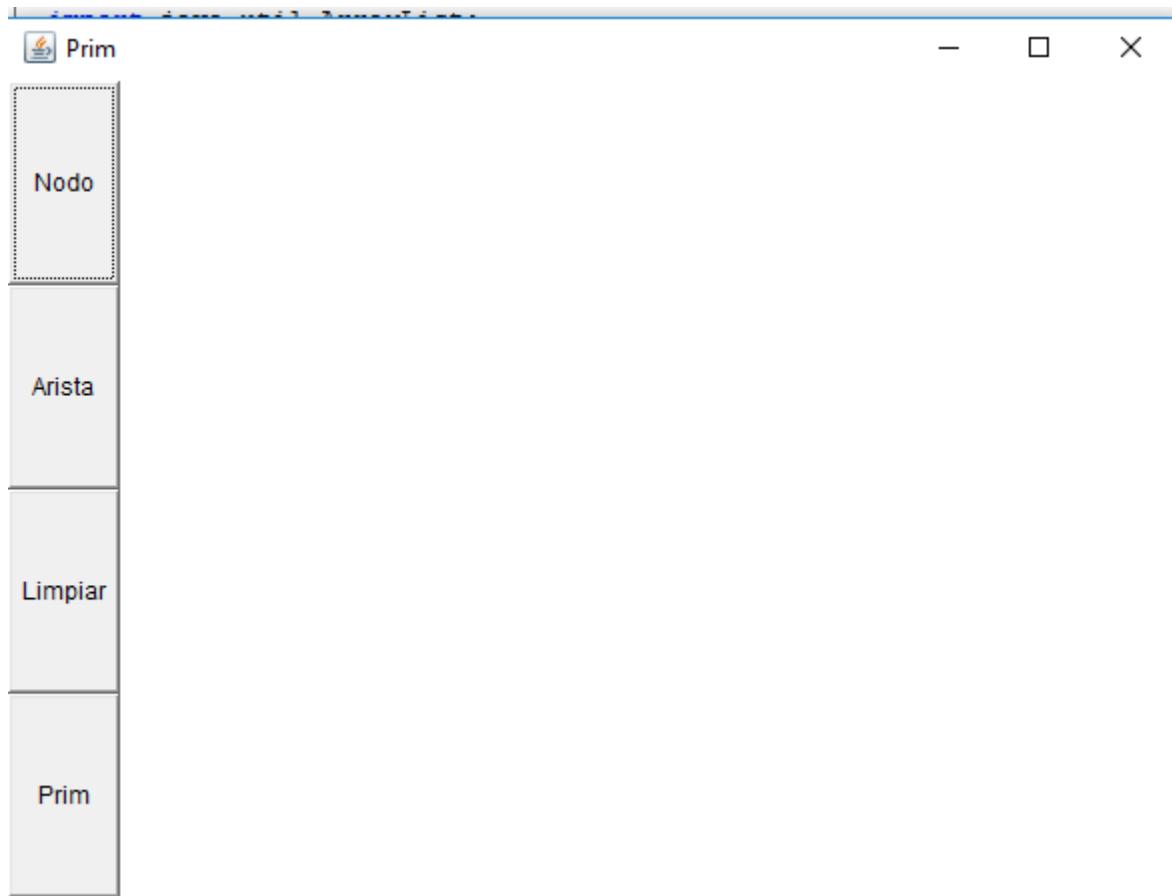
Quedando:

The screenshot shows a Java IDE interface. On the left is the project tree under the 'Prim' project. It contains a 'Source Packages' folder with a 'Prim' package containing five files: Arista.java, Grafo.java, Nodo.java, Prim.java, and Programa.java. Below this are 'Test Packages', 'Libraries', and 'Test Libraries'. At the bottom left is a 'Navigator' window titled 'grafo - Navigator' showing 'Members' with no content. The main right pane is a code editor with tabs for 'Source', 'History', and other tools. The 'Source' tab is active, displaying the following Java code:

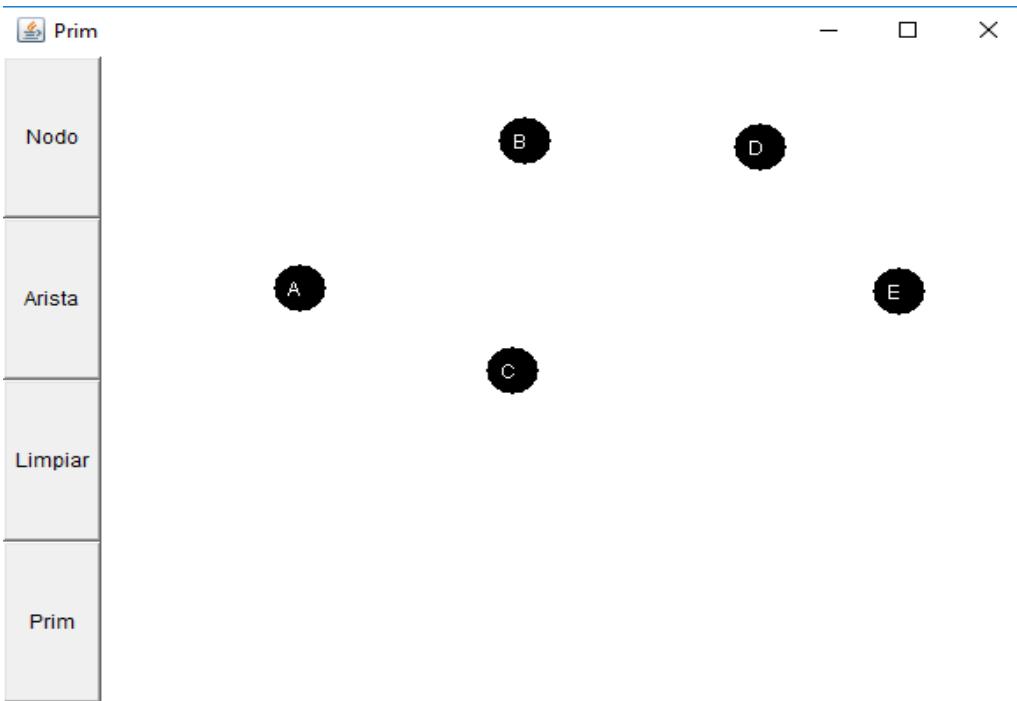
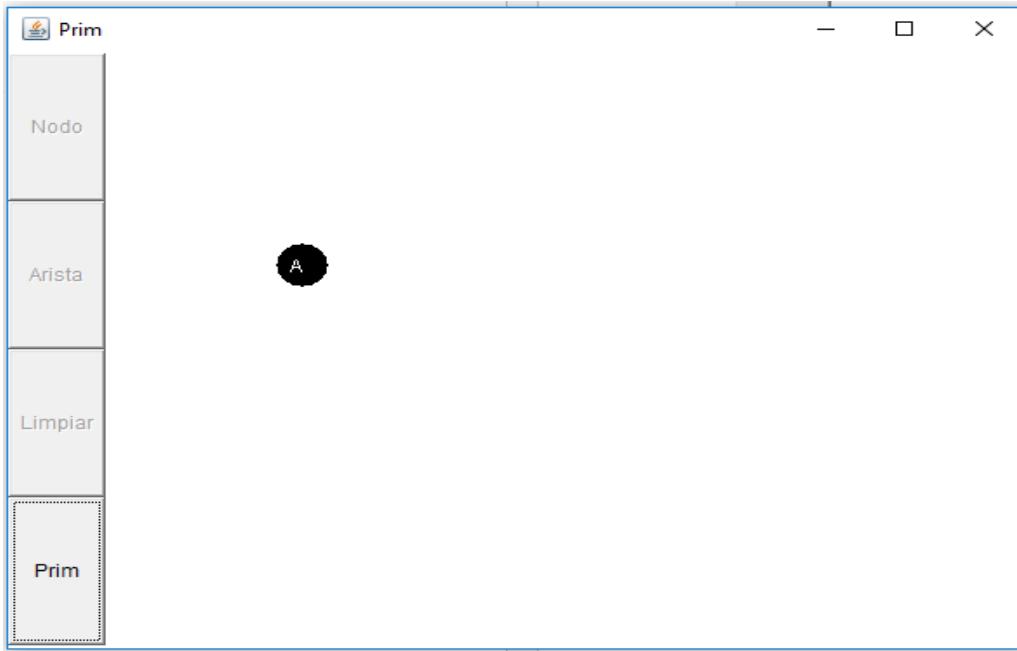
```
1 package prim;
2 import java.awt.*;
3 import java.awt.event.*;
4 import java.awt.Dialog;
5 import java.util.ArrayList;
6 public class Programa extends Frame implements ActionListener{
7     private Button nuevo;
8     private Button arista;
9     private Button limpia;
10    private Button prim;
11    private Grafo grafo;
12    private Dialog aristan;
13    private List nodos1;
14    private List nodos2;
15    private TextField peso;
16    private Button aristac;
17    private Dialog mensaje;
18    private Label etiqueta;
19
20    public Programa(){
21        Panel panel=new Panel(new GridLayout(4,0));
22        grafo = new Grafo(this);
```

Ejecución del Programa





Se adicionan los Nodos pulsando...



Después las aristas...

Agregar arista

Desde nodo:

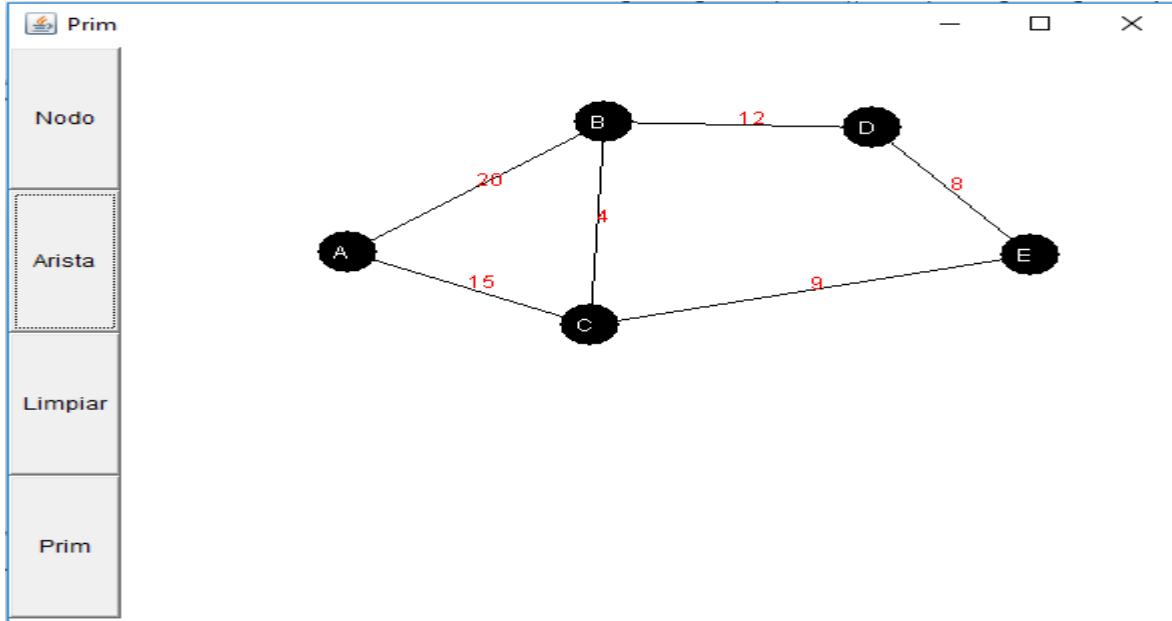
A
B
C
D

Hasta nodo:

A
B
C
D

Peso:

Agregar



Pulsar el Botón Prim

Prim

