



## Título de la actividad: recorridos en un grafo: componentes conexas.

Profesor Responsable: Sergio Alonso | Actividad II | Dificultad: media | Inicio : semana del 16 de abril de 2018 y corrección la siguiente

### Objetivo

El objetivo de esta actividad es un primer uso de la clase GRAFO que se implementó en la primera actividad, dotada de lo esencial para poder codificar los grafos y trabajar con ellos bajo distintos algoritmos. Ampliaremos el menú en sus opciones para grafos no dirigidos, permitiendo que se identifiquen las componentes conexas.

### Temporalización

Esta actividad se divide en dos prácticas. En la primera hora de tutorización, se introducirán los recorridos en grafos como herramienta básica de algoritmia en estas estructuras, y se implementará el recorrido en profundidad. Este método será la base para el procedimiento que identifica las componentes conexas de un grafo no dirigido, opción que se añadirá a la “calculadora” de grafos.

En la segunda y última fase, se presentará por parte del estudiante el ejecutable, que será corregido.

### Implementación

Un recorrido sobre un grafo es un método que permite visitar sus nodos de una forma sistemática y ordenada. Se pretende que, dada cierta filosofía de orden o prioridad que sea de interés, se visiten todos los nodos de manera eficiente: todos visitados y sin reiteración.

La analogía más cercana a la filosofía de un recorrido es la búsqueda en un laberinto: debemos asegurarnos de no dejar zona por estudiar, pero sin reiterar zonas que ciclen nuestra búsqueda. Por ello, precisamente, es útil llevar una memoria de los sitios visitados y de aquellos que dejamos pendientes por investigar. Siguiendo con la analogía, se trata de que, llegando a un punto (nodo) desde el que se abren a varios pasillos, (aristas o arcos), elijamos cualquiera de los pasillos para continuar el examen, pero apuntemos que debemos volver al mismo punto para optar por los pasillos pendientes. Esto lo haremos con dos sencillas herramientas:

- Un atributo para cada nodo que indique si ha sido o no visitado.
- Una lista de nodos pendientes por visitar.

Un esquema general de recorrido desde el nodo  $i$  sería el siguiente:

```

{inicialización}
Para todo nodo v, visitado[v] = falso;
{preparamos el inicio del recorrido desde el nodo i}
Visitado[i] = verdadero; ToDo = {i};
{bucle principal}
Mientras ToDo no vacío hacer
    Sea k en ToDo
    ToDo = ToDo - {k}
    Para todo adyacente j de k hacer
        Si visitado[j] = falso entonces
            Visitado[j]=verdadero;
            ToDo = ToDo + {j}

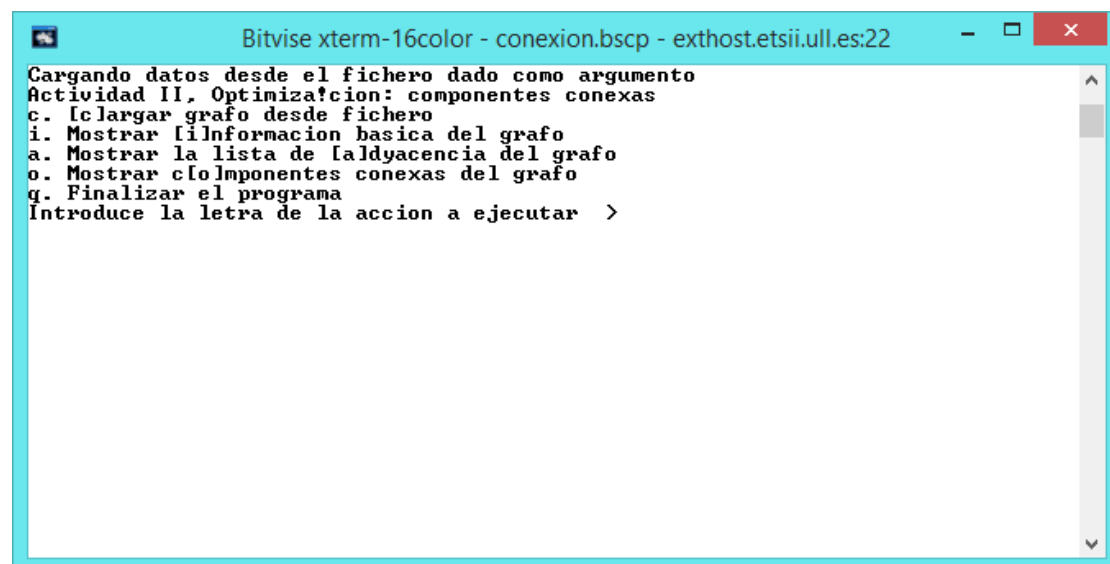
```

Durante la primera hora de tutorización veremos qué ocurre cuando la lista de nodos ToDo se gestiona como una pila o cuando lo hace como una cola, dando lugar a los dos típicos recorridos sobre un grafo, el recorrido en profundidad y el recorrido en amplitud. Asimismo, se aplicará la esencia de un recorrido a la identificación de las componentes conexas de un grafo no dirigido, pudiendo detectarse por tanto si el grafo es conexo. Para este caso, implementaremos un recorrido en profundidad, *deep first search*, **dfs**, en su expresión más sencilla, la recursiva.

### Ficheros de la actividad

Al igual que en la actividad anterior, trabajaremos con los ficheros **grafo.h** en el que se definen las estructuras, constantes, y se carguen el resto de ficheros de cabecera, así como con **grafo.cpp**, que las desarrolla.

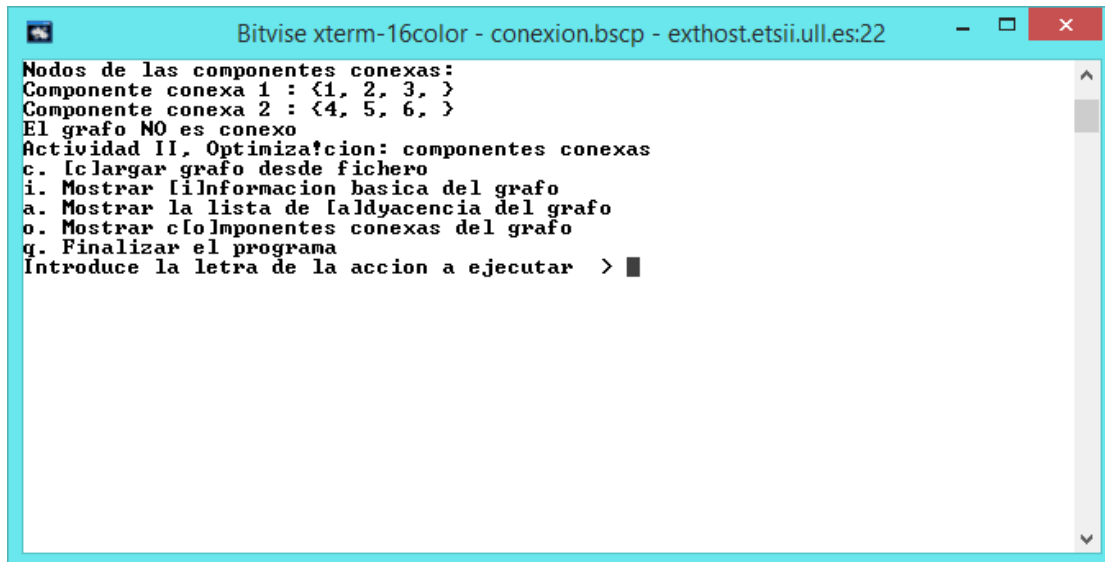
Además, la actividad deberá incluir un fichero con el programa principal `main`, que será un simple gestor tipo menú, al igual que la práctica anterior. y que estará en un fichero denominado **pg2.cpp**. Sólo se incluirá una opción en el menú de los grafos no dirigidos, que quedará así:



```

Bitwise xterm-16color - conexion.bscp - exthost.etsii.ull.es:22
Cargando datos desde el fichero dado como argumento
Actividad II. Optimización: componentes conexas
c. Cargar grafo desde fichero
i. Mostrar información básica del grafo
a. Mostrar la lista de adyacencia del grafo
o. Mostrar componentes conexas del grafo
q. Finalizar el programa
Introduce la letra de la acción a ejecutar >

```



Bitvise xterm-16color - conexion.bsccp - exthost.etsii.ull.es:22

```
Nodos de las componentes conexas:  
Componente conexas 1 : {1, 2, 3, }  
Componente conexas 2 : {4, 5, 6, }  
El grafo NO es conexo  
Actividad II, Optimizaci3n: componentes conexas  
c. Cargar grafo desde fichero  
i. Mostrar informaci3n b3sica del grafo  
a. Mostrar la lista de adyacencia del grafo  
o. Mostrar componentes conexas del grafo  
q. Finalizar el programa  
Introduce la letra de la acci3n a ejecutar > █
```

## Métodos de la clase GRAFO

Se añadirán, en esta práctica, los métodos necesarios para esta actividad son:

```
void dfs(unsigned i, vector<bool> &visitado);
```

El procedimiento dfs permite visitar todos los nodos de un grafo no dirigido, viajando de uno a otro por la adyacencia. Su expresi3n más sencilla es recursiva, y por ello, será objeto de estudio en la segunda hora de tutorizaci3n de esta actividad. Aún así, el método es:

```
void GRAFO::dfs(unsigned i, vector<bool> &visitado)  
{  
    visitado[i] = true;  
    cout << i+1 << ", ";  
    for (unsigned j=0; j<LS[i].size(); j++)  
        if (visitado[LS[i][j].j] == false)  
            dfs(LS[i][j].j, visitado);  
}
```

```
void ComponentesConexas();
```

Finalmente, usaremos el método dfs para identificar las componentes conexas de un grafo no dirigido, como una aplicaci3n básica de la clase GRAFO en esta actividad. También esta parte será objeto de estudio en la segunda hora de tutorizaci3n de esta actividad.

## Evaluaci3n

Para superar esta actividad, los procedimientos de carga de las listas de sucesores, predecesores y adyacencia, para los dos tipos de grafos, deben estar correctas. Asimismo, debe funcionar correctamente la identificaci3n de las componentes conexas, y ser adecuado el uso del procedimiento de recorrido. El profesor podr3 valorar la limpieza, documentaci3n del c3digo y la optimizaci3n de los recursos para puntuar por encima del apto.