

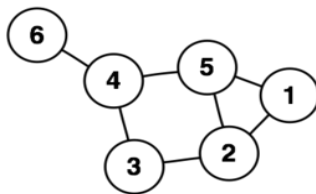
Fuerza Bruta, Búsqueda Exhaustiva y Recursividad

Fuerza Bruta

Muchos problemas en ciencias de la computación pueden ser modelados empleando el TDA grafos. Un grafo G se puede definir como un par (V, A) . Donde V es el conjunto de los vértices y A es el conjunto de las aristas. Una arista es el arco que une dos vértices. Para nombrar un vértice emplearemos letras minúsculas, por ejemplo los vértices u, v . Mientras que el par (u, v) representa la arista que une estos dos vértices.

Los grafos pueden ser dirigidos y no dirigidos. En un grafo no dirigido significa que da lo mismo (u, v) que (v, u) . Mientras que en el caso contrario la diferencia es significativa. Nosotros en este material trabajaremos con grafos no dirigidos.

Sea el grafo descrito por la figura siguiente:

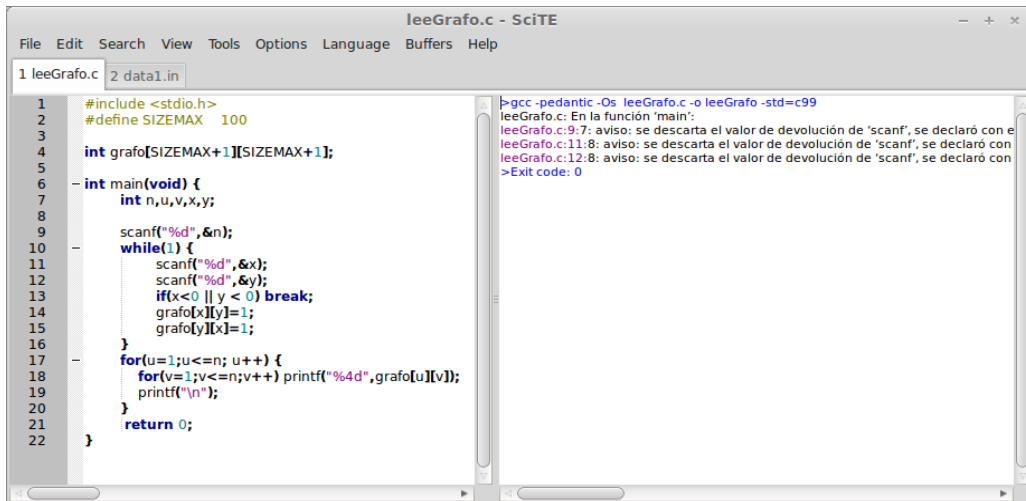


Los vértices son $V = \{1, 2, 3, 4, 5, 6\}$
y las aristas $A = \{(1, 5), (5, 1), (1, 2), (2, 1), (5, 2), (2, 5), (4, 5), (5, 4), (2, 3), (3, 2), (3, 4), (4, 3), (6, 4), (4, 6)\}$

Esta forma de representar el grafo sigue la definición, sin embargo para que un programa pueda aceptarlo en un ingreso de datos debe tener otra forma. Existe dos formas comúnmente empleadas para representar los grafos: la *matriz de adyacencia* y las *listas de adyacencias*. Nosotros emplearemos la primera en este material.

Sea un grafo no dirigido $G=(V, A)$ donde $V=\{1, 2, \dots, n\}$ la matriz de adyacencia para G es una matriz M de dimensión $n \times n$, de elementos booleanos, donde $A[i, j]$ es verdadero si, solo si, existe un arco que vaya del vértice i al j . Con frecuencia se muestran matrices de adyacencia con 1 para verdadero y 0 para falso. Por último remarcaremos el hecho de que la matriz de adyacencia de un grafo no dirigido siempre es simétrica.

A continuación se muestra un programa que lee del teclado el número de vértices que tiene un grafo, luego lee los arcos y a continuación muestra la matriz de adyacencia. Como la matriz es simétrica sólo se leen los arcos en un sólo sentido, cuidando de realizar la asignación en ambos sentidos. Cuando se ingresa algún valor negativo como arco, es un indicativo de que el ingreso de datos ha terminado. A continuación se muestra el programa y su compilación respectiva:



```
#include <stdio.h>
#define SIZEMAX 100

int grafo[SIZEMAX+1][SIZEMAX+1];

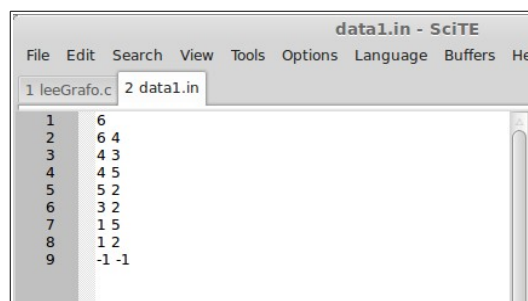
int main(void) {
    int n,u,v,x,y;

    scanf("%d",&n);
    while(1) {
        scanf("%d",&x);
        scanf("%d",&y);
        if(x<0 || y<0) break;
        grafo[x][y]=1;
        grafo[y][x]=1;
    }
    for(u=1;u<=n; u++) {
        for(v=1;v<=n;v++) printf("%4d",grafo[u][v]);
        printf("\n");
    }
    return 0;
}
```

```
>gcc -pedantic -Os leeGrafo.c -o leeGrafo -std=c99
leeGrafo.c: En la función 'main':
leeGrafo.c:9:7: aviso: se descarta el valor de devolución de 'scanf', se declaró con e
leeGrafo.c:11:8: aviso: se descarta el valor de devolución de 'scanf', se declaró con
leeGrafo.c:12:8: aviso: se descarta el valor de devolución de 'scanf', se declaró con
>Exit code: 0
```

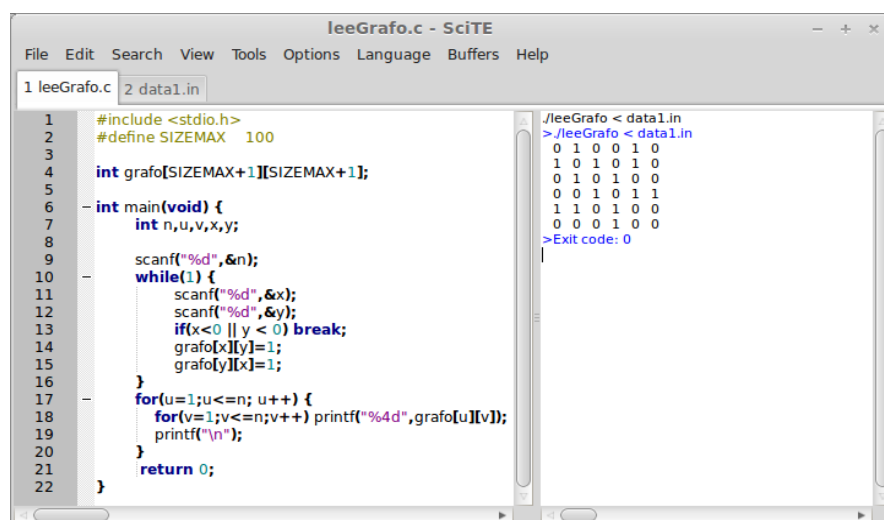
Los mensajes mostrados no son errores son avisos (*warnings*) o advertencias, acerca de algunas acciones que podrían causar problemas. En este caso se nos advierte que el valor devuelto por la función *scanf* no lo estamos empleando.

Antes de ejecutar el programa crearemos un archivo de texto con nombre *data1.in* y en ella colocaremos los datos que deseamos ingresar. Luego redireccionamos la entrada estándar para que en lugar de leer los datos del teclado los lea del archivo. A continuación se muestra el archivo *data1.in* que se debe encontrar en el mismo directorio donde se encuentra el ejecutable.



```
1 6
2 6 4
3 4 3
4 4 5
5 5 2
6 3 2
7 1 5
8 1 2
9 -1 -1
```

Luego en el lado derecho de la IDE de SciTE ejecutamos el programa de la siguiente forma:



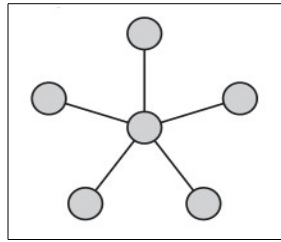
```
1 leeGrafo.c 2 data1.in
1 #include <stdio.h>
2 #define SIZEMAX 100
3
4 int grafo[SIZEMAX+1][SIZEMAX+1];
5
6
7 int main(void) {
8     int n,u,v,x,y;
9
10    scanf("%d",&n);
11    while(1) {
12        scanf("%d",&x);
13        scanf("%d",&y);
14        if(x<0 || y<0) break;
15        grafo[x][y]=1;
16        grafo[y][x]=1;
17    }
18    for(u=1;u<=n; u++) {
19        for(v=1;v<=n;v++) printf("%4d",grafo[u][v]);
20        printf("\n");
21    }
22    return 0;
23 }
```

```
./leeGrafo < data1.in
>./leeGrafo < data1.in
0 1 0 0 1 0
1 0 1 0 1 0
0 1 0 1 0 0
0 0 1 0 1 1
1 1 0 1 0 0
0 0 0 1 0 0
>Exit code: 0
```

Observe que la orden para ejecutar ha sido: *./leeGrafo < data1.in*

Ejercicios

1.- Una topología de red especifica cómo las computadoras, impresoras, y otros dispositivos están conectados sobre una red. La figura de abajo muestra la topología de red conocida como estrella.



Elabore un programa a fuerza bruta (en C) que reciba un arreglo $M[0..n-1,0..n-1]$ de booleanos donde $n > 3$, el mismo que representa la matriz de adyacencia de un grafo. El programa deberá determinar si el grafo representa o no una topología estrella.

2.- Modifique el programa anterior para que imprima el número de vértices que tiene el grafo y el número de iteraciones que realizó el programa hasta encontrar la respuesta. Además imprima el tiempo transcurrido en milisegundos (vea laboratorio preliminar)

3.- A la luz de los datos obtenidos en la pregunta anterior, calcule cuál debe ser el tamaño ser el tamaño de la matriz para que el programa demore un día (24 horas).

4.- Se tiene una secuencia aleatoria de números enteros (positivos y negativos), se asegura además que no hay ceros, elabore un programa a fuerza bruta que coloque los negativos a la izquierda y los positivos a la derecha. Sólo está permitido una operación: el intercambio entre dos números consecutivos. El programa deberá imprimir el número de iteraciones que realizó.

Búsqueda exhaustiva

Muchos problemas importantes requieren encontrar un elemento con una propiedad especial en un dominio que crece exponencialmente (o muy rápido) con un tamaño específico. Típicamente, tales problemas surgen en situaciones que involucran -explícita o implícitamente- combinación de objetos tales como permutaciones, combinaciones, y subconjuntos de un conjunto dado. Muchos de dichos problemas, son problemas de optimización: ellos solicitan encontrar un elemento que maximice o minimice alguna característica deseada tal como la longitud de una ruta o una asignación de costo.

La búsqueda exhaustiva es simplemente un enfoque de fuerza bruta a un problema combinatorio. Este sugiere generar todos y cada uno de los elementos del dominio del problema, seleccionando de estos aquellos que satisfacen todas las restricciones, y luego encontrar el elemento deseado (por ejemplo el elemento que optimiza alguna función objetivo). Note que aunque la idea de una búsqueda exhaustiva es bastante sencilla, su implementación típicamente requiere un algoritmo para generar cierta combinación de objetos.

Cuadrados mágicos

Un cuadrado mágico de orden n es una disposición de enteros de 1 hasta n^2 en una matriz de orden $n \times n$, en la que cada número aparece una sola vez, de modo que cada fila, columna y diagonal principal tengan la misma suma.

Por ejemplo el cuadrado mostrado abajo es un cuadrado mágico de orden 4.

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

Ejercicios

- 1.- Elabore un programa en C, de búsqueda exhaustiva para generar todos los cuadrados de orden n .
- 2.- Modifique el programa anterior para que imprima el número n y el número de iteraciones que realice el programa hasta encontrar la respuesta. Además imprima el tiempo transcurrido en milisegundos (vea laboratorio preliminar)
- 3.- A la luz de los datos obtenidos en la pregunta anterior, calcule cuál debe ser el tamaño del cuadrado mágico para que el programa demore un día (24 horas).

Recursividad

- 1.- Elaborar en C, una función recursiva que sume los números desde 1 hasta n . Donde n es ingresado por teclado. Escriba una función *main* apropiada para probar su función recursiva.
- 2.- Elabore en C, una función recursiva que reciba una cadena y devuelve otra conteniendo la cadena invertida. Escriba una función *main* apropiada para probar su función recursiva.
- 3.- La forma de calcular las diferentes maneras de elegir r cosas distintas de un conjunto de m cosas es: $C(m, r) = m! / (r! * (m - r)!)$ Donde ! Representa la función factorial. Elabore en C una función recursiva de la fórmula anterior que calcule dicho valor sin calcular el factorial. Escriba una función *main* apropiada para probar su función recursiva.
- 4.- Escriba una función recursiva que ordene de menor a mayor un arreglo de enteros basándose en la siguiente idea: coloque el elemento más pequeño en la primera ubicación, y luego ordene el resto del arreglo con una llamada recursiva. Escriba una función *main* apropiada para probar su función recursiva.

El laboratorio estará basado en un problema análogo a los propuestos o con una variante modificada. No podrá llevar ningún dispositivo de almacenamiento (CD,DVD, usb). Sólo puede llevar material impreso.