HPPS
Informe 3 - Ejercicios de grafos

Juan Braun

19 de mayo de 2013

Introducción

El objetivo de este informe es comentar los resultados obtenidos para los ejercicios propuestos.

1. Ejercicio 1

El objetivo de este ejercicio fue hacer un programa para calcular el histograma de grados de todos los vértices para un grafo cualquiera de orden *N*.

La entrada al programa es un archivo con la matriz de adyacencia. Para generar este archivo se creo un ejecutable que genera matrices de adyacencia de forma aleatoria. El código fuente para este ejecutable esta en el archivo `adjMat.c`.

Los argumentos a ingresar son el orden del grafo, una bandera para hacer que la matriz de adyacencia sea simétrica y el nombre del archivo en el que se va a guardar la salida. A continuación se muestra una llamada al ejecutable.

```
./adjMat 4 1 mat
```

A continuación se muestra la salida de la función que calcula los grados y el histograma.

```
size= 4
0 1 1 1
1 0 0 0
1 0 0 0
1 0 0 0
GRADO
3 1 1 1
HISTOGRAMA
0 3 0 1
```

2. Ejercicio 2

El objetivo de este ejercicio fue implementar el algoritmo de búsqueda en profundidad (*DFS*), en sus dos variantes, la version recursiva y la version iterativa.

Se utilizaron diferentes arboles para probar ambas versiones.

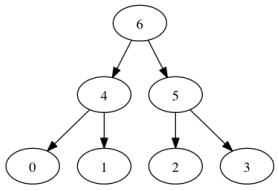

Figura 1: Árbol de prueba 1(A1)

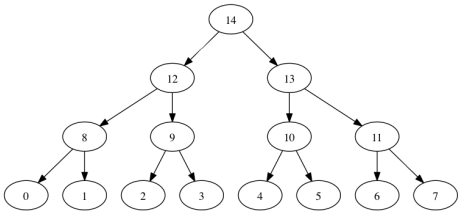

Figura 2: Árbol de prueba 2(A2)

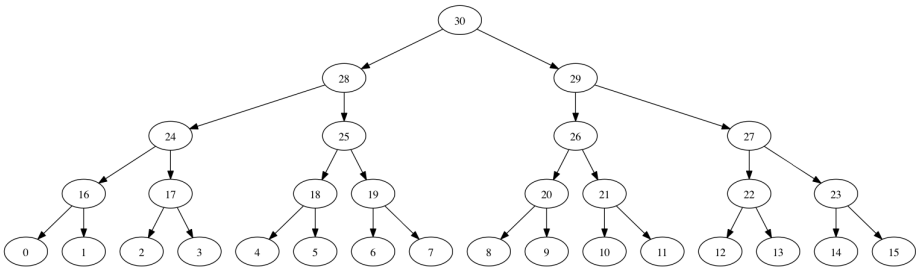

Figura 3: Árbol de prueba 3(A3)

	`dfs_rec`	`dfs_it`
A1	168	437
A2	376	893
A3	792	1805

Cuadro 1: Ciclos para versión recursiva e iterativa

Como se puede ver en el Cuadro 1 la versión recursiva del algoritmo necesita menos ciclos para realizar la búsqueda. Ademas se verifica que el algoritmo es de orden *O(|E|)* ya que para los diferentes arboles los ciclos crecen linealmente.

Para entender mejor porque una versión requiere menos ciclos que la otra se muestra es pseudo-código de ambas.

```
dfs_rec(A,n)

if (n no tiene hijos)
  estoy en el fondo->proceso n
else
  n = primer hijo de n
    dfs_rec(A,n)
    while (n tiene hermanos)
      n = hermano de n
      dfs_rec(A,n)
    end while
    proceso n
end if

end
```

```
dfs_it(A,n)

creo stack Abiertos
creo stack Cerrados
pongo nodo n en stack Abiertos
while(Abiertos no este vacio)
  saco nodo n de Abiertos
  pongo nodo n en Cerrados
  if (n tiene hijos)
    n =primer hijo de n
    pongo a n en Abiertos
    while(n tiene hermanos)
      n = hermano de n
      pongo a n en Abiertos
    end while
  end if
end while

proceso los nodos en el orden que aparecen en Cerrados

end
```

Lo primero que se nota a simple vista es el tamaño de ambos pseudo-códigos. Si

bien la versión iterativa es la mas intuitiva, la implementación de esta es mas compleja. Ademas hay que crear stacks para almacenar los nodos que se vistan, por lo que se requiere reservar memoria. Finalmente se procesan los nodos una vez terminada la búsqueda, ya que el stack que se utiliza para guardar los nodos es un stack LIFO.

En la versión recursiva se procesan los nodos a medida que se va recorriendo el arbol. Ademas no se necesita crear ningún stack para almacenar nodos visitados.

La diferencia entre la cantidad de ciclos de reloj se debe a la cantidad de movimientos de memoria que se realizan en la versión iterativa.

Cuando se realiza procesamiento sobre los nodos, el desempeño de ambas versiones deja de ser tan diferente. En el caso de las pruebas realizadas, se tomo como procesamiento del nodo la acción de imprimir el nombre del nodo en pantalla. Para este caso se puede ver en el Cuadro 2 que los ciclos son similares para ambas versiones.

	`dfs_rec`	`dfs_it`
A1	5767	6108
A2	12601	13266
A3	26545	27858

Cuadro 2: Ciclos para versión recursiva e iterativa con procesamiento

De todas formas se concluye que la versión recursiva del algoritmo de búsqueda en profundidad es la más eficiente y la más elegante.