Semana 7

Pedro O. Pérez M., PhD.

Análisis y diseño de algoritmos avanzados
Tecnológico de Monterrey

pperezm@tec.mx

09-2021

Usando DFS
- Subgrafos máximos
- Topological Sort

Búsqueda en profundidad - Depth First Search

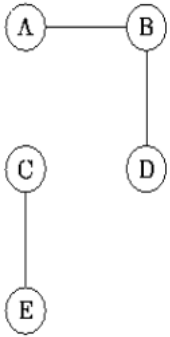
Procedure 1 DFS

Input: u : *Vertex*, G : *Graph*, *Reached* : *Set*

```
Mark u as Explored and add to Reached
for each (u, v) in G incident to u do
    if v is not marked Explored then
        DFS(v, G, Reached)
    end if
end for
```

Subgrafos máximos

Considere un grafo G formado a partir de un gran número de vértices conectados por arcos. G se dice que está conectado si existe un camino entre cualquier par de vértices en G. Por ejemplo, el siguiente grafo no está conectado, porque no hay trayectoria de A a C.



Este grafo contiene, sin embargo, un número de subgrafos que están conectados, uno para cada uno de los siguientes conjuntos de vértices: (A), (B), (C), (D), (E), (A, B), (B,D), (C, E), (A, B, D). Un subgrafo conectado es máximo si no hay vértices y arcos en el grafo original que podrían añadirse al subgrafo y todavía dejarlo conectado. En la imagen anterior, hay dos subgrafos máximos, uno asociada con los vértices (A, B, D) y el otro con los vértices (C, E). Desarrollar un algoritmo para determinar el número de subgrafos máximos conectados de un gráfico dado.

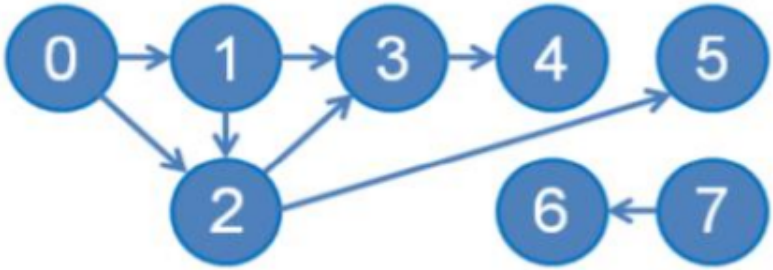
`http://bit.do/eNTvC`

Procedure 2 COUNTING_GRAPHS

Input: G : *Graph*
  Reached : *Set*
  $acum \leftarrow 0$
  Mark all the vertexes in G as *No_Explored*
  for *vertex* in G **do**
    if *vertex* is not marked *Explored* **then**
      $DFS(vertex, G, Reached)$
      $acum \leftarrow acum + 1$
    end if
  end for
  return *acum*

Topological Sort

Un "Topological Sort"de un Grafo Direccionado Acíclico (Directed Acyclic Graph, DAG) es un ordenamiento lineal de los vértices que aparecen en un DAG tal que si el vértice u aparece antes de v es porque existe un arco $(u \rightarrow v)$ en el DAG. Cada DAG tiene al menos, y posiblemente más, "topological sort".


0
1
3
4
5
2
6
7

Procedure 3 DFS2

Input: $u : Vertex, G : Graph, Reached : Set, TS : Stack$

```
Mark u as Explored and add to Reached
for each (u, v) incident to u do
    if v is not marked Explored then
        DFS2(v, G, Reached, TS)
    end if
end for
TS.push(u)
```

Procedure 4 TOPOLOGIGAL_SORT

Input: G : *Graph*
  Reached : *Set*
  TS : *Stack*
  for each *vertex* in G **do**
    if *vertex* is not marked *Explored* **then**
      $DFS2(G, v, Reached, TS)$
    end if
  end for
  while *TS* is not empty **do**
    print $TS.top()$
    $TS.pop()$
  end while

Procedure 5 TOPOLOGIGAL_SORT2

Input: G : *Graph*

Let d be an array of the same length as V; this will hold the shortest-path distances from s. Set $d[s] = 0$, all other d[u] = ínf.
Let p be an array of the same length as V, with all elements initialized to nil. Each $p[u]$ will hold the predecessor of u in the shortest path from s to u.
for each vertex u as ordered in V, starting from s **do**
  for each vertex v into u (i.e., there exists an edge from v to u) **do**
    Let w be the weight of the edge from v to u
    if d[u] > d[v] + w **then**
      d[u] $\leftarrow$ d[v] + w
      p[u] $\leftarrow$ v
    end if
  end for
end for