Técnicas para Explorar Gráficas

Emmanuel Peto Gutiérrez

María De Luz Gasca Soto

Mayo, 2020

Departamento de Matemáticas,
Facultad de Ciencias, UNAM.

Índice general

Luz Gasca y Emmanuel Peto. F.C, UNAM

Técnicas Básicas: BFS, DFS, TS

El primer problema que se encuentra al intentar diseñar un algoritmo que involucre gráficas es cómo revisar la entrada. Una estructura de una dimensión (como un arreglo o una lista) se puede revisar fácilmente de forma lineal. Explorar una gráfica no es tan sencillo. Presentamos dos algoritmos para explorar una gráfica: **búsqueda en amplitud**, *Breadth-first search*, BFS y **búsqueda en profundidad**, *Depht-first search*, DFS.

De ambos algoritmos se puede obtener información de la gráfica como: encontrar los vértices alcanzables desde un vértice fuente s, y obtener las componentes conexas. Del algoritmo BFS se puede obtener la distancia de un vértice fuente s a todos los vértices alcanzables desde s. Con el algoritmo DFS se pueden encontrar los vértices de corte y los bloques de la gráfica.

Otro algoritmo importante, que revisaremos, es *Topological Sorting* también llamado **ordenamiento topológico**. Aquí, cada tarea se representa con un vértice y existe una arista dirigida (o arco) de la tarea x a la tarea y si la tarea y no puede empezar hasta que x termine; la gráfica debe ser acíclica. Con el algoritmo TS se pueden organizar tareas.

Luz Gasca y Emmanuel Peto. F.C, UNAM

Capítulo 3

Topological Sorting, TS

Supongamos que existe un conjunto de tareas que se necesitan ejecutar una a la vez. Algunas tareas dependen de otras y estas no se pueden empezar hasta que las otras tareas se completen. Todas las dependencias se conocen y se quiere organizar un horario para realizar las tareas que sea consistente con las dependencias; es decir, cada tarea está programada para ser realizada sólo después de que todas las tareas de las cuales esta depende han sido completadas. Necesitamos diseñar un algoritmo rápido que genere ese horario. Este problema es denominado **ordenamiento topológico**.

Podemos asociar una gráfica dirigida con las tareas y sus dependencias de la siguiente manera. Cada tarea se representa con un vértice y existe una arista dirigida (arco) de la tarea x a la tarea y si y no puede empezar hasta que x se termine. Es claro que, la gráfica debe ser acíclica; en otro caso, algunas tareas nunca pueden empezar, o bien, algunas tareas podrían repetirse infinitamente.

El problema modelado con gráficas lo podemos enunciar de la siguiente manera:

Problema TS: Dada una digráfica acíclica $G = (V, E)$ con n vértices, se requiere etiquetar los vértices desde 1 hasta n tal que, si el vértice v se etiqueta con k, entonces todos los vértices que pueden ser alcanzados desde v por una trayectoria dirigida se etiquetan con números mayores a k.

Podemos diseñar este algoritmo por inducción, sobre el número de vértices. La hipótesis de inducción es la siguiente:

Hipótesis de Inducción. Sabemos cómo etiquetar todas las gráficas acíclicas con menos de n vértices de acuerdo a las condiciones establecidas en el Problema TS.

El caso base para un vértice es trivial, ya que únicamente se tiene un vértice.
Pasemos al paso inductivo: Consideramos una gráfica de n vértices, quitamos un vértice. Ahora, aplicamos la hipótesis de inducción para intentar extender la etiquetación. Al inicio, tenemos la libertad elegir cualquier vértice como el n-ésimo, así que nos conviene seleccionar uno que nos simplifique el trabajo. Ya que necesitamos etiquetar vértices, nos

preguntamos ¿Cuál es el más fácil de etiquetar? Claramente, un vértice (tarea) que no tenga dependencias, nos facilitará el trabajo; es decir, un vértice cuyo in-grado[1] sea 0. A este vértice lo podemos etiquetar con 1 sin problemas. Pero... ¿Se puede encontrar siempre un vértice de in-grado 0 en la digráfica dada? La respuesta intuitiva es sí, ya que debemos poder empezar en algún vértice. El siguiente resultado establece este hecho.

Teorema 3.1 Toda digráfica acíclica posee, al menos, un vértice con indegree 0, [1].

Antes de describir *Topological Sorting* necesitamos una forma de calcular el in-grado de cada vértice en la digráfica dada. Se puede tener un atributo `indegree` en cada vértice v, inicialmente su valor será 0, y lo iremos incrementando cada vez que se encuentre un arco (u, v). Para cada vértice u, se debe incrementar en 1 el in-grado de cada vértice $v \in u.ady$. El Cuadro 3.1 muestra el procedimiento para calcular el in-grado de cada vértice.

```
CalcIndegree(G)
1 for each u ∈ G.V
2     u.indegree = 0
3 for each u ∈ G.V
4     for each v ∈ u.ady
5         v.indegree++
```

Cuadro 3.1: Cálculo del in-grado para cada vértice.

Dar valor inicial a todos los in-grados en 0 toma tiempo $O(|V|)$ e incrementar el `indegree` de todos los vértices tiene tiempo $O(|E|)$, ya que se se están explorando todas las aristas (u, v). Así, la complejidad total del proceso CalcIndegree es $O(|V| + |E|)$.

Para *Topological Sorting*, el primer paso consiste en encontrar un vértice con `indegree` 0. Una vez que se encuentra, se etiqueta con 1, se elimina de la digráfica, junto con sus arcos de salida, y se etiqueta el resto de la gráfica (la cual sigue siendo acíclica) con números de 2 a n.

Los únicos problemas de implementación son:

(1) cómo encontrar un vértice con in-grado 0; y

(2) cómo ajustar los in-grados cuando un vértice es eliminado.

Para resolver estos problemas, primero calculamos el in-grado de cada vértice. Los vértices con `indegree` 0 se colocan en una lista (cola o una pila). Ahora es fácil seleccionar al primer vértice v: simplemente se remueve de la lista. Después, para cada arco (v, w) que sale de v, decrementamos en 1 el atributo `indegree` de w. Cuando el contador se vuelve 0, colocamos al vértice en la lista. Después de eliminar a v, la digráfica sigue siendo acíclica. Por lo tanto, por el Teorema 3.1, debe haber al menos un vértice con in-grado 0 en la digráfica restante. El algoritmo termina cuando la cola es vacía, en cuyo caso todos los vértices han sido etiquetados.

[1] Indegree.

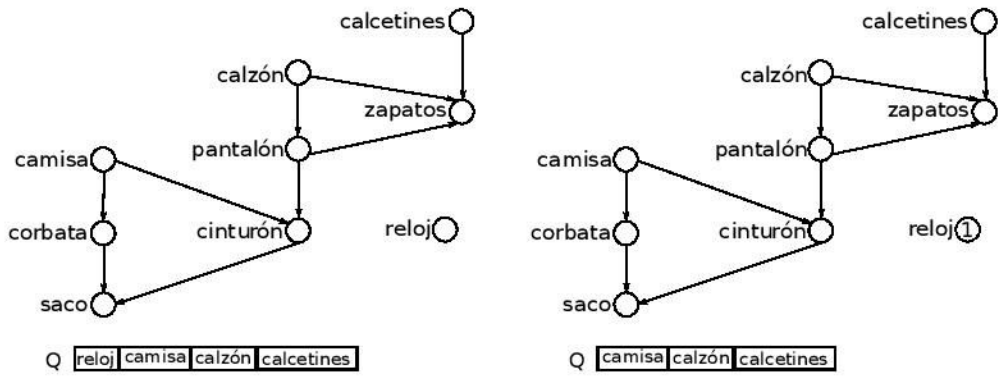
Supongamos que la etiqueta de *Topological Sorting*, para un vértice v, se almacena en un atributo llamado `tsi`, índice del TS, el algoritmo se da en el Cuadro 3.2.

```
TopologicalSorting(G)
 1 calcIndegree(G)
 2 Q = ∅
 3 label = 0
 4 for each v ∈ G.V
 5    if v.indegree == 0
 6       Q.enqueue(v)
 7 while Q ≠ ∅
 8    v = Q.dequeue()
 9    label++
10    v.tsi = label
11    for each w ∈ v.ady
12       w.indegree–
13       if w.indegree == 0
14          Q.enqueue(w)
```

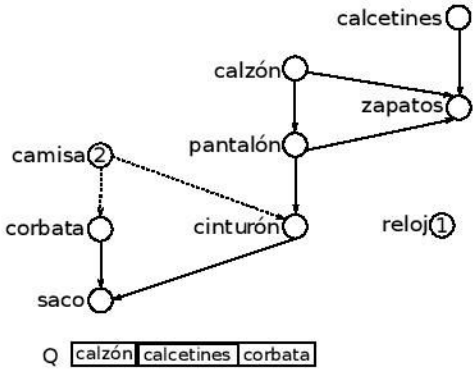
Cuadro 3.2: Proceso Topological Sorting.

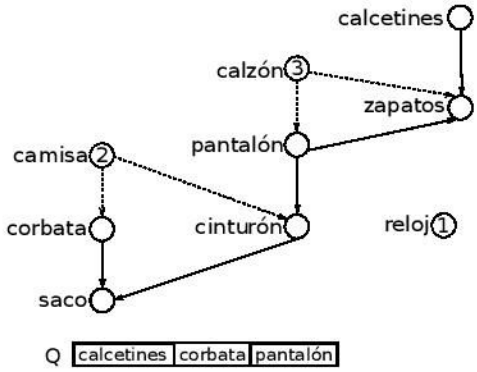
En la Figura 3.1 presentamos un ejemplo del *Topological Sorting*. En este caso es para decidir el orden en el que una persona se viste por la mañana antes de ir a la oficina. Existe una dependencia (es decir, un arco) de x a y si es necesario ponerse la prenda o accesorio x antes de y; por ejemplo, se debe poner los calcetines antes de los zapatos.

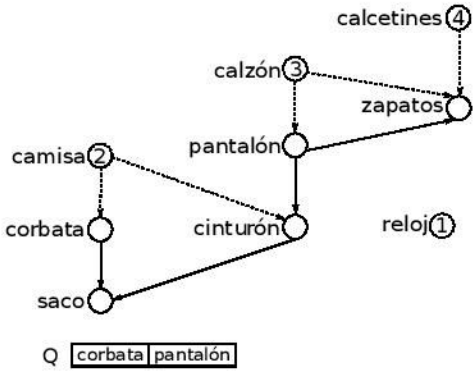
En vez de borrar un vértice, simplemente marcamos el contorno, del vérticie, con línas punteadas. Las aristas que se remueven también se representan con línas punteadas. No mostramos el atributo `indegree` pero se puede determinar por el número de flechas no punteadas incidentes en el vértice.

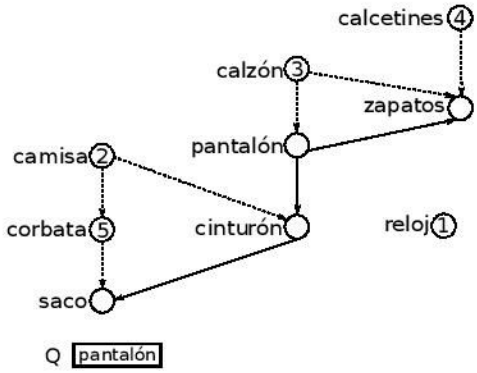


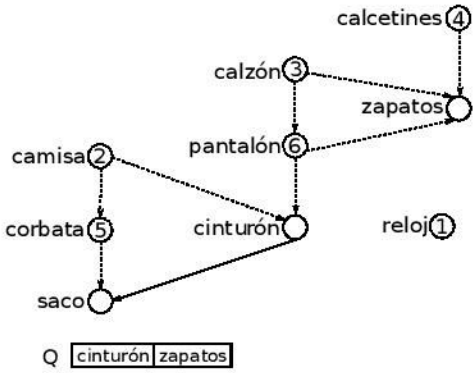
Así podemos determinar que la persona se vistió de la siguiente manera:

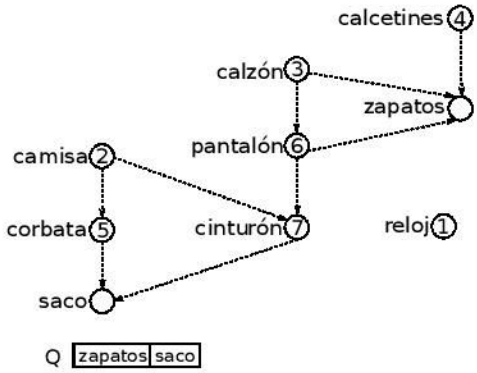

calcetines
calzón
zapatos
pantalón
camisa 2
corbata
cinturón
reloj 1
saco
Q
calzón
calcetines
corbata


calcetines
calzón 3
zapatos
pantalón
camisa 2
corbata
cinturón
reloj 1
saco
Q
calcetines
corbata
pantalón


calcetines 4
calzón 3
zapatos
pantalón
camisa 2
corbata
cinturón
reloj 1
saco
Q
corbata
pantalón


calcetines 4
calzón 3
zapatos
pantalón
camisa 2
corbata 5
cinturón
reloj 1
saco
Q
pantalón


calcetines 4
calzón 3
zapatos
pantalón 6
camisa 2
corbata 5
cinturón
reloj 1
saco
Q
cinturón
zapatos


calcetines 4
calzón 3
zapatos
pantalón 6
camisa 2
corbata 5
cinturón 7
reloj 1
saco
Q
zapatos
saco

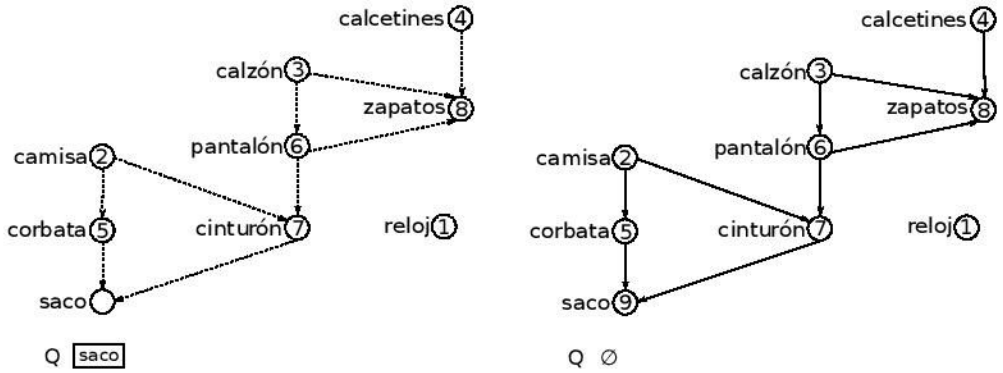


Figura 3.1: Aplicación de topological sort.

1. Reloj; 2. Calzón; 3. Camisa; 4.Calcetines; 5. corbata;
6. Pantalón; 7. Cinturón; 8. Zapatos; 9. Saco.

3.1. Análisis de complejidad

Dar valores iniciales al atributo `indegree` de cada vértice requiere tiempo $O(|V| + |E|)$. Encontrar un vértice con `indegree 0` es constante, pues solo se requiere sacar al elemento de la lista, en esta implementación de la cola (desencolar). Cada arista (v, w) es considerada solo una vez (cuando se desencola v). Así, el número de veces que la variable necesita ser actualizada es exactamente igual al número de aristas en la gráfica. Por lo tanto, el tiempo de ejecución del algoritmo es $O(|V| + |E|)$, en el peor de los casos.

Cuídate!

Quedate en Casa!

... es tu lugar seguro ;)

Luz Gasca

Bibliografía

[1] Udi Manber, *Introduction to Algorithms, A Creative Approach.* Addison-Wesley. 1989.

[2] Gary Chartran and O. Oellerman *Applied and Algorithmic Graph Theory.* McGraw-Hill, Inc. 1993.

[3] Thomas H. Cormen and C. E. Leiserson and R. L. Rivestn *Introduction to Algorithms.* The MIT Press, 2002.