Sort Topológico

Aplicación Práctica de este algoritmo :

Grafos de tipo PERT donde sea necesario identificar períodos de tiempo y cantidad máxima / mínima de recursos.

Introducción.

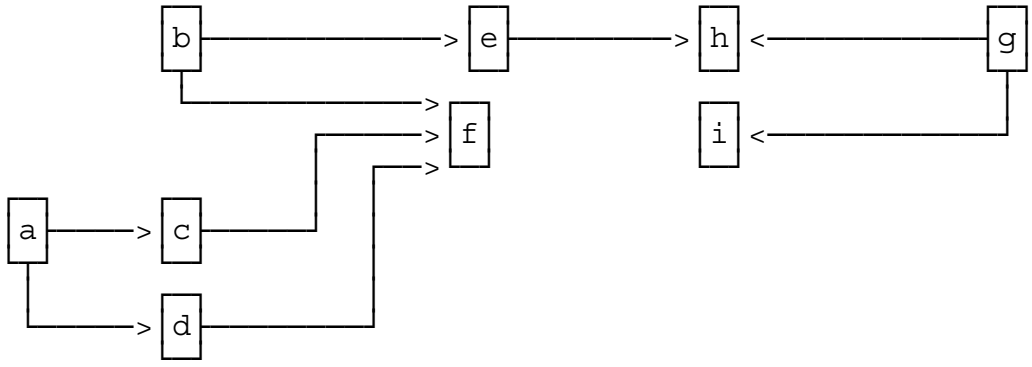
El sort topológico es un procedimiento basado en el concepto de paso. Este concepto es útil desde el punto de vista teórico, pues muchas de las definiciones algebraicas pueden ser definidas para la teoría de grafos, y desde el punto de vista práctico un paso puede verse como el proceso mediante el cuál se accede a un elemento de la estructura siguiendo en un cierto orden, las relaciones en esa estructura. De esta manera se puede determinar por ejemplo si existe una clave luego de haber recorrido una secuencia de nodos.

Definición.

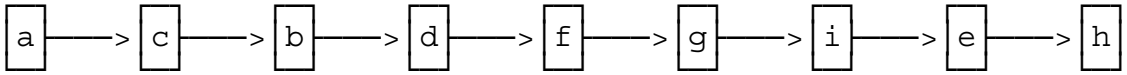
El sort topológico es el proceso de convertir un conjunto de precedencias en una lista lineal simple, en la cuál ningún elemento posterior precede a uno anterior.

Concepto del algoritmo.

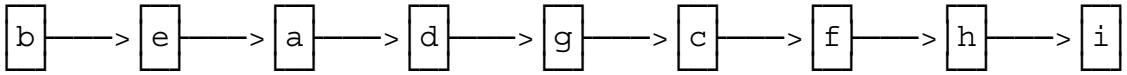
Sea **G (P, E)** un grafo. Una relación lineal S definida en P, se dice que es un sort topológico de P respecto de E si **_E(X,Z) => _S(X,Z)**. Sea el grafo **G (P, E)**



S_1 = (P, E_1)



S_2 = (P, E_2)



donde G es acíclico y S_1 y S_2 son sort topológico de G. La pregunta que cabe hacerse es si dado un grafo G , existirá necesariamente un sort topológico S, y de existir, si es único.

Dado un grafo finito G = (P, E), un sort topológico S de G existe <=> G es acíclico. Además S es único <=> E es una relación de orden total, en cuyo caso S es el grafo básico de G.

Procedimiento.

Llamaremos **TSORT** al algoritmo para el sort topológico. Puesto que G es acíclico, sabemos que existe por lo menos un minimal al cual llamaremos $\mathbf{x_1}$. Puesto que $|\mathbf{L(x_1)}| = 0$ ingresa en la secuencia de la estructura de salida. En forma simultanea se deletea el nodo $\mathbf{x_1}$ con sus respectivas relaciones y quedará un grafo $\mathbf{G_1}$ que también es acíclico. De esta manera $\mathbf{G_1}$ también tiene al menos un minimal al que llamaremos $\mathbf{x_2}$. Prácticamente puede haber varios minimales de los cuales uno será elegido en forma arbitraria. El procedimiento se repite hasta que todos los puntos de G han sido ingresados en la estructura de salida.

Implementación del procedimiento TSORT.

El problema a resolver, básicamente es obtener los minimales de un grafo por lo cual es posible utilizar la representación de la matriz de adyacencia. Este es un método sencillo pero inadecuado para solucionar problemas de actualización dinámica, pues requiere conocer de antemano la cantidad de nodos del grafo. Para solucionar este problema, recurrimos a las estructuras linkeadas. De las estructuras conocidas, la que más se adapta es la representación de Pfaltz, pues al representar el grafo en dos listas, una para nodos y una para arcos, es posible mantener información relacionada con los nodos y con las relaciones entre nodos. Sin embargo el principal problema que presenta esta representación es la complejidad de los algoritmos para mantener la estructura, por lo cuál se utiliza otro procedimiento pero con la misma filosofía de operación que utiliza Pfaltz.

Ejemplo. Consideremos una aplicación donde un jefe de cocina recibe un pedido de huevos fritos. Dispone de un número ilimitado de ayudantes y la consigna es realizar la tarea en el menor tiempo posible. La tarea freir un huevo puede descomponerse en las siguientes subtareas.

A. Conseguir un huevo

B. Conseguir aceite

C. Poner aceite en la sartén

D. Poner el huevo en la sartén

E. Romper el huevo

F. Prender el fuego

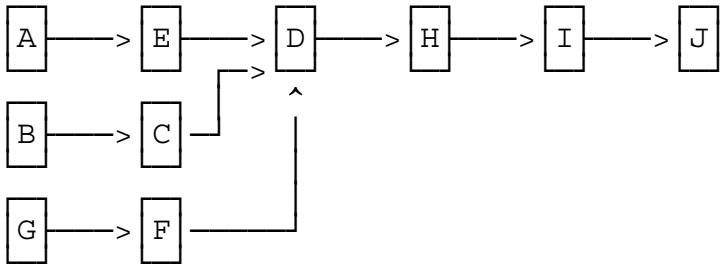
G. Conseguir fósforos

H. Controlar que no se pase

I. Retirar el huevo

J. Servirlo en el plato

Algunas tareas tienen una prioridad de precedencia mayor respecto de otras. Así por ejemplo, se debe romper el huevo antes de ponerlo en la sartén. Otras pueden ejecutarse en forma simultanea, como por ejemplo conseguir un huevo y conseguir el aceite. El problema consiste entonces en asignar las tareas a los ayudantes para cumplir el objetivo en el menor tiempo posible y sin desperdiciar recursos. En este caso los recursos serían los ayudantes. Representamos el problema a través de un grafo donde cada nodo representa una subtarea y cada arco **(x, y)** indica que la subtarea **x** precede a la subtarea **y**



Con lo cual queda representado que para realizar la tarea D deben haber sido completadas previamente las tareas **A, E, B, C, G** y **F** .

Antes de continuar con el desarrollo es necesario aclarar que el grafo debe ser acíclico pues de existir un ciclo para un nodo **x** implicaría que la subtarea **x** no se podría ejecutar hasta que no hubiera terminado la misma subtarea **x** .

Para comenzar el algoritmo, se obtienen los minimales, es decir aquellas subtareas que se pueden realizar en forma inmediata. De esta manera se extraen del grafo los nodos A, B, y G y quedan en la estructura de salida. Así queda identificado el primer período de tiempo y la cantidad máxima de recursos necesarios para concluirla. El menor tiempo posible que demande este primer período será el mayor de los tiempos de las subtareas. Ej. : si para completar A hacen falta 3', para completar B , 6' y para completar G 4' entonces el menor tiempo posible para finalizar este primer período será de 6'. La cantidad máxima de recursos necesarios es 3. Si hubiera más, se desperdiciarían recursos. Si hubiera menos, se perdería tiempo. Vale concluir entonces el análisis de este primer período de tiempo sabiendo que hay tres subtareas que pueden ser ejecutadas en forma simultánea.

Al grafo resultante de la eliminación de los nodos minimales con sus respectivas relaciones, se aplica

nuevamente el algoritmo de obtener los minimales. De esta manera se obtienen **E**, **C** y **F**, es decir aquellas subtareas que pueden realizarse en la misma unidad de tiempo.

Así, sucesivamente se completan las 4 unidades de tiempo restantes, quedando en resumen:

Ayudante 1	***Ayudante 2***	***Ayudante 3***
Conseguir un huevo	Conseguir aceite	Conseguir fósforos
Romper el huevo	Poner aceite en la sartén	Prender el fuego
Poner el huevo en la sartén		
Controlar que no se pase		
Retirar el huevo		
Servirlo en el plato		

```
Program Sortopológico;

const CantNod = 5;
type  Nodos  = array[1..CantNod] of char;
      Matriz = array[1..CantNod,1..CantNod] of integer; {*
Mat.de Adyac.*}
var   Linea  : array[1..CantNod] of char; {* Contiene la
estruc. lineal *}
      c,s,h,i,j,k : integer; Conta : boolean; MAT : Matriz;
Nod : Nodos;

Procedure CargoNodos;
begin
   writeln('Ingreso de Nodos de la estructura -',CantNod,'
','como Max.');
   For i := 1 to CantNod do
      write(' Ingrese Nodo : '); readln(Nod[i]);
end;

Procedure ArmoMatriz;
begin
   For i := 1 to CantNod do
       for j := 1 to CantNod do
           MAT[i,j] := 0;
end;

Procedure Relaciones;
var   eureka, inicio : boolean; DatoX, DatoY : char;
begin
   writeln(' Ingreso de relaciones de la estructura -( "0"
finaliza )');
   Repeat
      eureka := false;
      While eureka = false do
      begin
         i := 1;
         write('Ingrese nodo de donde parte el arco : ');
readln(DatoX);
         while (eureka = false) AND (i <= CantNod) do
         begin
            if (Nod[i] = DatoX) OR (DatoX = '0')
               then eureka := true
               else i := i + 1;
         end;
         if eureka = false
            then begin
                    writeln('* Dato ingresado no _ a la
estructura *');
                    writeln('* ingrese el Nodo del cual
parte  *');
                 end;
      end;
      eureka := false;
      while eureka = false do
      begin
         j := 1;
         write('Ingrese nodo a donde llega el arco : ');
readln(DatoY);
```

```
         while (eureka = false) AND (j<= CantNod) do
         begin
            if (Nod[j] = DatoY) OR (DatoY = '0')
               then eureka := true else j := j + 1;
         end;
         if eureka = false
            then begin
                 writeln('* Dato ingresado no _ a la
                         estructura *');
                 writeln('* ingrese el nodo al cual llega
                         *');
                 end;
      end;
      if (DatoX <> '0') AND (DatoY <> '0')
         then MAT[i,j] := 1;
   until (DatoX = '0') AND (DatoY = '0');
   writeln('Listado de la Matriz de Adyacencia ');
   For c := 1 to CantNod do {* display matriz adyacencia *}
       write(Nod[c]:9);
   for i := 1 to CantNod do
   begin
      write('     ',Nod[i]);
      for j := 1 to CantNod do
          write(' ',MAT[i,j]:8);
   end
end;

Procedure Sortopo;
begin
   s := 1;
   h := 0;
   Repeat
      j := 0;
      Repeat
         j := j + 1;
         i := 1;
         Conta := true;
         While ( Conta = True ) AND ( i <= CantNod ) do
         begin
            if Mat [i,j] = 1
               then Conta := False;
            i := i + 1;
         end;
         if Conta
            then begin
                    h := h + 1;
                    Linea[h]  := Nod[j];
                    For c := 1 to CantNod do
                    begin
                       MAT[c,j] := 1;
                       MAT[j,c] := 0;
                    end;
                    MAT[j,j] := 0;
                 end;
      until j = CantNod;
   until h = CantNod;
end;
```

```
Procedure ListoVector;
begin
   writeln(' Listado del array unidimensional ordenado');
   For i := 1 to CantNod do
       writeln(Linea[i]);
end;

begin       {* program body *}
   CargoNodos;
   ArmoMatriz;
   Relaciones;
   Sortopo;
   ListoVector;
end.
```