Instituto Politécnico Nacional

Escuela Superior de Cómputo

Sesión 8b: Programación Dinámica I

Integrantes:

Escutia López Arturo

27/NOV/2016

Pseudocódigos

BFS O(V+E)

- Marcar Todos los nodos como no visitados <- O(V)

```
BFS (initial node)
  q <- new Queue()
  q.add (initial node)
  visited[initial node]<-true
  while ( q is not empty ) do
        aux<-q.peek()   <- O(1)
        q.poll()        <-O(1)
        for ( every adjacent node (aux,Y))   <-  O(V+E)
              if ( Y has not been visited )
                    q.add(y)          <- O(1)*O(V+E)
                    visited[y] <-true      <-O(1)*O(V+E)
              end if
        end for
  end while
end BFS
```

DFS O(V+E)

- Marcar Todos los nodos como no visitados <-O(V)

```
DFS(V)
  If V is visited
        Return          <-O(1)
  Else
        Visited[V]<-true          <-O(1)
        for ( every adjacent node (V,Y))          <-O(V+E)
              DFS(Y)          <-O(1)* O(V+E)
        End for
  End if
End DFS
```

Topological Order (Kahn´s) O(V+E)

1. Computar el número de entradas de cada vértice <-O(E)
2. Agregar a una cola aquellos vértices sin entradas <-O(V)
3. Desencolar un vértice de la cola <- O(1)
   - Decrementar -1 las entradas de los vértices adyacentes respecto al vértice desencolado <- O(V+E)
   - Si un vértice adyacente ya no tiene entradas,agregar a la cola <-O(1)
4. Repetir paso 3 hasta que la cola este vacía

Topological order (DFS) O(V+E)

- Marcar Todos los nodos como no visitados

```
TO_DFS(V)
   If V is visited
          Return            <-O(1)
   Else
          Visited[V]<-true                  <-O(1)
          for ( every adjacent node (V,Y))          <- O(V+E)
                 DFS(Y)            <-O(1)* O(V+E)
          End for
          Stack.push(Y)             <-O(1)* O(V+E)
  End if
End DFS
```

- Imprimimos el contenido de la pila

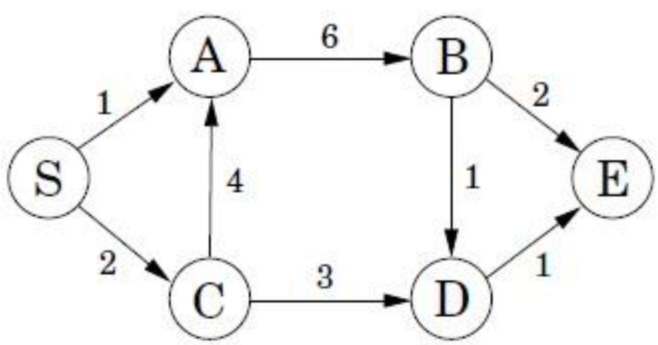
LIS(Largest Increasing subsequence) O(n^2)

```
LIS(int x[])
   input x[]
   n ← length of x
   for i<-1 to n             <-O(n)
      for  j<-0 to i          <-O(n)*O(n)
        if x[i] ≥ x[j] then            <-O(1)*O(n^2)
           dis[i] ← max(dis[i],1+dis[j])
           solution[i]<-j
        end if
     end for
   end for
end LIS
```

DAG Shortest Path

```java
public void Path(int Start){
    Arrays.fill(distance,Integer.MAX_VALUE);
    distance[Start]=0;
    for(int i=TL.indexOf(Start);i<TL.size(); i++){
        int aux=TL.get(i);
        for(int j=0; j<Adj_L.get(aux).size();j++){
            int v=Adj_L.get(aux).get(j).getV();
            int w=Adj_L.get(aux).get(j).getW();
            if(distance[v]>(distance[aux]+w))
                distance[v]=(distance[aux]+w);
        }
    }
    System.out.print("Start:"+Start+"\n");
    for(int i=0;i<distance.length;i++ ){
        if(distance[i]==Integer.MAX_VALUE)
            System.out.println(i+":INF");
        else
            System.out.println(i+":"+distance[i]);
    }

}


}
```

En esta función se llenan las distancias de todos los vértices en un valor muy grande para simular el valor de infinito, se recibe el nodo de inicio y la distancia de ese nodo la cambiamos a 0.Despues recorremos la lista de orden topológico a partir del índice de nuestro vértice inicial.Por cada nodo en la lista de orden topológico buscamos sus adyacencias, si la distancia del nodo anterior (V) más la distancia para llegar de (V,U) es menor a la distancia actual en U se sustituye la distancia.



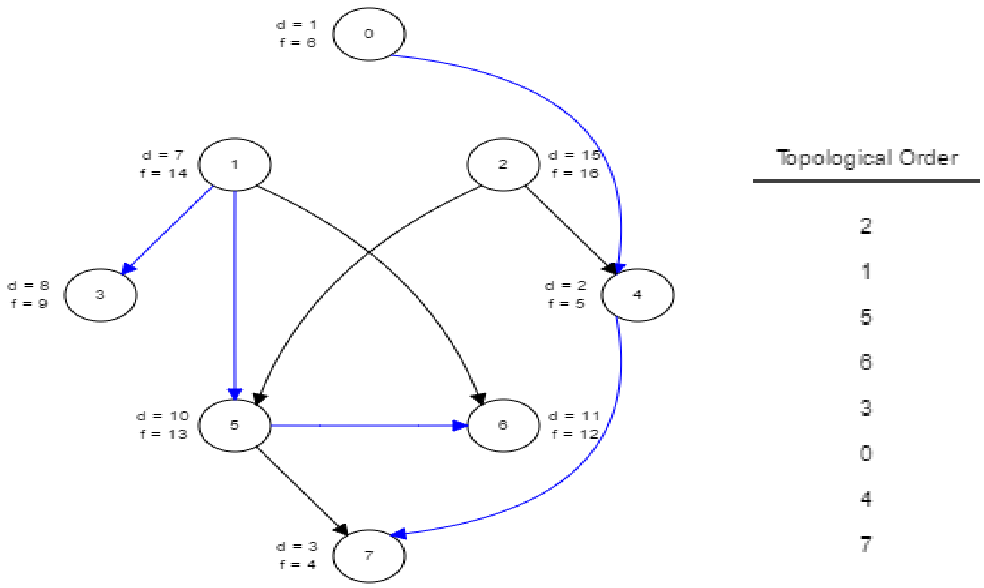
S=1 A=2 B=3 C=4 D=5 E=6

```
Output - orden_topologico (run)    shortesP_TO.java    file

run:

Print all adjacency lists with corresponding vertex:
0:
1: V:2 W:1 | V:4 W:2 |
2: V:3 W:6 |
3: V:6 W:2 | V:5 W:1 |
4: V:2 W:4 | V:5 W:3 |
5: V:6 W:1 |
6:

TOPOLOGICAL ORDER:[0, 1, 4, 2, 3, 5, 6]

Start:1
0:INF
1:0
2:1
3:7
4:2
5:5
6:6
BUILD SUCCESSFUL (total time: 2 seconds)
```

Topological Order with DFS

```java
private void Topological(){
    for(int i=0;i<Adj_L.size();i++)
        if(visited[i]==false)
            DFS(i);
    while(!stack.isEmpty())
        System.out.print(stack.pop()+" ");
}

private void DFS(int V){
    if(visited[V]==true) return;
    visited[V]=true;
    for(int i=0;i<Adj_L.get(V).size();i++)
        DFS(Adj_L.get(V).get(i));
    stack.push(V);
}
```

Es exactamente el mismo procedimiento que una DFS normal, llamamos recursivamente a la función por cada nodo adyacente del nodo actual para llegar a lo más profundo del grafo, pero en este caso nos apoyamos de una pila donde se irán almacenando los vértices no visitados conforme a las llamadas recursivas, finalmente solo queda imprimir el contenido de la pila para obtener el orden topológico




```
Output - orden_topologico (run)    DFS_TO.java    file
run:
2 1 5 6 3 0 4 7 BUILD SUCCESSFUL (total time: 2 seconds)
```

Longest Incremental Subsequence

```java
public void longestSubsequence(int arr[]){

    int distance[] = new int[arr.length];
    int Solution[] = new int[arr.length];

    for(int i=0; i < arr.length; i++){
        distance[i] = 1;
        Solution[i] = i;
    }
    for(int i=1; i < arr.length; i++){
        for(int j=0; j < i; j++){
            if(arr[i] > arr[j]){
                if(distance[j] + 1 > distance[i]){
                    distance[i] = distance[j] + 1;
                    Solution[i] = j; //set the actual
                }
            }
        }
    }

    //find the index of max number in T
    int maxIndex = 0;
    for(int i=0; i < distance.length; i++){
        if(distance[i] > distance[maxIndex])
            maxIndex = i;
    }

    int t = maxIndex;
    int newT = maxIndex;
    Stack s=new Stack();
    System.out.print("Subsequence ");
    do{
        t = newT;
        s.push(arr[t]);
        newT = Solution[t];
    }while(t != newT);

    while(!s.isEmpty())
        System.out.print(s.pop()+" ");

}
```

Se recibe el arreglo con la secuencia de números original, se hace uso de dos arreglos, uno para las distancias y otro para guardar los índices del arreglo original que contienen la solución. Se recorre de inicio a fin el arreglo, y por cada número se compara con aquellos número anteriores a éste de forma que $j<i$ y $aj<ai$, si la distancia actual es menor a la distancia del número anterior +1 se sustituye la distancia actual ya que nos interesa la distancia máxima en éste caso no la mínima, y se guarda en el otro arreglo el índice j para poder imprimir después la solución.