

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)

1. 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

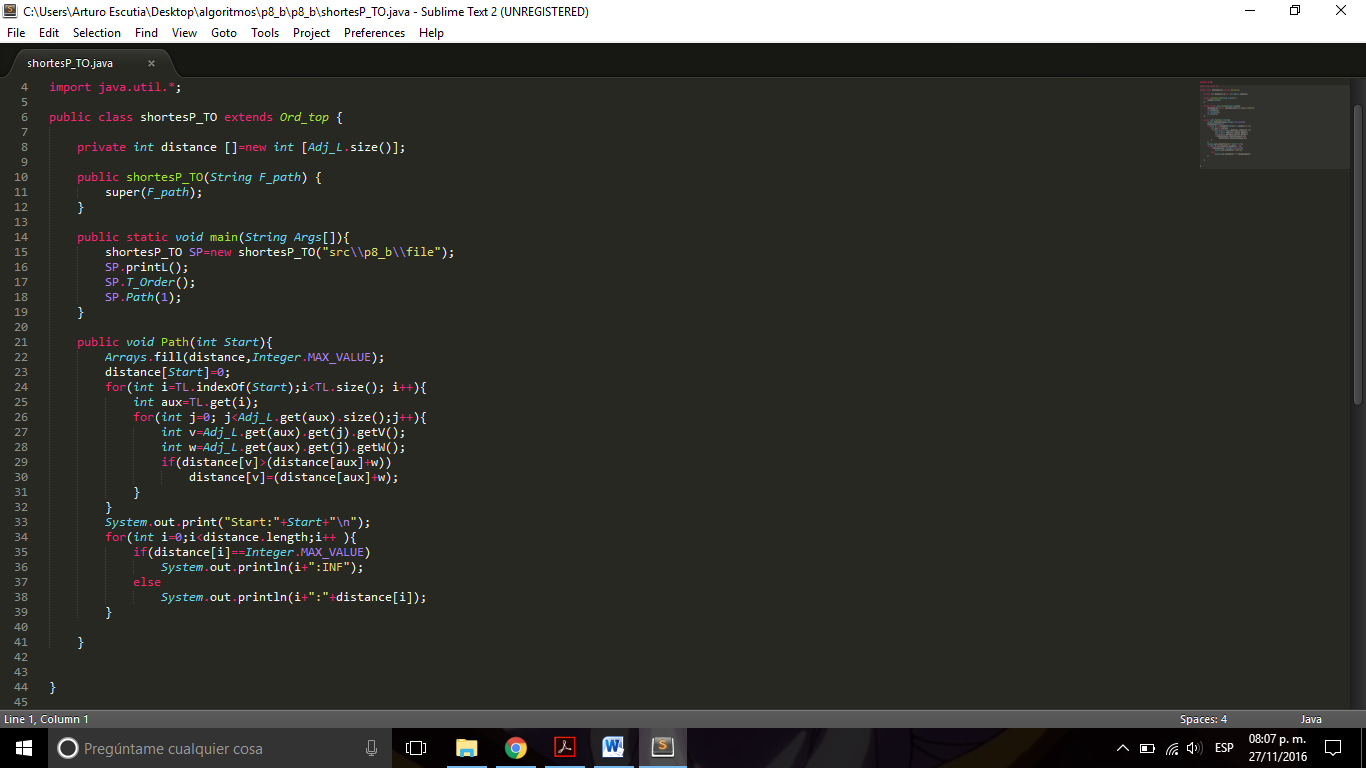
end if

end for

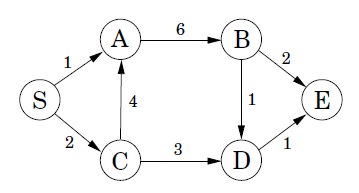
end for

end LIS

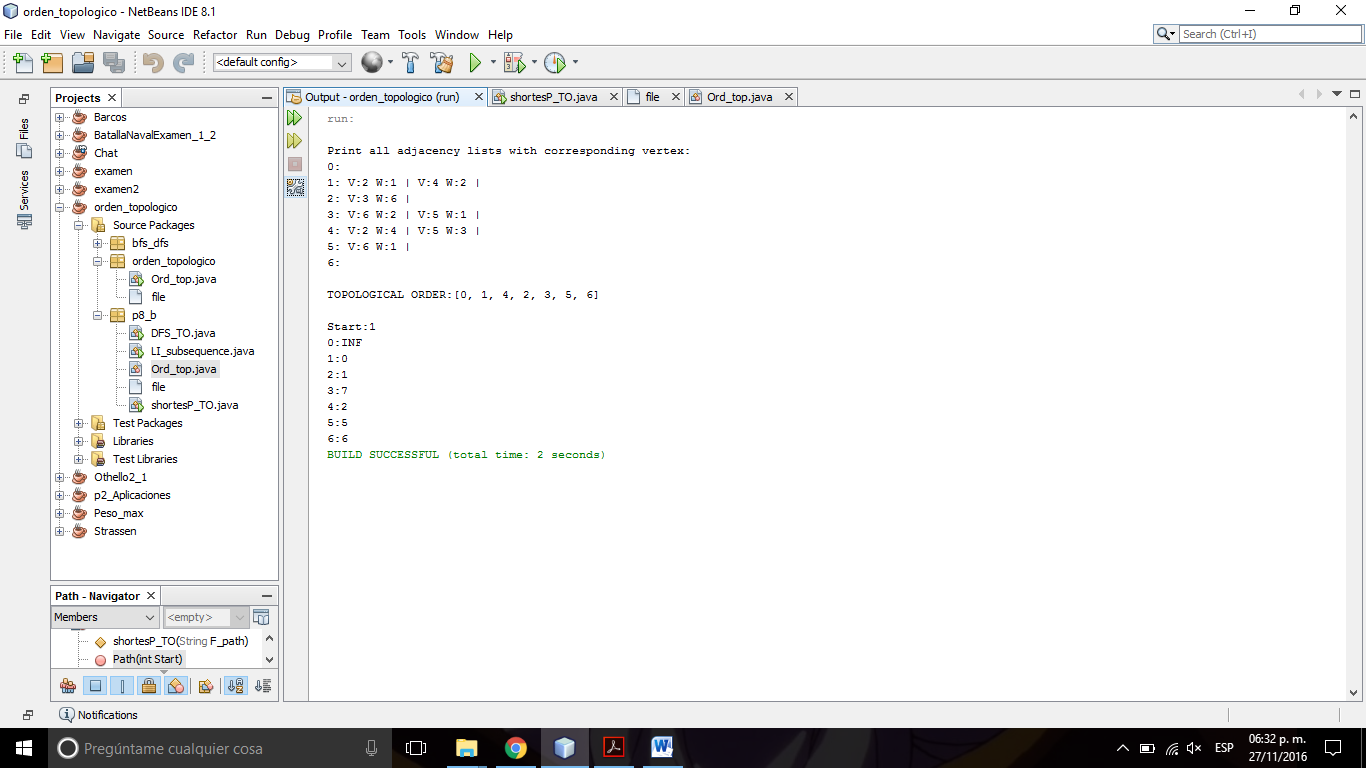
**DAG Shortest Path**



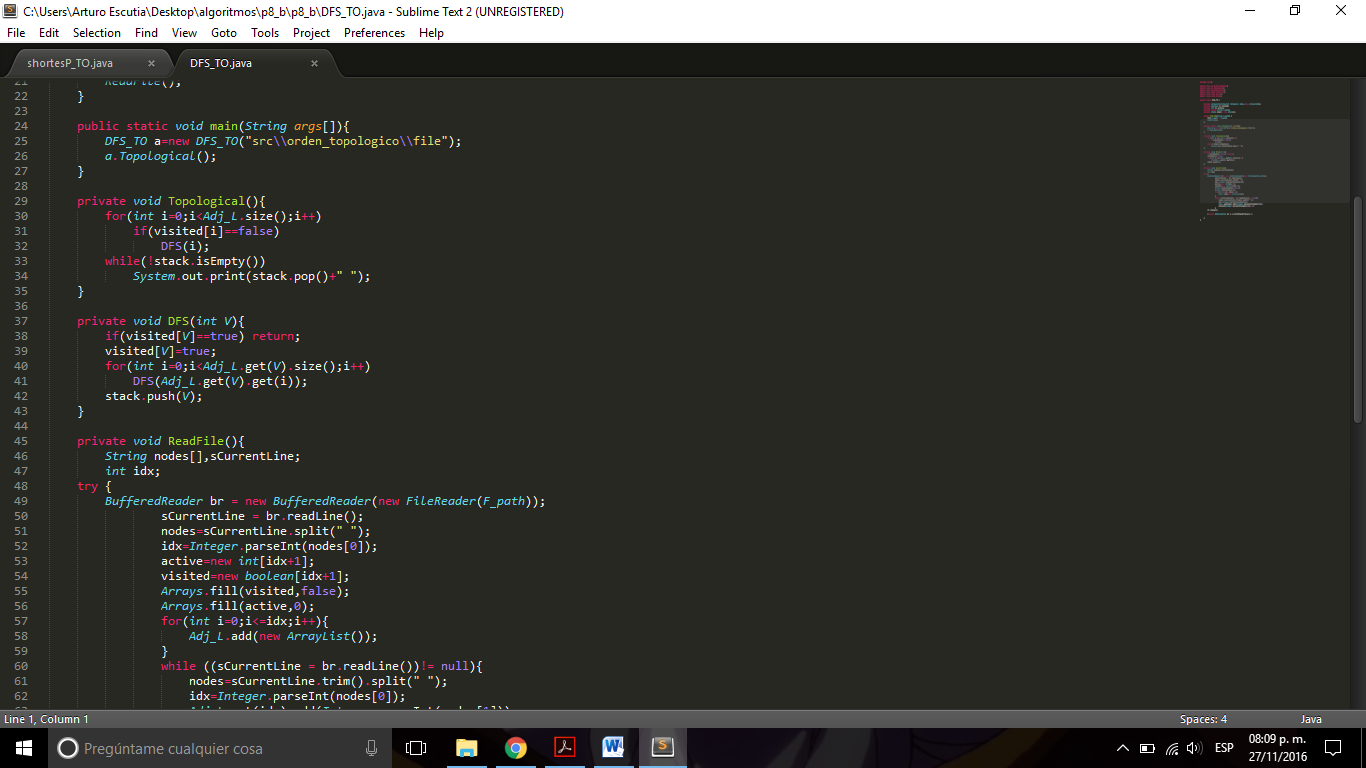
En esta función se llenan las distancias de todos los vértices en un valor muy grande para simular el valor de infinito, se recibe el nodo de inicio y la distancia de ese nodo la cambiamos a 0.Despues recorremos la lista de orden topológico a partir del índice de nuestro vértice inicial.Por cada nodo 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.



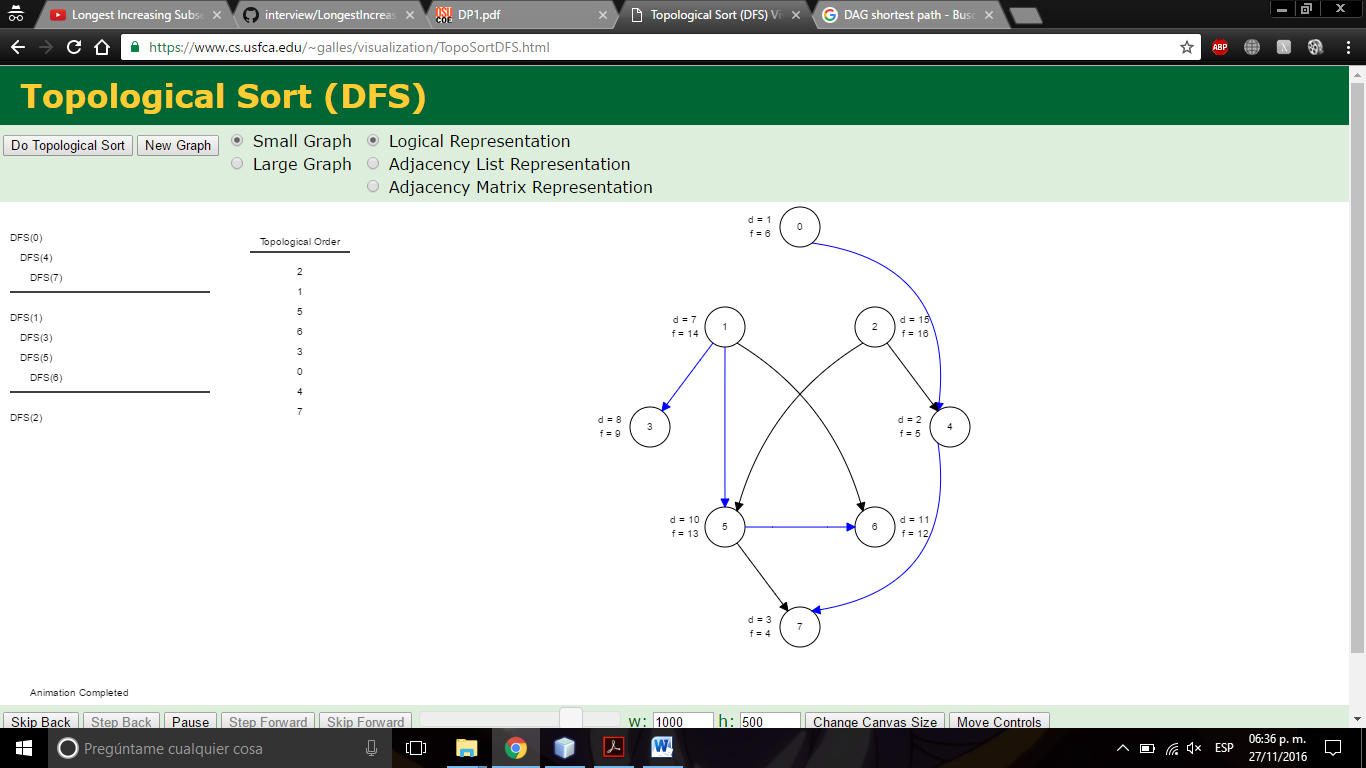
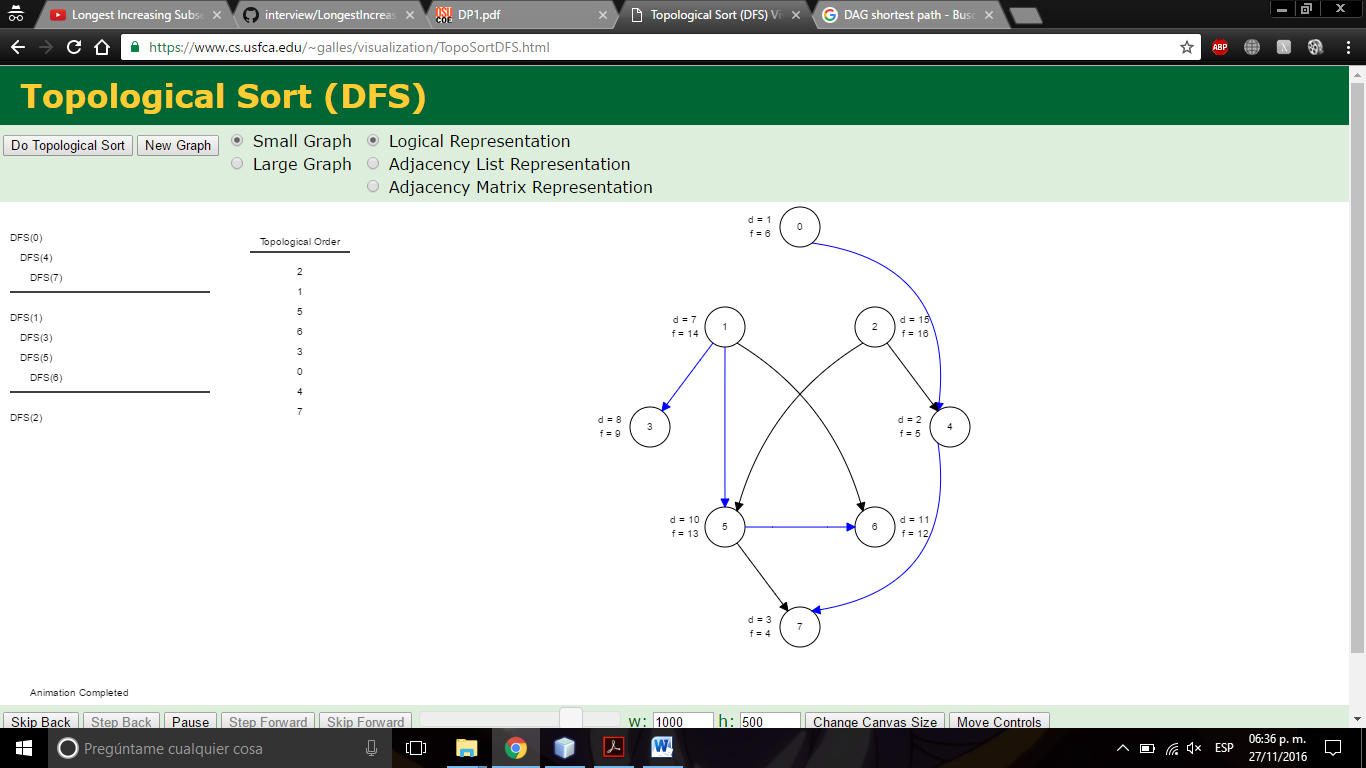
S=1 A=2 B=3 C=4 D=5 E=6

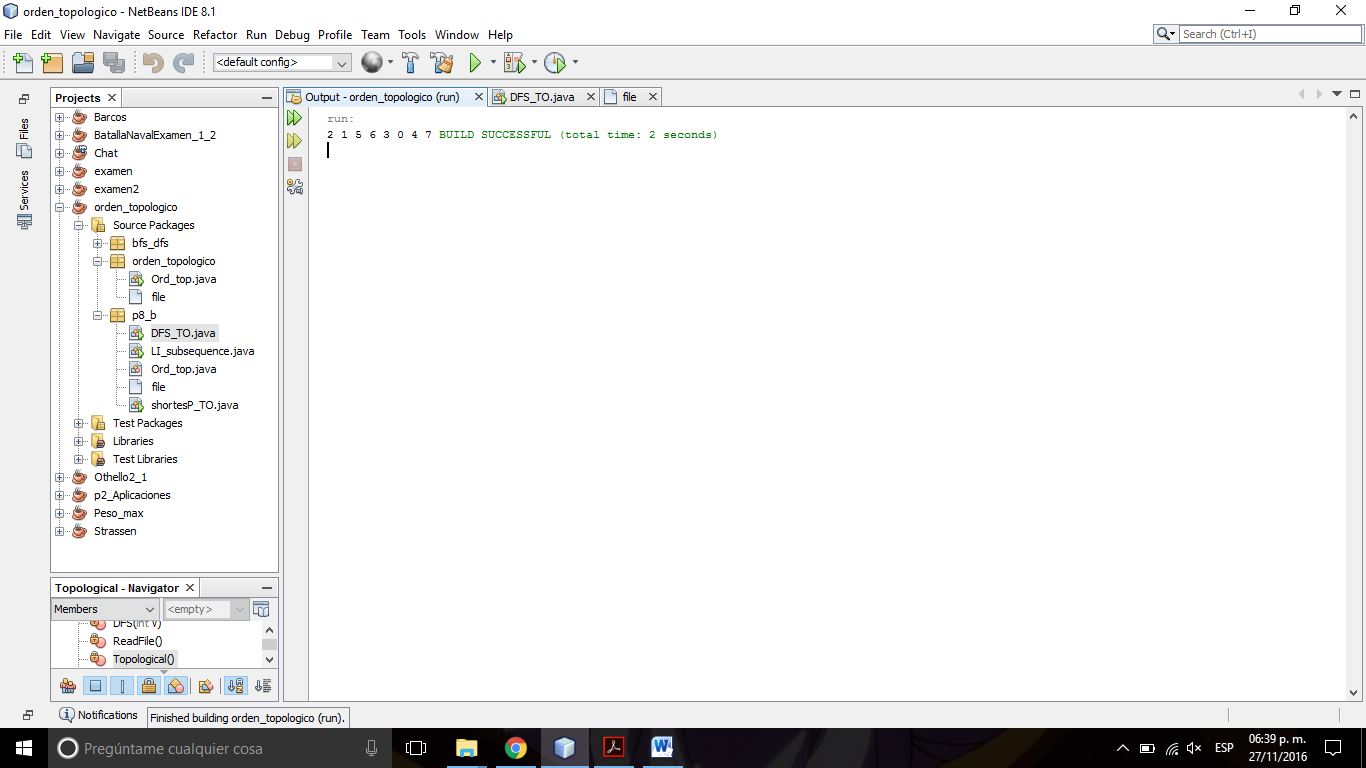


**Topological Order with DFS**

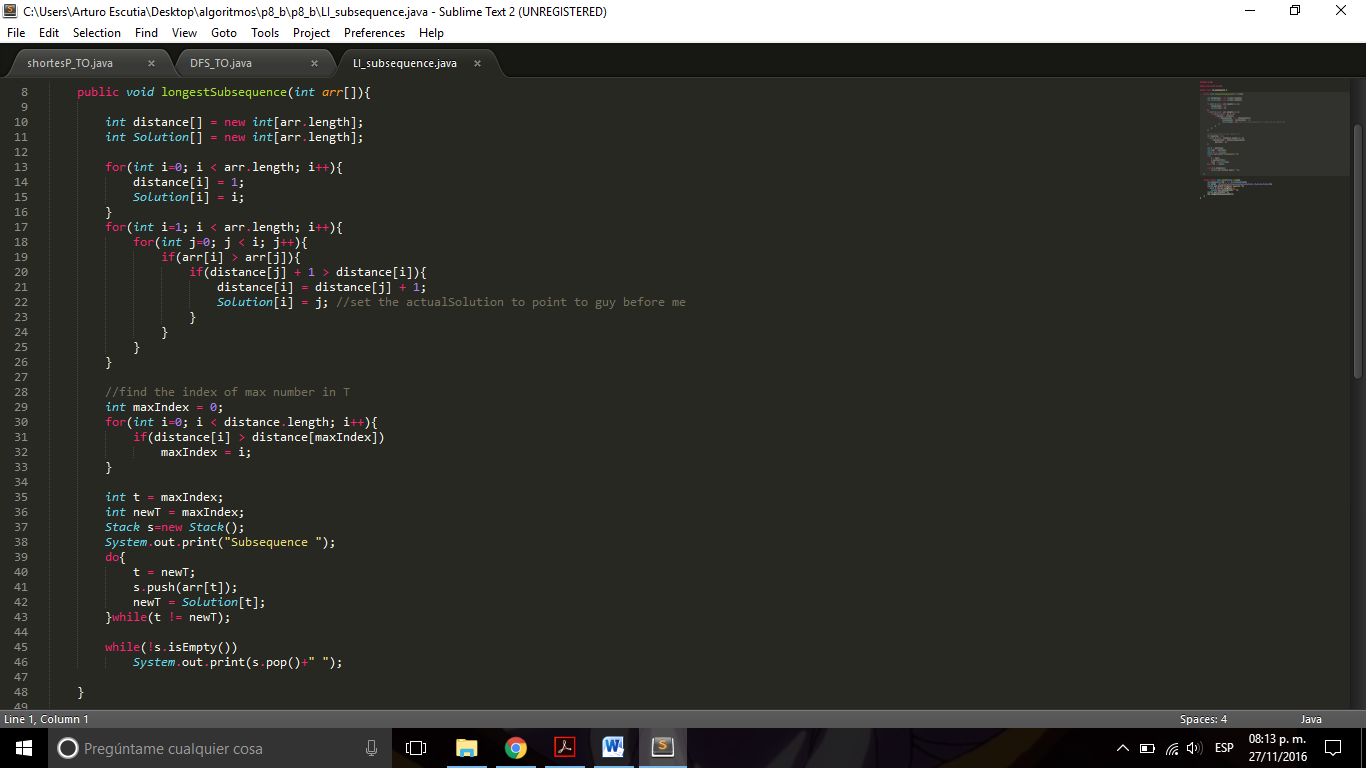


Es exactamente el mismo procedimiento que una DFS normal, llamamos recursivamente a la función por cada nodo adyacente del nodo actual para llegar a lo más profundo del grafo, pero en este caso nos apoyamos de una pila donde se irán almacenando los vértices no visitados conforme a las llamadas recursivas,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





**Longest Incremental Subsequence**



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.

