**Clase Dijkstra**

**Linea 1:** se importa la clase ArrayList de java.util.

**Linea 11:** se declara adjMatrix como una matriz de double.

**Linea 13:** se declara distances como un array de double.

**Linea** **14:** se declara visited como un array de boolean.

**Linea** **15:** se declara prev como un array de enteros.

**Linea 16:** se declara nVertices como un entero.

**Linea 18:** se declara error como un boolean y se inicializa a false.

**Linea 19:** se declara dijkstraExec como un boolean.

**Lineas 11-19:** declaración de todos los atributos de la clase.

**------------------------------------------------------------------------------------------------------------------------**

**Función Dijkstra()**

**Linea 42:** iguala la variable adjMatrix al atributo de la clase adjMatrix.

**Linea 43:** iguala la variable nVertices al atributo de la clase nVertices.

**Lineas 41-44:** la función es el constructor de la clase con dos parámetros: adjMatrix y nVertices.

Función initializeDataStructures()

**Linea 52:** inicializa el vector distances

**Linea 53:** inicializa el vector visited

**Linea 54:** inicializa el vector prev

**Linea 58:** iguala distances[i] a infinito

**Linea 59:** iguala visited[i] a false

**Linea 60:** iguala prev[i] a -1

**Linea 57-61:** Recorre todos los elementos de las estructuras de datos y las Inicializa, iguala cada elemento de distances, visited y prev a sus valores iniciales.

**Líneas 50-64:** Es una función que inicializa todas las estructuras de datos de la clase y le pone sus valores iniciales a cada elemento de estas estructuras

**------------------------------------------------------------------------------------------------------------------------**

**Función nextCur()**

**Linea 79:** asigna el valor de i a la variable next.

**Linea 80:** obtiene el valor i-ésimo de distances y lo asigna a cost.

**Líneas 78-81:** si la variable visitado en la posición i esta en false y la distancia de i es menor que la variable cost, entonces asignamos el valor de i a la variable next y el valor de distancia de i en la variable cost.

**Linea 74:** iniciamos la variable next igual a -1.

**Linea 75:** iniciamos la variable cost igual a mas infinito.

**Lineas 77-82:** se recorre el bucle for nVertices número de veces y en cada iteración se comprueba si la variable visited de i es igual a false y el valor de distances de i es menor que cost y en caso positivo asignamos el valor de i a next y el valor de distances de i a cost.

**Linea 84:** devuelve el valor de next.

**Líneas 73-85:** la función de entre todos los posibles nodos busca el nodo que no ha sido visitado y tenga menor coste y lo devuelve para que sea el siguiente en visitar.

**------------------------------------------------------------------------------------------------------------------------**

**Función computeShortestPath()**

**Linea 108:** Inicializa la variable newDistance

**Linea 109:** Llama la función initializeDataStructures()

**Linea 111:** Inicializa el vertice inicial, iguala cur al parámetro ini que reciba la función

Iguala la distancia inicial a si misma a 0.0

**Linea 112:**Linea 120: Iguala newDistance a la distancia del elemento actual (cur) + el coste de viaje de su elemento vecino (adjMatrix[cur][neigh])

**Linea 124:** Iguala la distancia del elemento vecino con newDistance

**Linea 125:** Iguala al elemento previo prev[neigh] al elemento actual (cur)

**Linea 130:** Marca como visitado el elemento actual, Iguala visited[cur] a true

**Linea 131**: iguala elemento actual (cur) al siguiente elemento (nextCur())

**Linea 134:** iguala dijkstraExec a true para demostrar que el algoritmo Dijkstra se ha ejecutado

**Linea 135:** Retorna la distancia del elemento final (distances[end]

**Línea 123-126:** Si la distancia actual del elemento vecino es mayor a la nueva distancia calculada (newDistance) iguala esta distancia actual a la nueva distancia y el elemento previo del vecino al actual

**Línea 118-127:** Si el elemento actual tiene al elemento neigh como vecino (adjMatrix[cur][neigh]>0) y este elemento neigh no ha sido visitado calcula la nueva distancia entre ellos, y si es menor que la distancia calculada previamente lo cambia por esta nueva distancia y iguala el elemento previo del vecino al actual.

**Línea 115-128:** Se comprueba para cada elemento del grafo si es vecino del elemento actual (cur) y si lo es y no ha sido visitado previamente, calcula la distancia entre ellos que en caso de que sea menor que la distancia calculada previamente la reemplaza y iguala el elemento previo del vecino al actual.

**Linea 113-132:** Mientras no se halla visitado el elemento final (end) se hace todo mencionado previamente y al final se marca el elemento actual como visitado y el elemento actual pasa a ser otro

**Línea 107-136:** La función representa el algoritmo de Dijkstra para calcular el camino más corto de un elemento inicial a uno final de un grafo retornando al final la distancia más corta calculada entre estos dos elementos

**------------------------------------------------------------------------------------------------------------------------**

**Función getPath()**

**Linea 155:** iguala la variable error a true.

**Linea 156:** devuelve null.

**Linea 166:** se llama a la función add de path con el parametro prev de cur.

**Linea 167:** iguala la variable cur al valor de prev de cur.

**Lineas 154-157:** si la variable dijkstraExec es igual a false, iguala maos la variable error a true y devolvemos null y terminando la función.

**Linea 159:** inicializa la variable path como un arraylist de enteros.

**Linea 163:** inicializa la variable cur igual a la variable end.

**Linea 164:** llamamos a la función add de path con parametro la variable cur.

**Líneas 165-168:** mientras la variable cur sea distinta de ini se llama a la función add de path con el valor de prev de cur y asignamos el valor de prev de cur a cur.

**Linea 170:** devuelve la variable path.

**Líneas 153-171:** en esta función si el valor de dijkstraExec es false ponemos la variable error a treu y devolvemos null. En caso contrario creamos un arraylist path que va a recorrer todos los nodos que haya entre end y ini y devolvemos dicha variable que contiene todos los nodos que hay para llegar desde ini a end.

**------------------------------------------------------------------------------------------------------------------------**

**Función hasErrorHappened()**

**Linea 180:** devuelve el valor de la variable error.

**Lineas 179-181:** la función devuelve el valor de la variable error.

**------------------------------------------------------------------------------------------------------------------------**

**Clase Dijkstra.java:**  
La clase implementa el algoritmo de Dijkstra. Recibe como parámetros el número de elementos del grafo y una matriz que representa la adyacencia entre estos elementos para encontrar el camino más corto de un elemento inicial a otro denominado como elemento final. La clase retorna dos salidas importantes, primero la distancia calculada del elemento inicial al final y los otros elementos recorridos para encontrar este camino más corto.