Reading by abstraction: Dijkstra

**Variables**

Línea 11: Se crea la matriz de tipo Double llamada *adjMatrix*.

Línea 13: Se crea el array de tipo Double llamada *distances*.

Línea 14: Se crea el array de tipo boolean llamado *visited*.

Línea 15: Se crea el array de tipo Integer llamado *prev*.

Línea 16: Se crea la variable de tipo Integer llamado *nVertices*.

Línea 18: Se crea la variable de tipo boolean llamada *error* y se inicializa a false.

Línea 19: Se crea la variable de tipo boolean llamada *dijkstraExec*.

**Constructor**

Línea 42: Inicializa la variable *adjMatrix*.

Línea 43: Inicializa la variable *nVertices*.

Línea 41-44: Constructor de la clase.

**initializeDataStructures**

Línea 52: Inicializa la variable *distances*.

Línea 53: Inicializa la variable *visited*.

Línea 54: Inicializa l variable *prev*.

Línea 58: Pone el valor de la variable *distances* en la posición *i* un valor infinito.

Línea 59: Pone el valor de la variable *visited* en la posición *i* a false.

Línea 60: Pone el valor de la variable *prev* en la posición *i* a -1.

Línea 57: Es un bucle que recorre todos lo vértices del grafo e inicializa sus valores.

Línea 50-64: Inicializa las variables globales de la clase.

**nextCur**

Línea 74: Crea la variable *next* de tipo Integery la inicializa a -1.

Línea 75: Crea la variable cost de tipo Double y la inicializa a infinito.

Línea 79: Pone el valor de la variable *next* a *i*.

Línea 80: Pone el valor de la variable cost a *distances* en la posición *i*.

Línea 78: Comprueba si la no se ha visitado la posición actual y la distancia actual es menor que el coste.

Línea 77: Recorre todas las posiciones del grafo y si no se ha visitado la posición actual y la distancia es menor que el coste, establece que *next* es igual a la posición actual y el coste a la distancia en la misma posición.

Línea 84: Devuelve el valor de *next*.

Línea 73-85: Devuelva el siguiente vértice no visitado del grafo cuyo coste sea menor.

**computeShortestPath**

Línea 108: Crea la variable *newDistance*.

Línea 109: Se llama a la función que inicializa las variables globales.

Línea 111: Crea la variable cur y la inicializa al vértice inicial.

Línea 112: Pone el valor de la distancia en la posición inicial igual a 0.

Línea 124: Guarda el valor de la nueva distancia.

Línea 125: Guarda el vértice actual junto a los vértices previos.

Línea 123: Si la distancia actual es mayor a la nueva, se actualiza.

Línea 120: La distancia nueva es igual a la distancia a ir a la posición vecina.

Línea 118: Si el nodo actual está conectado con el vecino y no se ha visitado entra en el if.

Línea 115: Recorre todos los nodos.

Línea 113: Un while que recorre todos los nodos no visitados hasta el final.

Línea 130: La variable visited en la posición actual pasa a valer true.

Línea 131: Pasa al siguiente nodo.

Línea 134: La variable dijkstraExec se instancia como ejecutado.

Línea 135: Retornamos la mejor distancia.

Línea 107-136: La función selecciona el camino más corto entre dos nodos.

**getPath**

Línea 155: Pone la variable *error* a true.

Línea 156: Devuelve null.

Línea 154: Si no se ha ejecutado el algoritmo se establece un error.

Línea 159: Crea la variable *path* de tipo ArrayList y lo inicializa.

Línea 163: Crea la variable *cur* de tipo Integer y lo inicializa la última posición.

Línea 164: Añade *cur* en el array *path*.

Línea 166: Añade el vértice anterior al array del camino.

Línea 167: Establece el nodo actual al anterior.

Línea 165: Recorre el camino mientras no esté en la posición inicial.

Línea 170: Devuelve el camino de luz.

Línea 153-171: Devuelve un ArrayList con todos lo nodos del camino más corto.