Pseudocódigos

Para el BFS:

Inicio de *la función BFS* (recibo parámetro **grafo**)

Declarar variable Tiempo_Inicio = FuncionQueMeDaElTiempo()

Declarar lista recorrido = vacío

Declarar cola = agregamos el origen a la cola

Mientras la cola *no esté vacía*

Declarar variable_auxiliar= *función para encolar*

Agregar elementos de variable_auxilar a la lista recorrido

Si la variable_auxiliar encuentra el nodo destino

Declarar variable

Tiempo_Termino= FuncionQueMeDaElTiempo()

Retornar lista recorrido,

(El Tiempo_Inicio – Tiempo_Termino) *100

Para vértice en el rango del tamaño de la matriz de grafo

Si el valor en la matriz de grafo del nodo actual es vacio
//es decir sin visitar

Declarar vertice_candidato= vértice actual de grafo
//se convierte en el nodo candidato a visitar

Si el vértice no esta en ninguna de nuestras listas
//para verificar

Agregar el vertice_candidato a la cola

Retornar recorrido,-1

Para el DFS:

Inicio de la *función* DFS (recibo como parámetro **grafo**)

Declarar Tiempo_Inicio = FuncionQueMeDaElTiempo()

Declarar lista_recorrido = vacio

Declarar pila = Agregamos el origen

Mientras la pila *no este vacia*

Declarar variable_auxiliar = funcion meter elementos a la pila()

Agregar elementos de variable_auxilar a la lista recorrido

Si la variable_auxiliar encuentra el nodo destino

Declarar variable Tiempo_Termino= FuncionQueMeDaElTiempo()

Retornar lista recorrido, (El Tiempo_Inicio – Tiempo_Termino) *100

Si la variable_auxiliar no esta en el recorrido

Agregamos la variable_auxiliar al recorrido

Declarar condición = verdadera

Para cada vertice en rango del tamaño de la matriz

Si el valor en la matriz de grafo del nodo actual es vacio //es decir sin visitar

Declarar vertice_candidato= vértice actual de grafo //se convierte en el nodo candidato a visitar

Si el vértice no está en ninguna de nuestras listas //para verificar

Condición = False

Agregar la variable auxiliar a la pila

Agregar la variable del vértice candidato

Declarar Tiempo_terminado = FuncionQueMeDaElTiempo()

Retornar recorrido, -1

Para A* Star:

Inicio de la *función* A* (recibo como parámetro **grafo**)
Declarar Tiempo_Inicio = FuncionQueMeDaElTiempo()
Declarar lista_recorrido abierta y cerrada = vacio
Declarar nodo inicial = obtenernodoInicio()
Declarar nodo objetivo = ObtenerNodoFinal()
Declarar Lista_abierta (nodo inicial)
Mientras que la lista_abierta no este vacia
    Declarar Lista_abierta()
    Declarar nodo actual = Lista_abierta(0)
    Declarar Lista_cerrada(Nodo actual)
    Si nodo actual == nodo objetivo
        Declarar lista Camino = vacio
        Mientras nodo actual ¡= nodo inicial
            Declarar Camino(nodo actualNombre)
            Declarar nodo actual = nodo actualPadre
        Declarar Caminoadjunto(nodo inicialNombre)
        Declarar Tiempo_Termino = FuncionQueMeDaElTiempo()
        Declarar Retoran camino -1,(Tiempo_Termino-Tiempoinicio)*1000
    Declarar Vecino = grafico.tomar(nodo actualNombre)
    Para Llave, evalua a vecino.objetos()
        Declarar Vecinos= nodo(Llave, nodo actual)
        Si Vecinos esta en Lista_cerrada
            Continuar:

```
//Calculamos el camino completo del costo
Declarar Vecino.g =
nodo actual.g +grafico.tomar(nodo    actualNombre, vecinoNom
bre)
Declarar Vecino.h = g.tomarHeuristico().tomar(vecinoNombre)
Declarar vecino.f = vecino.g + vecino.h
 Si el vecino en la lista_abierta == verdadero
        Declarar Lista_abierta(Vecino)
Declarar Tiempo_Termino = FuncionQueMeDaElTiempo ()
Retronar -1
```