buscar_camino : Esta funcion tiene como objetivo encontrar el camino más corte entre dos nodos en el grafo del metro. Para hacer esto se utiliza un variante del algoritmo de Dijkstra llamado algoritmo de busqueda A*. Es el algoritmo de Dijksta con la adicion de una funcion heuristica para añadir precisión y no tener que explorar nodos/caminos de forma innecesaria, pero manteniendo la busqueda del camino más corto.

En el algoritmo definimos el camino más corto como el camino con menos metros desplazados, por lo tanto el peso de las aristas será la distancia entre su nodo de origen y su nodo de destino. Utilizaremos las coordenadas proporcionadas de cada estacion para calcular la distancia entre cada nodo.

$$f(n) = g(n) + h(n)$$

Donde n es el nodo siguiente, $g(n)$ es el calculo del coste (en metros) desde el origen hasta n, $h(n)$ es la funcion heuristica y $f(n)$ es la función estimatoria de la distancia entre el nodo sigueinte (n) y el destino

Para calcular $f(n)$ se requiere la función heuristica $h(n)$. En este proyecto decidimos utilizar las distancia euclidiana entre el origen y el destino como función heuristica.

Distancia euclidiana entre dos puntos:

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - q_i)^2}$$

Simplificamos en este caso ya que el grafo es de dos dimensiones

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Donde d es la distancia entre los puntos con coordenadas (x_1, y_1) y (x_2, y_2).

Para visualizar este proceso, tomemos la estación *Lago* como origen y *Gran Via* como Destino. Primero se exploran los nodos vecinos a *Lago*, *Batán* y *Principe Pio.* La linea roja que conecta la estacion origen con su vecino es el cálculo de la funcion $g(n)$ el coste entre el origen y el nodo siguiente. La linea morada que conecta los nodos siguiente con el destino, son la distancia euclidiana entre las estaciones, la funcion heuristica $h(n)$. La suma de ambas expresiones produce el calculo estimatorio. De los dos nodos, el nodo con el

valor $f(n)$ menor será el que se continuará explorando y el proceso se repetirá con ese nodo.

En el codigo este funcionamiento se implementa con el uso de pilas, ***open_set*** es la pila que contiene los nombres y el y coste desde el origen de cada nodo que será explorado. En cambio ***closed_set*** contiene el nombre de los nodos ya explorados. El bucle se repetirá siempre y cuando exista un elemento dentro de ***open_set.*** Una vez se comienza el bucle, se selecciona *current_node* el nodo que será explorado; se selecciona este nodo ordenando la pila de ***open_set*** con una funcion anonima para extraer el elemento con el menor $f(n)$. En otras palabras, dentro de la funcion anonima, para todos los nodos **n** dentro de ***open_set*** se calcula su distancia euclidiana desde **n** hasta el destino ($h(n)$) y se le suma el coste desde el origen hasta **n** ($g(n)$)**.** Basandose en esa suma el que tenga el menor valor es igualado a *current_node.*

Dentro del bucle, se elimina *current_node* de la pila ***open_set*** y se añade a ***closed_set***.

Por cada vecino a ese nodo, se calcula el coste de ese vecino $g(vecino)$ (en el caso que ese vecino este dentro de ***closed_set****, osea se ha procesado ya, se salta ese vecino)* se suma $g(n)$ más la distancia euclidiana entre n y $vecino$. En el caso que el vecino no este en ***open_set*** se añade a la lista para ser procesado, en el caso que ya esta en la pila, se compara si el valor potencial de $g(vecino)$ es menor que el que ya esta guardado. En el caso que si es menor, significa que se ha encontrado una forma más rapida de llegar a ese nodo y por lo tanto se sobre escribe su información en la pila.

El bucle se finaliza cuando *current_node* = *destino.*