TB1 - Inteligencia Artificial

Universidad Peruana de Ciencias Aplicadas

Carrera:
Ciencias de la Computación

Periodo:
2021-02

Curso:
Inteligencia artificial

Profesor:
Reyes Silva, Patricia

Nombre de la aplicación:
Red de aeropuertos

Integrantes:

Sergio Antonio Nuñez Lazo	- U201910357
Jack Yefri Cruz Mamani	- U201912400
Oscar Daniel Flores Palermo	- U201716498

Agosto - 2021

Índice

Planteamiento del problema a resolver

Nuestro problema se basa en una red de aeropuertos y la duración de viaje entre ellos, el objetivo es encontrar el vuelo o el conjunto de vuelos que resultaría en la menor cantidad de tiempo de vuelo. Además, el programa mostrará la cantidad de kilómetros recorridos durante todos los vuelos que hemos tomado para llegar a nuestro destino.

Para este trabajo solo usaremos aeropuertos y vuelos nacionales, esto con el objetivo de no tener que esperar tanto a que el algoritmo dé una respuesta, pero este proyecto podría ser implementado en un futuro para más aeropuertos para nivel internacional, o tambien se podria tomar en cuenta viajes terrestres para poder aumentar el número de ciudades y pueblos del país que se analizarían.

El programa ya tendría codificado un grafo en el que los nodos representan los aeropuertos y el peso de las aristas representa el tiempo de vuelo entre 2 aeropuertos. En el código mostramos un ejemplo de la aplicación usando valores predeterminados, pero también hay una interfaz donde el usuario puede ingresar una ciudad de entrada y una ciudad de llegada para que el programa pueda dar resultados a más casos. Finalmente el programa muestra una tabla donde te dice los vuelos que tienes que tomar.

Fundamentación del uso del algoritmo de IA

En Días y otros (1996) definen:

> "Un método heurístico es un procedimiento para resolver un problema de optimización bien definido mediante un aproximación intuitiva, en la que la estructura del problema se utiliza de forma inteligente para obtener una buena solución"

Sobre la cita anterior, se puede definir optimización como el proceso de tratar de encontrar la mejor solución posible para un determinado problema, estos problemas se pueden expresar como encontrar el valor de una de las *variables de decisión* para que una *función objetivo* alcanza su valor máximo o mínimo.

Dado la problemática de la red de aeropuertos, donde el objetivo es encontrar la ruta más corta, podemos aplicar distintos algoritmos de camino corto. Para el presente proyecto se usó *A star* el cuál es una combinación entre búsquedas de tipo *Breadth First Search (BFS)* y *Depth First Search (DFS).* Esto debido a que el $h'(n)$ se asemeja al *DFS* y el $g(n)$, al *BFS.* Luego, se cambia la dirección de procesamiento cuando se encuentran nodos más convenientes.

Se usa el algoritmo "A asterisco", "A estrella" o "A star", que se encuentra dentro de los algoritmos de búsqueda de grafos (Dijkstra, Prim, Kruskal, etc). Este algoritmo sienta sus bases en los algoritmos greedy (donde el camino escogido no es siempre el más corto) y algoritmos hill climbing (donde su objetivo es encontrar un óptimo local que no puede ser mejorada considerando la una configuración de la vecindad). Entonces, el algoritmo A star recoge todas las circunstancias anteriores y propone un método basado en una heurística.

Explicación de la heurística

A continuación se muestra la función de evaluación del algoritmo A*:

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

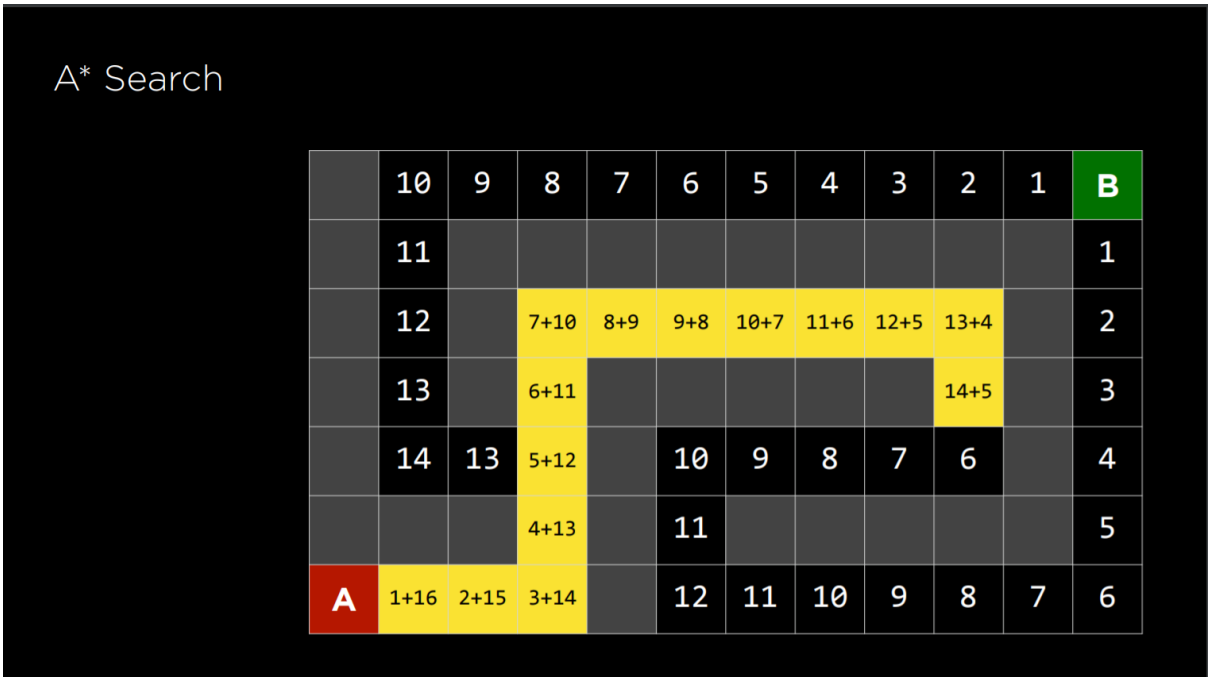
donde $h'(n)$ representa el valor heurístico del nodo a evaluar desde el actual,n hasta el final y $g(n)$es el coste real del camino recorrido para llegar a dicho nodo n desde el nodo inicial.

Además, A* mantiene dos estructuras de datos auxiliares:

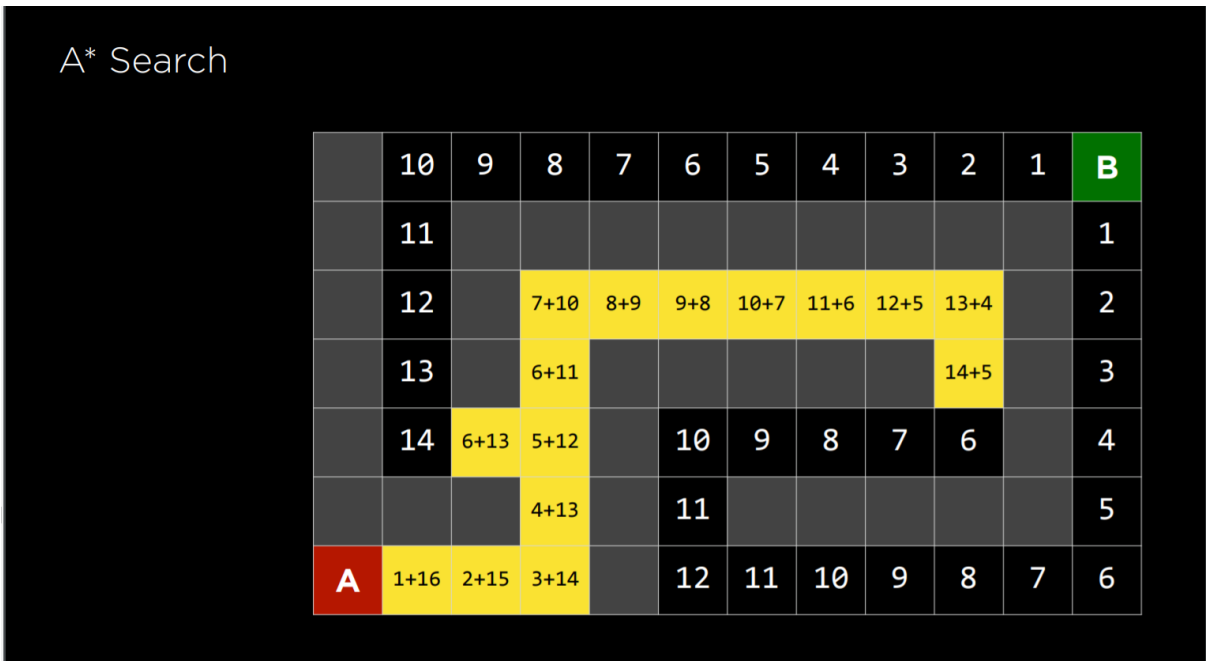
- lista abierta: una cola de prioridad ordenada por el valor de $f(n)$
- lista cerrada: guarda la información de los nodos que ya han sido visitados

El algoritmo consiste en procesar el primer nodo de la lista abierta, si este no cumple con ser objetivo, se calcula $f(n)$de todos los hijos y se almacena en la lista abierta, pasando este último a la lista cerrada.

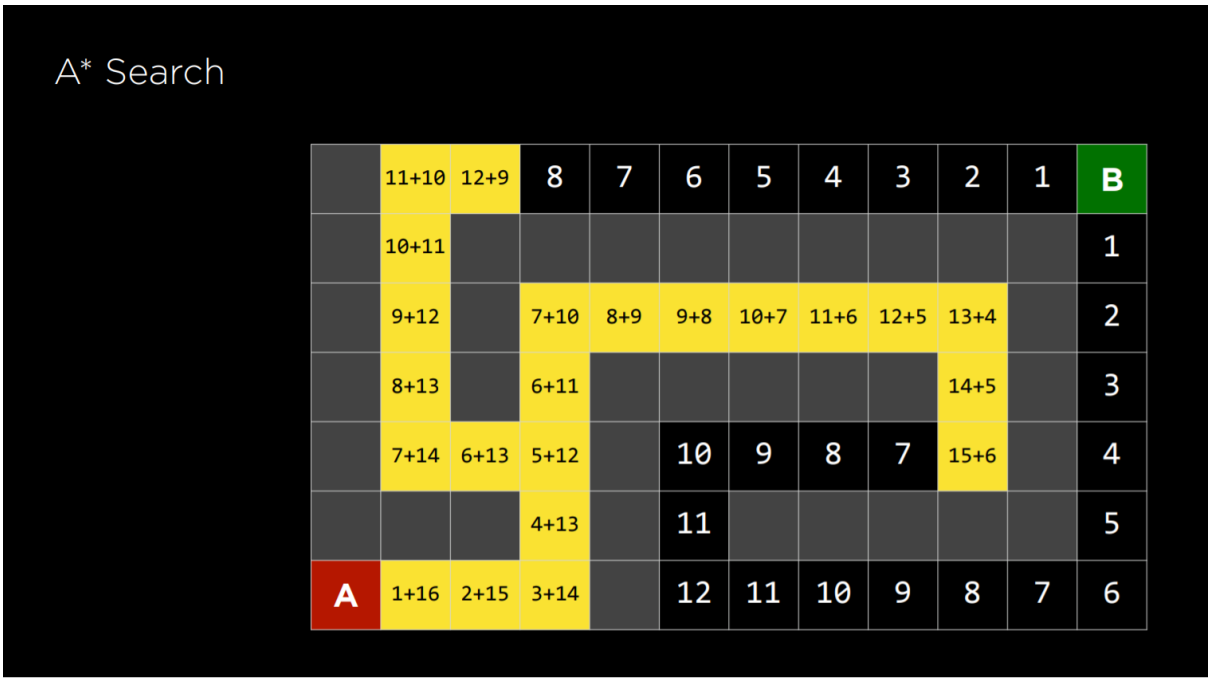
A continuación se muestra un ejemplo ejecutando el algoritmo usando las distancias Manhattan.



El algoritmo busca el camino más corto entre los nodos a los que puede acceder.

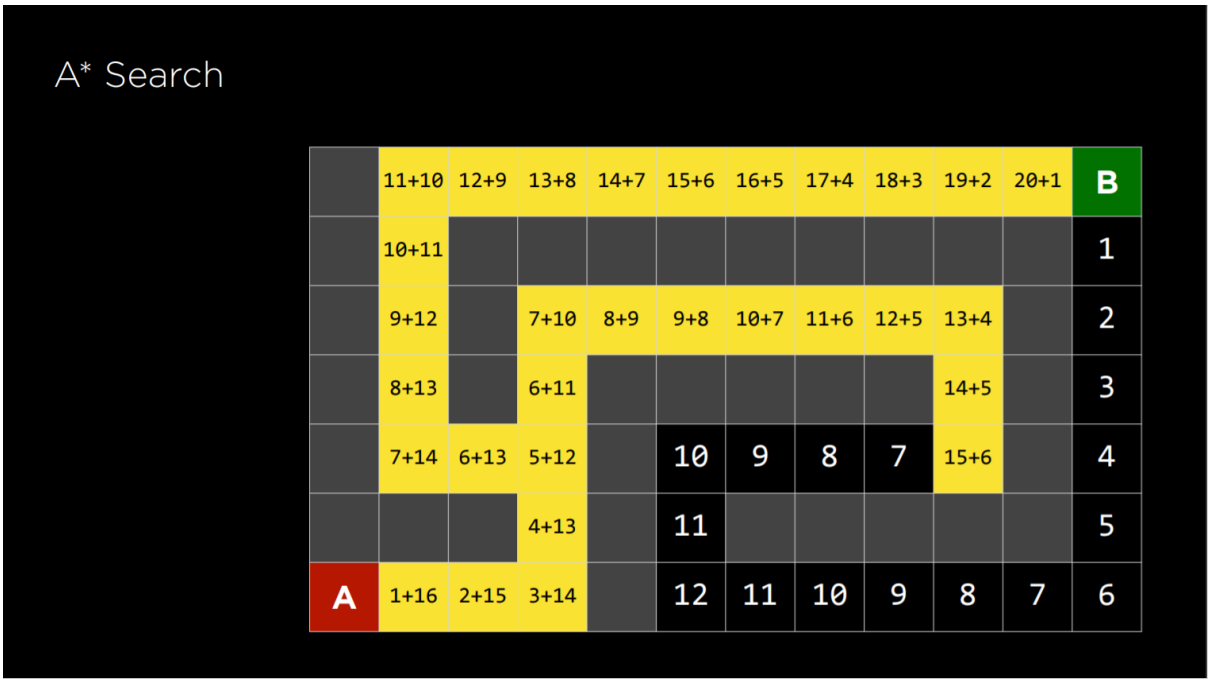


En este caso el algoritmo cambia de ruta para resolver este laberinto ya que g(n) + h(n) es menor en el camino de la izquierda (6 + 13 izquierda < 15 + 6 derecha).



El algoritmo continúa avanzando, este siempre se dirige al nodo que tenga el menor valor

tras usar la ecuación del algoritmo

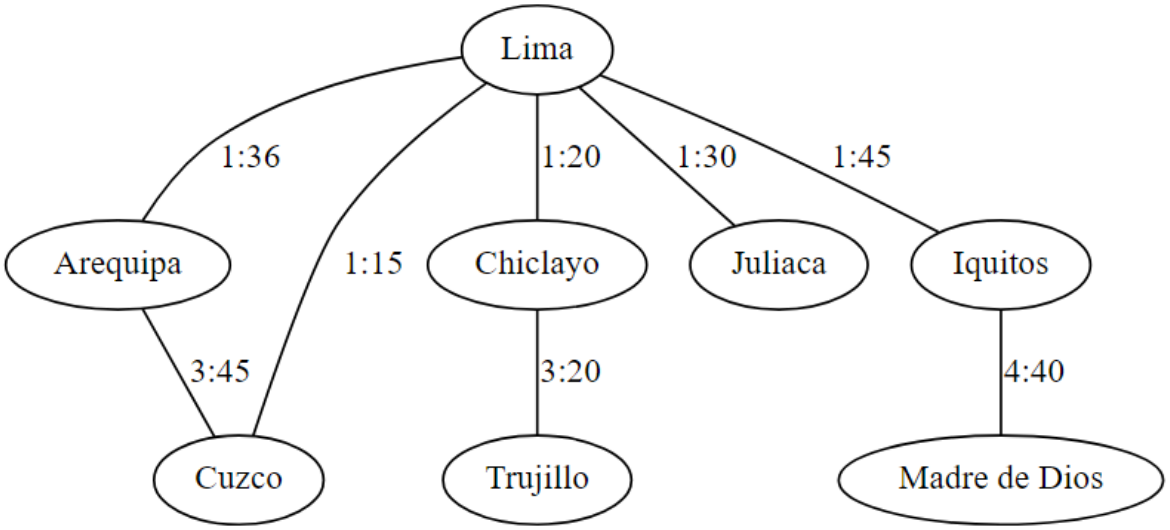


Finalmente el algoritmo llega al final del laberinto, aunque como se muestra en la imagen este puede tomar rutas equivocadas pero se corrige a sí mismo.

Por otro lado, nosotros hemos representado nuestra heurística como una matriz donde la fila representa el origen y la columna el destino. El valor es la distancia en kilómetros desde el origen hacia el destino, estas distancias fueron recuperadas de https://es.distance.to/1.

	0	1	2	3	4	5	6	7
0	0.00	762.64	573.18	666.37	490.91	836.71	1017.34	739.15
1	762.64	0.00	323.18	1400.86	1226.88	180.75	1419.07	508.36
2	573.18	323.18	0.00	1142.49	975.60	294.88	1095.91	251.69
3	666.37	1400.86	1142.49	0.00	176.05	1435.16	802.98	1201.75
4	490.91	1226.88	975.60	176.05	0.00	1266.36	801.38	1051.48
5	836.71	180.75	294.88	1435.16	1266.36	0.00	1349.64	388.46
6	1017.34	1419.07	1095.91	802.98	901.38	1349.64	0.00	975.19
7	739.15	508.36	251.69	1201.75	1051.48	388.46	975.19	0.00

Se visualiza el grafo no dirigido que hemos hecho con las 8 diferentes ciudades por la cual puedes viajar al lugar que quieras y encontrara la manera con menor distancia para llegar a tu destino.



Luego se usan cinco estructuras para el almacenamiento de los siguientes componentes:

- open_ls: lista ordenada de tuplas, donde contienen (F, nodo) sin procesar.
- closed_ls: lista de tupla (F, nodo) procesados
- parent: lista de padres de cada elemento, donde el padre indica la procedencia del nodo.
- G: lista donde cada elemento representa la distancia recorrida del nodo inicio a ‘n’.
- F: lista donde cada elemento representa, la siguiente ecuación

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

A continuación se mostrará cómo se representan las listas anteriores:

open_ls	(8, 0)						
closed_ls							

En este caso, se representa el estado inicial donde se ingresa el nodo 0, entonces la tupla representa (F(0), 0).

$$f(0) = g(0) + h'(0)$$
$$f(0) = 0 + 8$$
$$f(0) = 8$$

Por otro lado, para representar las listas restantes se usa los índices como nodos, entonces, F(0) será el valor de F en el nodo 0.

	0	1	2	3	4	5
G(i)	0	3	5	5		
H(i)	8	7	3	4		
F(i)	8	10	8	9		
parent(i)	-1	0	1	0		

Adaptación de la problemática

Para este problema lo que usamos es el algoritmo A star o A*, no hemos tenido que hacer una gran modificación al código base del algoritmo para poder trabajarlo. Nosotros comenzamos trabajando con un grafo no dirigido con el cual nos facilita para que pueda viajar por las 8 diferentes ciudades algunas conectadas con diferentes distancias que investigamos y colocamos los cuales identificamos con un ID además también colocamos el tiempo en que tomaría en llegar a su destino, este grafo se manda como un parámetro, asimismo hicimos una tabla para a heurística porque para trabajar con A* lo necesitamos, aparte también le mandamos el nodo inicial que es de donde queremos partir y el nodo final al cual queremos llegar.

Hemos hecho que el algoritmo nos devuelva dos listas una abierta y otra cerrada, estas listas guardan, cuentan y muestran los ID de las ciudades por las que vamos a pasar. Luego se usarán esas ID para crear una tabla donde se mostrará una tabla donde se muestra las ciudades por las que viajaremos, además de cuanta distancia recorreremos y la distancia total adicionalmente también cuenta el tiempo.

Finalmente hicimos una función resultado donde se ordena y se puede visualizar mejor la respuesta del trayecto que hace para poder llegar a su destino, muestra entre qué ciudades tuvo que viajar para poder llegar a su destino, la distancia que recorrió por cada ciudad y el total de las distancias recorridas, también enseña el tiempo entre las ciudades por las que pasó y el tiempo total que tomaría llegar al destino deseado.

Detalles del código fuente de la aplicación

Para la implementación del código se realizó en Google Collaboratory el cual permite ejecutar y programar en Python en el navegador. Además, nos permitió elaborar código en equipo. Luego, se usaron las siguientes librerías para implementar el algoritmo.

- **graphviz**
  Graphviz es un software de visualización de gráficos de código abierto. La visualización de gráficos es un método para expresar información estructurada como gráficos abstractos y gráficos de red. Tiene importantes aplicaciones en la interfaz visual de redes, bioinformática, ingeniería de software, diseño de bases de datos y web, aprendizaje automático y otros campos técnicos.

- **numpy**
  NumPy es el paquete básico de computación científica de Python. Es una biblioteca de Python que proporciona un objeto de matriz multidimensional, varios objetos derivados (como matrices y matrices enmascaradas) y varias rutinas para operaciones rápidas en matrices, incluidas matemáticas, lógica, operaciones de forma, clasificación, selección, E / S, transformada discreta de Fourier, álgebra lineal básica, operaciones estadísticas básicas, simulación aleatoria, etc.

- **pandas**
  Pandas es una biblioteca de código abierto que permite realizar manipulación y análisis de datos en Python. La biblioteca Pandas Python proporciona manipulación de datos y manipulación de datos para tablas numéricas y series de tiempo. Pandas proporciona una forma sencilla de crear, manipular y organizar datos

Pruebas de uso, ejecución y descripción de las funcionalidades

En la implementación del algoritmo de A star, se recibe como parámetro el *grafo, source* y *target.* El primero representa el grafo en sí, el segundo representa en nodo origen y el tercero, el nodo destino.

```
1  def aStar(grafo, source, target):
2      open_ls = []                  # lista abierta con elementos tupla (F, nodo)
3      closed_ls = []                # lista cerrada
4
5      parent = [-1] * len(grafo)    # Reconstruir
6
7      G = [np.inf] * len(grafo)     # G(n) distancia recorrida del inicio al nodo 'n'
8      F = [np.inf] * len(grafo)     # H(n) distancia del nodo 'n' hasta el final
9                                    # Luego, F(n) = G(n) + H(n)
```

Luego, siguiendo con el código, se realiza un *loop* para encontrar el nodo *target.*

```
19
20  while len(open_ls):
21
22    #Sacar de lista abierta y poner en lista cerrada
23    open_ls = sorted(open_ls)   # Para sacar el menor
24    _, nodo = open_ls.pop()
25    closed_ls.append((_, nodo))
26
27    if nodo == target:    # ¿Se encontró el destino?
28      break
29
30    if nodo in closed_ls:  # ¿Esta en lista cerrada?
31      continue
32
33
34    for v, w in grafo[nodo]:
35      if v in closed_ls:  # ¿Esta en lista cerrada?
36        continue
37
38      Gaux = G[nodo] + w
39      Faux = Gaux + H.loc[v, target]
40
41      if Gaux < G[v]:
42        G[v] = Gaux
43        F[v] = Faux
44
45        tupla = (F[v], v)
46        open_ls.append(tupla)
47        parent[v] = nodo
48
```

Luego, se implementó el siguiente código para reconstruir el camino, para ello se usa la lista *parent.*.

```
48
49    camino = []
50
51    while nodo != -1:
52      camino.append(nodo)
53      nodo = parent[nodo]
54
55    return camino
```

Finalmente, se agregó un código para la visualización del camino, los tiempos y las distancias.

```python
def Resultado(camino):

  distanciaT = 0
  tiempoT = 0
  resultado = []

  for i in range(len(camino) - 1):

    origen = camino[i]
    destino = camino[i+1]

    distancia = H.loc[origen, destino]

    tiempo = 0
    for v, w in grafo[origen]:
      if v == destino:
        tiempo = w
        break

    distanciaT += distancia
    tiempoT += tiempo

    viaje = [ciudades[origen], ciudades[destino], distancia, convertMinutes2Hours(tiempo)]
    resultado.append(viaje)

  tiempoT = convertMinutes2Hours(tiempoT)
  resultado.append(["TOTAL", "-", distanciaT, tiempoT])
  return pd.DataFrame(data = resultado, columns = ['Desde', 'Hacia', 'Distancia (Km.)', 'Tiempo (H.)'])

```

Y se ve el ejemplo en la tabla que viaja de Trujillo hasta Madre de Dios contando con la distancia y tiempo parcial y total recorridos.

	Desde	Hacia	Distancia (Km.)	Tiempo (H.)
0	Trujillo	Chiclayo	176.05	3:20
1	Chiclayo	Lima	666.37	1:20
2	Lima	Iquitos	1017.34	1:45
3	Iquitos	Madre de Dios	975.19	4:40
4	TOTAL	-	2834.95	11:5

Referencias bibliográficas

- Cunquero R. (s.f.) Algoritmos Heurísticos en Optimización Combinatoria. Recuperado de:http://yalma.fime.uanl.mx/~roger/work/teaching/class_tso/docs/1-Introduction/Intro-by-Rafa%20Marti.pdf [consulta: 6 de septiembre de 2021].
- Mota S. (2015) ¿Que es un algoritmo? Una respuesta desde la obra de Wittgenstein. Recuperado de: http://e-spacio.uned.es/fez/eserv/bibliuned:Endoxa-2015-36-7140/Que_es_un_algoritmo.pdf [consulta: 6 de septiembre de 2021].

- Sigeral B. (2017) Food Delivery System with the Utilization of Vehicle Using Geographical Information System (GIS) and A Star Algorithm. Recuperado de: https://iopscience.iop.org/article/10.1088/1742-6596/801/1/012038/pdf [consultado el 6 de septiembre de 2021].