

UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS

Escuela de Ingeniería

# Trabajo Final

Docente: Luis Canaval

Curso: Complejidad Algorítmica

Integrantes: Luis Kcomt Lam U201614074

Ricardo Guevara U201618078

**Introducción**

The travelling salesman (TSP) o el problema del vendedor viajero, es un notorio problema matemático que dice lo siguiente: Dado un conjunto de ciudades y la distancia entre cada par de ciudades, el problema es encontrar la ruta más corta posible que visite cada ciudad exactamente una vez y regrese al punto de partida. Los costos de viaje son simétricos en el sentido de que viajar de la ciudad X a la ciudad Y cuesta tanto como viajar de Y a X; La "forma de visitar todas las ciudades" es simplemente el orden en que se visitan las ciudades. Para decirlo de otra manera, los datos consisten en pesos enteros asignados a los bordes de un gráfico completo finito; el objetivo es encontrar un ciclo hamiltoniano (es decir, un ciclo que pase por todos los vértices) del peso total mínimo. En este contexto, los ciclos hamiltonianos son comúnmente llamados tours.

Los problemas relacionados al TSP surgieron en el siglo XIX por un matemático irlandés Sir William Rowan Hamilton, y un matemático británico Thomas Penyngton KirKman.

El número de instancias de ciudades que se ha podido lograr para él ha ido incrementado a través de los tiempos, debido especialmente a avances en el mundo de la tecnología y los procesadores. El primer número significativo fue logrado en 1954 por R. Fulkerson, que llego a solucionar el problema con 49 instancias. El mayor número de instancias que se ha logrado fue en 2004 por el matematico R. Bixby que logro un numero de 24 978 ciudades.

Debido a la dificultad de este problema, y su contexto con los objetivos del curso, se nos ha encargado implementar este algoritmo usando técnicas aprendidas en clase. A continuación, se le presenta lo desarrollado.

**Objetivos:**

* Implementar un algoritmo por integrante del grupo usando las técnicas de UFDS, MST(Kruskal o Prim), Programacion Dinamica, Bellman-Ford, Floyd-Warshall o Johnson en el lenguaje de programación preferido.
* Analizar y hallar la complejidad algorítmica de cada algoritmo desarrollado para después poder comparar.
* Hallar una posible solución al problema del Vendedor Viajero o Traveller Salesman (TSP), el cual hasta ahora no cuenta con una solución totalmente correcta.
* Practicar y aprender más acerca de los algoritmos de búsqueda de camino más corto.

**Marco Teórico:**

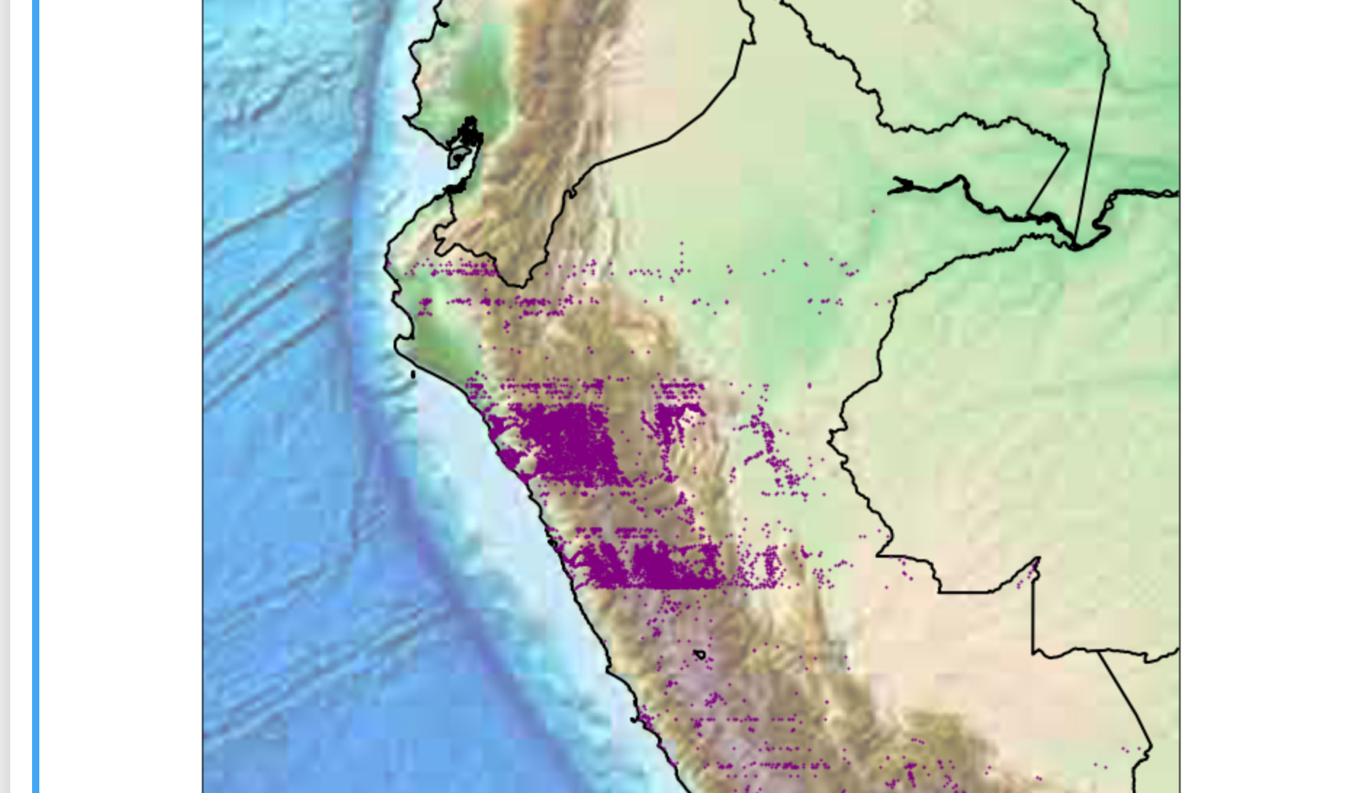
Un algoritmo es un grupo finito de operaciones organizadas de manera lógica y ordenada que permite solucionar un determinado problema. Se trata de una serie de instrucciones o reglas establecidas que, por medio de una sucesión de pasos, permiten arribar a un resultado o solución. Con esto, se han analizado dos alternativas para la solucion mas proxima al Traveller Salesman Problem.

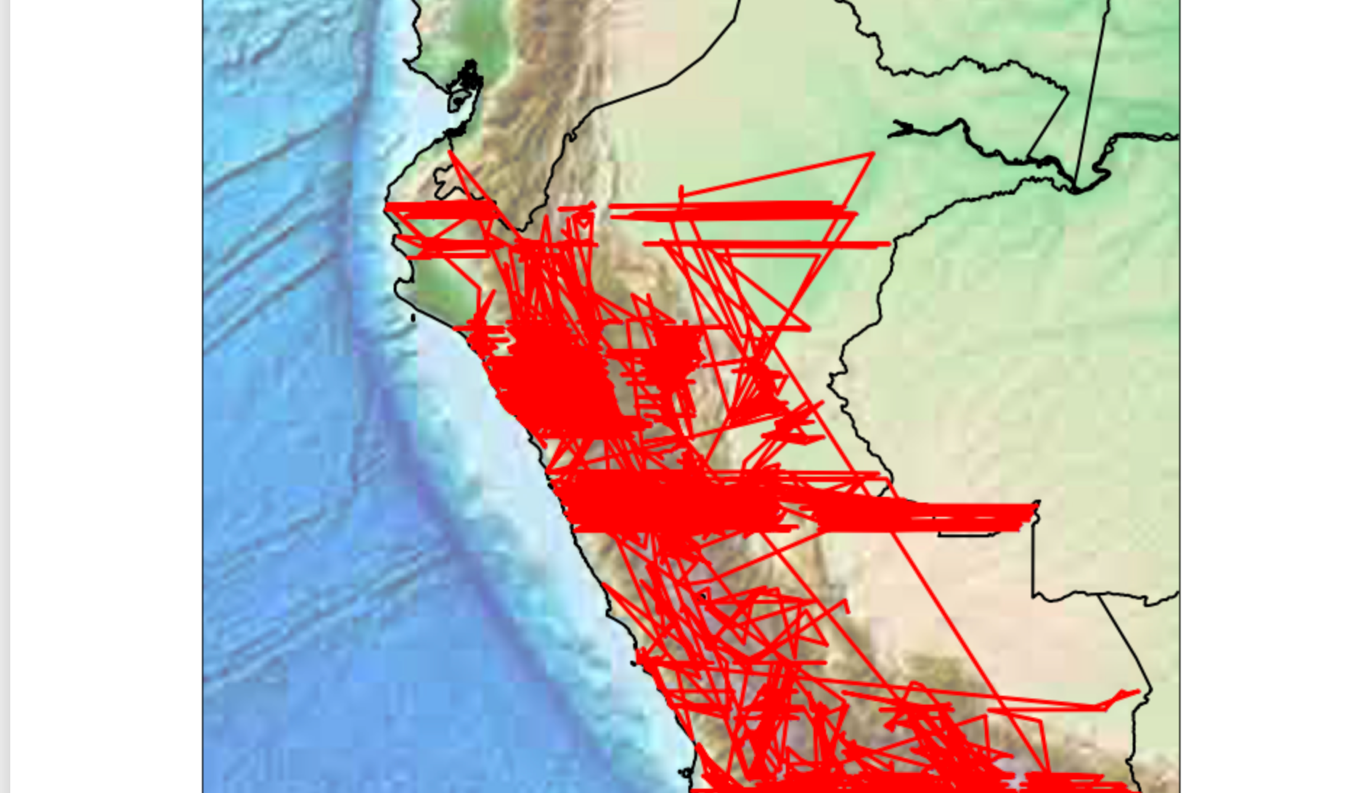
* **Kruskal:** El **algoritmo de Kruskal** es un [algoritmo](https://es.wikipedia.org/wiki/Algoritmo" \o "Algoritmo) de la [teoría de grafos](https://es.wikipedia.org/wiki/Teor%C3%ADa_de_grafos)para encontrar un [árbol recubridor mínimo](https://es.wikipedia.org/wiki/%C3%81rbol_recubridor_m%C3%ADnimo) en un grafo conexo y ponderado. Es decir, busca un subconjunto de aristas que, formando un árbol, incluyen todos los vértices y donde el valor de la suma de todas las aristas del árbol es el mínimo. Si el grafo no es conexo, entonces busca un bosque expandido mínimo (un *árbol expandido mínimo* para cada [componente conexa](https://es.wikipedia.org/w/index.php?title=Componente_conexa_(teor%C3%ADa_de_los_grafos)&action=edit&redlink=1)). Este algoritmo toma su nombre de [Joseph Kruskal](https://es.wikipedia.org/wiki/Joseph_Kruskal), quien lo publicó por primera vez en 1956. Otros algoritmos que sirven para hallar el **árbol de expansión mínima** o **árbol recubridor mínimo** es el algoritmo de Prim, el algoritmo del borrador inverso y el algoritmo de Boruvka.
* **Fuerza Bruta:** El **algoritmo de Prim** es un [algoritmo](https://es.wikipedia.org/wiki/Algoritmo" \o "Algoritmo) perteneciente a la [teoría de los grafos](https://es.wikipedia.org/wiki/Teor%C3%ADa_de_los_grafos" \o "Teoría de los grafos) para encontrar un [árbol recubridor mínimo](https://es.wikipedia.org/wiki/%C3%81rbol_recubridor_m%C3%ADnimo) en un [grafo](https://es.wikipedia.org/wiki/Grafo" \o "Grafo) [conexo](https://es.wikipedia.org/wiki/Teor%C3%ADa_de_los_grafos" \l "Grafos_conexos" \o "Teoría de los grafos), **no**dirigido y cuyas [aristas](https://es.wikipedia.org/wiki/Arista_(Teor%C3%ADa_de_grafos)" \o "Arista (Teoría de grafos)) están etiquetadas.

En otras palabras, el [algoritmo](https://es.wikipedia.org/wiki/Algoritmo" \o "Algoritmo)encuentra un subconjunto de [aristas](https://es.wikipedia.org/wiki/Arista_(Teor%C3%ADa_de_grafos)" \o "Arista (Teoría de grafos)) que forman un [árbol](https://es.wikipedia.org/wiki/%C3%81rbol_(teor%C3%ADa_de_grafos)) con todos los [vértices](https://es.wikipedia.org/wiki/V%C3%A9rtice_(Teor%C3%ADa_de_grafos)" \o "Vértice (Teoría de grafos)), donde el peso total de todas las [aristas](https://es.wikipedia.org/wiki/Arista_(Teor%C3%ADa_de_grafos)" \o "Arista (Teoría de grafos)) en el árbol es el mínimo posible. Si el grafo no es conexo, entonces el algoritmo encontrará el [árbol recubridor mínimo](https://es.wikipedia.org/wiki/%C3%81rbol_recubridor_m%C3%ADnimo) para uno de los componentes conexos que forman dicho grafo no conexo.

El [algoritmo](https://es.wikipedia.org/wiki/Algoritmo" \o "Algoritmo) fue diseñado en 1930 por el matemático [Vojtech Jarnik](https://es.wikipedia.org/wiki/Vojtech_Jarnik" \o "Vojtech Jarnik) y luego de manera independiente por el científico computacional [Robert C. Prim](https://es.wikipedia.org/wiki/Robert_C._Prim) en 1957 y redescubierto por [Dijkstra](https://es.wikipedia.org/wiki/Dijkstra) en 1959. Por esta razón, el algoritmo es también conocido como **algoritmo DJP** o **algoritmo de Jarnik**.

**Resultados de la implementacion:**

****

****

**Pruebas Unitarias:**

****

**Def test1():**

**pD, pla, c = readCPxDto(‘test2000’)**

**if c[0]!=’JUNIN’:**

**raise TypeError**

**return c[0]**

****

**Def tes2(x1,y1,x2,y2):**

**A = calculateDistance(1,1,1,1)**

**If A ¡= 0:**

**Raise TypeError**

**return A**

**Herramientas:**

**Matplotlib:** es una [biblioteca](https://es.wikipedia.org/wiki/Biblioteca_(programaci%C3%B3n)" \o "Biblioteca (programación)) para la generación de gráficos a partir de datos contenidos en listas o arrays en el lenguaje de programación [Python](https://es.wikipedia.org/wiki/Python" \o "Python) y su extensión matemática [NumPy](https://es.wikipedia.org/wiki/NumPy" \o "NumPy). Proporciona una API, **pylab**, diseñada para recordar a la de [MATLAB](https://es.wikipedia.org/wiki/MATLAB" \o "MATLAB).

**Basemap**: es una excelente herramienta para crear mapas usando python de una manera sencilla. Es una extensión de matplotlib, por lo que tiene todas sus características para crear visualizaciones de datos, y agrega las proyecciones geográficas y algunos conjuntos de datos para poder trazar líneas costeras, países, etc. directamente desde la biblioteca.

**Conclusiones:**

* Cada integrante del grupo logró implementar su objetivo, utilizando las técnicas especificadas en el lenguaje Python
* Se realizó el análisis de complejidad para cada algoritmo, de tal manera que logramos comparar cual era el más eficiente para solucionar el problema de TSP.
* Se logró solucionar el problema del Vendedor Viajero de manera parcial, debido a que no se puede ejecutar con todo el dataset, sino con solo una parte de este.
* Se practicó y se aprendieron nuevos algoritmos de búsqueda, así como también se profundizó aún más en el lenguaje Python.