- **Universidad de Guadalajara**
- **CUCEI**
- **División de Tecnologías para la Integración Ciber-Humana**
- **Departamento de Ciencias Computacionales**
- **Tema:** Algoritmo de Prim Kruskal
- **Integrantes del equipo:**
- Nicolás Alejandro Garín Gutiérrez
- Gerardo Dagnino Zatarain
- **Materia:** Análisis de algoritmos
- **Sección:** D01
- **Calendario:** 2025-B
- **Profesor:** Jorge Ernesto López Arce Delgado
- **Fecha:** 20/11/2025

Introducción

Los algoritmos de Prim y Kruskal son métodos clásicos para encontrar el árbol de expansión mínima (MST) en grafos ponderados y no dirigidos. Ambos buscan conectar todos los vértices con el menor costo posible, pero lo hacen de formas distintas.

¿Qué es un árbol de expansión mínima?

Un *árbol de expansión mínima* conecta todos los vértices de un grafo sin formar ciclos y con el menor peso total posible. Es útil en redes de computadoras, diseño de circuitos, planificación de rutas, entre otros.

Algoritmo de Prim

- **Inicio**: Parte de un vértice arbitrario.
- **Proceso**: En cada paso, agrega el borde de menor peso que conecta un vértice dentro del árbol con uno fuera.
- **Estrategia**: Expansión progresiva desde un nodo.
- **Ventaja**: Eficiente en grafos densos (muchos bordes).
- **Estructura usada**: Cola de prioridad (heap) para seleccionar el borde mínimo.

Algoritmo de Kruskal

- **Inicio**: Ordena todos los bordes por peso.
- **Proceso**: Agrega el borde más ligero que no forme ciclos, usando estructuras de conjuntos disjuntos (Union-Find).
- **Estrategia**: Construcción desde los bordes más livianos.
- **Ventaja**: Eficiente en grafos dispersos (pocos bordes).
- **Estructura usada**: Lista de bordes y estructura Union-Find para detectar ciclos.

Comparación

Característica	Prim	Kruskal
Punto de partida	Vértice	Lista de bordes
Selección	Borde mínimo desde el árbol	Borde mínimo global
Prevención de ciclos	Implícita	Union-Find
Ideal para	Grafos densos	Grafos dispersos
Complejidad	O(E log V) E = número de aristas V = número de vértices	O(E log V) E = número de aristas V = número de vértices

Aplicaciones comunes

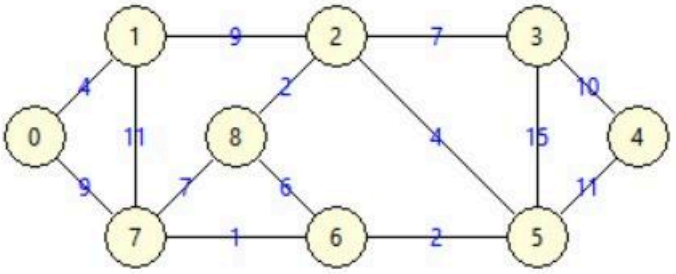
- Diseño de redes (eléctricas, de datos)
- Optimización de rutas
- Agrupamiento jerárquico en machine learning

Objetivos

El objetivo de esta actividad, es programar los algoritmos voraces de Prim y Kruskal en python, para poder observar y entender de mejor manera su funcionamiento.

Desarrollo

Antes que nada era necesario trabajar sobre el mismo grafo para los dos programas, para hacer una justa comparación entre los dos algoritmos. Se decidió en utilizar el siguiente grafo:



- **Prim**:

  En principio, se revisó cómo funciona el algoritmo de Prim, y qué lo distingue de Kruskal, especialmente el hecho de que Prim si parte de un nodo inicial y va construyendo el árbol agregando siempre la arista más ligera disponible. Después de entender ese proceso, se consultaron algunas referencias para ver distintas formas de implementarlo, solo como guía.

  Con esa base, se decidió usar una cola de prioridad, ya que Prim necesita elegir constantemente la arista más barata entre todas las que conectan al árbol con un nodo nuevo. También fue necesario llevar un control de qué nodos ya habían sido incluidos, para evitar volver a visitarlo o generar ciclos.

  El desarrollo se hizo de forma incremental: se inicia marcando el nodo seleccionado por el usuario, se agregan sus aristas a la estructura de prioridad y, cada vez que se toma una arista, se verifica que conduzca a un nodo todavía no incorporado. Si cumple esto, se añade al árbol y se insertan las aristas del nuevo nodo para seguir creciendo.

  Este ciclo continúa hasta conectar todos los nodos o hasta que ya no haya aristas disponibles. Al final se muestran las aristas elegidas, el peso de estas, y el peso total del árbol resultante.

- **Kruskal**:

  Primero se investigó la manera en que funciona, después se buscaron códigos para poder basarnos en esos. Debido a eso, se creó una clase con tres funciones principales. Debido a que kruskal, empieza con el grafo sin aristas (conjuntos disjuntos), osea cada nodo por separado y se va agregando las aristas de menor costo.

Es necesario tener una función que busca el nodo padre de un nodo (dentro del árbol de expansión mínima), con eso se pueden hacer validaciones necesarias en otra función que es, unión, se usa para unir dos nodos con las arista más barata, la validación que se ocupa hacer es que primero no se esté uniendo al mismo nodo y después que no se forme un ciclo validando que no tengan el mismo nodo padre. Unión se repite hasta que la lista de aristas se acabe.

Por último la función que es el algoritmo de kruskal, que ejecuta el algoritmo en sí, separa cada nodo como su propio conjunto y padre (empezar sin aristas, conjuntos disjuntos), obtiene la lista de aristas con su peso de manera descendente y arma el árbol de expansión mínima usando las funciones antes descritas.

Conclusión

Se podría decir que Prim es más fácil de entender, y por lo tanto implementar, al menos con la ayuda de heapq, porque el Union-Find y el tema de los conjuntos disjuntos, adicionalmente Prim es más lineal, algo tipo: visitar nodo - agrega aristas - elige la más barata. Y listo, no requiere ordenar nada desde el inicio, solo manejar el heap, y ser más feliz.

Referencias

***Algoritmo de Kruskal en Python – Guía completa, ejemplos y mejores prácticas*. (2025, noviembre 15). Asimov Ingeniería.** https://asimov.cloud/blog/programacion-5/algoritmo-de-kruskal-teoria-implementacion-en-python-y-casos-practicos-677

***Comparativa de los Algoritmos de Prim y Kruskal en Teoría de Grafos*. (s. f.).** https://wolfdisenoweb.com/2025/04/19/algoritmo-de-prim-y-kruskal/

***Kruskal's Algorithm in Python*. (2025, julio 23). GeeksforGeeks.** https://www.geeksforgeeks.org/python/kruskals-algorithm-in-python/