Dijkstra's algorithm

Dijkstra's algorithm helps find the shortest route in a network. Its speed depends on how we keep track of possible paths: $O(V^2)$ when checking each option one by one, $O((V + E) \log V)$ when using a smarter queue like a binary heap, and $O(V \log V + E)$ with an even faster approach (Fibonacci heap). The heap based methods make a huge difference, especially when dealing with big networks with fewer connections.

Kruskal's algorithm

Kruskal's algorithm helps find the cheapest way to connect all points in a network without creating loops. The biggest time-consuming part is sorting the edges, which takes $O(E \log E)$ time. Once sorted, it quickly connects the pieces using a smart merging trick (union-find with path compression), making the process smoother and faster in real-world cases.

Prim's algorithm

Prim's algorithm finds the minimum spanning tree (MST) of a graph, meaning it connects all points with the smallest total edge weight without forming loops. It starts from any node and grows by adding the shortest possible connection. Its time complexity depends on how we handle edge selection: $O(V^2)$ with a basic approach, $O(E \log V)$ with a binary heap, and $O(E + V \log V)$ using a Fibonacci heap. The more efficient versions make a big difference in large, sparse networks.