

Universidad de Ingeniería y Tecnología
Algoritmos y Estructuras de Datos (CS2023)



**UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA**

Tarea 4 - LimaMapGraph

Profesor:
Sanchez Enriquez, Heider Ysaias

Integrantes:

Martínez Olivos, Félix Alberto	202310685
Rivera Matta, Thiago Gabriel	202310290
Cadillo Espantoso, Jesús Sebastian	202310038

Lima - Perú
26 de Noviembre, 2025

1. Análisis de Complejidad de los Algoritmos Implementados

Sea V el número de nodos y E el número de aristas en el grafo. En las tres implementaciones se utiliza una `priority_queue` basada en un heap binario, donde cada operación `push` y `pop` tiene costo $O(\log N)$. Sin embargo, debido a que en estas implementaciones no se realiza una operación de *decrease-key*, un mismo nodo puede ser insertado múltiples veces en la cola, lo cual afecta la complejidad espacial.

Dijkstra

El algoritmo de Dijkstra es uno de los métodos más conocidos para encontrar el camino más corto en un grafo ponderado con pesos no negativos. Su complejidad temporal en el peor caso es

$$O((V + E) \log V),$$

pues cada extracción e inserción en la cola de prioridad tiene costo $\log V$, y en el peor escenario es necesario procesar todas las aristas del grafo. Dado que la implementación utilizada realiza un `push` cada vez que se encuentra una mejor distancia, la cola de prioridad puede llegar a contener hasta $O(E)$ elementos.

El algoritmo inicia asignando distancia infinita a todos los nodos, excepto al nodo origen. Luego, se inserta el origen en la cola de prioridad. Iterativamente se extrae el nodo con menor distancia acumulada y se relajan sus aristas. Si se encuentra un camino más corto hacia un vecino, se actualiza su distancia y se vuelve a insertar en la cola. El proceso continúa hasta llegar al destino o agotar los nodos disponibles.

Respecto a la complejidad espacial, es necesario almacenar las distancias mínimas conocidas, el conjunto de nodos visitados y todas las entradas que se acumulan en la cola de prioridad. Además, la implementación registra cada arista visitada para efectos de visualización. En conjunto, esto produce un uso de memoria de

$$O(V + E).$$

A*

El algoritmo A* extiende a Dijkstra incorporando una heurística que estima el costo restante hacia el destino. Para cada nodo n se evalúa

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

donde $g(n)$ es el costo real desde el origen y $h(n)$ es la heurística utilizada (en este caso, la distancia euclíadiana al nodo destino). Cuando la heurística es admisible, A* mantiene la optimalidad y reduce el número de nodos explorados en comparación con Dijkstra.

No obstante, en el peor de los casos la heurística puede no aportar información útil, haciendo que A* se comporte de manera idéntica a Dijkstra. Por ello, la complejidad temporal sigue siendo

$$O((V + E) \log V).$$

El algoritmo inicializa $g(n) = \infty$ para todos los nodos salvo el origen, y asigna a la cola de prioridad el valor $f(n)$. En cada iteración se selecciona el nodo con menor $f(n)$ y se exploran sus vecinos. Cuando se encuentra un mejor camino hacia un vecino, se actualiza su costo y se inserta nuevamente en la cola. El proceso termina cuando se extrae el nodo destino o cuando se agotan los nodos disponibles.

En cuanto a memoria, A* almacena los valores $g(n)$, el conjunto de nodos visitados, la cola de prioridad (que puede contener hasta $O(E)$ entradas debido a reinserciones múltiples) y las aristas exploradas utilizadas para visualización. Esto da una complejidad espacial de

$$O(V + E).$$

Greedy Best-First Search

El algoritmo *Greedy Best-First Search* utiliza únicamente la heurística para decidir qué nodo expandir, con función de prioridad

$$f(n) = h(n).$$

A diferencia de A*, no considera el costo acumulado, por lo que no garantiza encontrar rutas óptimas. Su eficiencia depende fuertemente de la calidad de la heurística, y puede tanto explorar muy pocos nodos como terminar recorriendo una gran parte del grafo.

En el peor escenario, la heurística no ayuda a dirigir la búsqueda, de modo que la complejidad temporal alcanza

$$O((V + E) \log V).$$

El algoritmo comienza insertando el nodo inicial en la cola de prioridad y, en cada iteración, selecciona el nodo cuya heurística estimada es mínima. Luego expande sus vecinos, insertándolos nuevamente en la cola. Al no utilizar costos acumulados, puede desviarse hacia caminos engañosamente prometedores.

La complejidad espacial incluye el almacenamiento del conjunto de nodos visitados, la estructura de padres y la cola de prioridad, la cual puede contener múltiples inserciones del mismo nodo. Además, se registran las aristas exploradas para visualización. En total, la complejidad espacial es

$$O(V + E).$$