Act-Integradora-4

Conceptos básicos y algoritmos fundamentales

Programación de estructuras de datos y algoritmos fundamentales
(Gpo 850)

Profesor
Eduardo Arturo Rodríguez Tello

Nombres:

Federico Melo B. A00833536

Analiza la importancia y eficiencia del uso grafos en una situación problema de esta naturaleza.

Usar grafos fue clave porque las IPs y sus conexiones se representan de forma natural como nodos y aristas. Con la lista de adyacencia se recorren solo las conexiones reales lo que hace más eficiente el manejo de datos grandes. Ademas es ideal para grafos dispersos como este.

Reflexiona por qué en la solución de este reto es preferible emplear un Binary Heap y no otra estructura de datos jerárquica, basa tu reflexión en la complejidad temporal de las operaciones de las estructuras de datos.

El heap nos ayudó a sacar rápido las IPs con más conexiones. Soporta inserciones y extracciones con buena velocidad (log n), y es más simple que otras estructuras como árboles balanceados. Es práctico y suficiente para lo que necesitábamos.

Analiza la complejidad computacional de las operaciones básicas de las estructura de datos (grafo y binary heap) empleadas en tu implementación y cómo esto impacta en el desempeño de tu solución.

Las operaciones del grafo (agregar nodos, aristas y grados) son constantes. Dijkstra sin heap tiene complejidad $O(n^2 + m)$. El heap hace push/pop en log n. Combinando ambos, logramos buen rendimiento incluso con muchos datos.

Reflexiona acerca del algoritmo empleado para identificar el camino más corto entre la IP identificada como el bot master y el resto de las IPs. Justifica la elección del algoritmo empleado con base en su complejidad temporal y la de otros algoritmos alternativos.

Usamos Dijkstra porque todas las conexiones tienen pesos positivos y nos interesaba la distancia mínima desde el bot master a todas las IPs. Aunque no usamos heap en Dijkstra, fue suficiente para la escala del problema.

Presenta el pseudo-código del método empleado para identificar el camino entre el bot master y la dirección IP que presumiblemente requiere mayor esfuerzo para ser atacada (punto 8), así como el análisis de su complejidad temporal.

```
reconstruirCamino(prev, destino):
    camino ← lista vacía
    actual ← destino

    mientras actual ≠ -1:
        camino.agregarAlInicio(actual)
        actual ← prev[actual]

    regresar camino
```

Este algoritmo reconstruye el camino desde la IP más lejana al bot master siguiendo los predecesores que dejó Dijkstra. Solo recorre una vez cada nodo del camino, así que su complejidad es O(n) en el peor caso, si el camino toca todos los nodos.