Reflexión Actividad Integradora 4

Gonzalo Flores García - A01644771

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

Debido a que esta tarea se enfoca en encontrar movimientos sospechosos para encontrar a los bots, es importante usar grafos para poder modelar las conexiones entre direcciones IP. Debido a la enorme cantidad de elementos, no es nada óptimo analizar los patrones de comportamiento en búsqueda de anomalías sin los algoritmos eficientes de los grafos.

Ventajas del Binary Heap frente a otras estructuras jerárquicas

La razón por la cuál se usó un binary heap para encontrar las IPs con mayor grado de salida es debido a que como estructura nos permite acomodar los elementos de manera que obtener el máximo, (o mínimo) se pueda realizar en un tiempo constante. También, poblarlo con inserciones se puede lograr con una complejidad de O(log n). Esto es a diferencia de un BST que no necesariamente estaría balanceado o un AVL que necesitaría dicho balanceo dándonos una mayor complejidad en la implementación. El heap es excelente para esto ya que nos permite conseguir las IPs más sospechosas sin tener que recorrer todos los nodos.

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

Grafo: Primero que nada nuestra inserción tiene una complejidad que en el peor de los casos si se inserta hasta el final sería de O(n). También la parte de insertar los registros en orden cronológico para cada IP nos deja con un peor caso de O(m), siendo el número de IPs iguales ya insertadas.

Binary Heap: La inserción en el heap nos da una complejidad de O(log n), por otra parte la extracción de las IP más sospechosas también nos da una complejidad de O(log n).
Esto quiere decir que para poblar completamente nuestro heap tenemos una complejidad de O(n log n)

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.

Para conseguir los caminos más cortos nosotros decidimos usar el algoritmo BFS, (Breadth-First Search). Este lo escogimos más que nada basándonos en nuestro grafo. Debido a que nuestras aristas realmente no tenían diferencia de pesos, solo necesitábamos conseguir el mínimo número de saltos para llegar de un nodo a otro, haciendo que otras alternativas famosas como el algoritmo Dijkstra que suele ser más versátil resultaba más costoso debido a su tiempo de O(N+C log N) en comparación con el del BFS que es de O(N + C). También el hecho de que no necesita de estructuras avanzadas como AVL nos da una implementación bastante sencilla.

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.

```
función identificarCaminoMasLargo(listaAdyacencias):
    heap = BinaryHeapMax() // basado en grado de salida
    para cada nodo en listaAdyacencias:
        heap.insertar(nodo)

    botMaster = heap.extraerMaximo()
    ipBotMaster = botMaster.IP

   // Ejecutar BFS desde el bot master
    distancias = mapa IP → int
    padres = mapa IP → IP
    // marcar ipBotMaster como visitado
    distancias[ipBotMaster] = 0
    cola.enqueue(ipBotMaster)

    mientras cola no esté vacía:
        ipActual = cola.dequeue()
        para cada vecino conectado a ipActual:
            si no visitado:
                distancias[vecino] = distancias[ipActual] + 1
                padres[vecino] = ipActual
```

```
            cola.enqueue(vecino)

    // Buscar la IP más lejana desde el bot master
    para cada ip en distancias:
        si distancias[ip] > maxDistancia:
            maxDistancia = distancias[ip]
            ipMasLejana = ip
```

Análisis de Complejidad:

Construir el heap: O(n log n)
Extraer el bot master: O(log n)
BFS: O(N + C) Nodos y conexiones
IP más lejana: O(N) Checar todas las IPs

Bibliografía

GeeksforGeeks. (2023). Graph and its representations. Recuperado de https://www.geeksforgeeks.org/graph-and-its-representations/

TutorialsPoint. (n.d.). Graph Data Structure. Recuperado de https://www.tutorialspoint.com/data_structures_algorithms/graph_data_structure.htm

GeeksforGeeks. (2023). Binary Heap. Recuperado de https://www.geeksforgeeks.org/binary-heap/

Programiz. (n.d.). Breadth First Search (BFS) Algorithm. Recuperado de https://www.programiz.com/dsa/graph-bfs

GeeksforGeeks. (2023). Difference between Dijkstra's and BFS. Recuperado de https://www.geeksforgeeks.org/difference-between-dijkstras-and-bfs/

Brilliant.org. (n.d.). Breadth-First Search. Recuperado de https://brilliant.org/wiki/breadth-first-search-bfs/