Reflexión Individual – Act-Integradora-4

Nombre: Arturo Gómez
Matrícula: A07106692

A. Importancia y eficiencia del uso de grafos en esta problemática

El uso de grafos en esta actividad es fundamental, ya que permite modelar relaciones entre direcciones IP como un conjunto de nodos conectados. Cada conexión de red en el archivo de bitácora representa una arista entre dos nodos. Esta estructura facilita operaciones como el análisis de comunicación entre nodos, detección de nodos clave (bot master) y rutas de ataque. Además, permite almacenar y consultar eficientemente información relacionada con conexiones, grados de salida, y rutas, lo cual sería mucho más costoso con estructuras lineales.

B. Preferencia por Binary Heap

El uso de un Binary Heap es preferible sobre otras estructuras jerárquicas (como árboles binarios de búsqueda o AVL) debido a su eficiencia en operaciones de inserción y extracción del valor mínimo (o máximo). En este caso, usamos el heap para encontrar las 5 IPs con mayor grado de salida. Esta operación requiere múltiples extracciones del mayor elemento, lo cual es **O(log n)** por extracción. Un árbol balanceado tendría un costo similar, pero la sobrecarga para mantenerlo balanceado no es necesaria aquí. El heap es más simple de implementar y suficientemente eficiente para esta tarea.

C. Complejidad computacional de grafos y heap

- **Grafo con lista de adyacencia:**
    - Espacio: O(V + E)
    - Agregar conexión: O(1)
    - Recorrido: O(V + E)
- **Binary Heap:**
    - Insertar: O(log n)

- Extraer máximo: O(log n)

Estas complejidades hacen que el algoritmo completo sea eficiente incluso para un gran número de IPs, permitiendo análisis rápidos y procesamiento en tiempo razonable.

D. Algoritmo del camino más corto

El algoritmo utilizado para encontrar el camino más corto fue **Dijkstra**, ya que permite encontrar distancias mínimas desde un nodo (bot master) al resto de los nodos con pesos uniformes. Dijkstra tiene una complejidad de **O((V + E) log V)** usando un min heap, lo cual lo hace muy eficiente para grafos dispersos como este. Algoritmos alternativos como Bellman-Ford no son necesarios ya que no hay pesos negativos, y BFS sería menos flexible si existieran pesos o diferentes niveles de conexión.

E. Pseudo-código y análisis temporal del camino más costoso

```
void bfsDesdeIP(ListaAdyacencias& grafo, string ipInicio,
        unordered_map<string, int>& distancias,
        unordered_map<string, string>& padres) {
  unordered_map<string, bool> visitado;
  queue<string> cola;

  cola.push(ipInicio);
  distancias[ipInicio] = 0;
  visitado[ipInicio] = true;

  while (!cola.empty()) {
    string actual = cola.front();
    cola.pop();
```

```
        Node* nodo = grafo.obtenerNodo(actual);
        if (!nodo) continue;

        for (const auto& reg : nodo->registros) {
            string vecino = reg->destino.ip;
            if (!visitado[vecino]) {
                visitado[vecino] = true;
                distancias[vecino] = distancias[actual] + 1;
                padres[vecino] = actual;
                cola.push(vecino);
            }
        }
    }
}

void guardarDistancias(const unordered_map<string, int>& distancias) {
    ofstream salida("distancia_botmaster.txt");
    for (const auto& par : distancias) {
        salida << par.first << " " << par.second << endl;
    }
    salida.close();
}

void guardarCamino(const unordered_map<string, string>& padres, const string&
ipFinal) {
    stack<string> camino;
```

```
    string actual = ipFinal;

    while (padres.find(actual) != padres.end()) {
        camino.push(actual);
        actual = padres.at(actual);
    }
    camino.push(actual);

    ofstream salida("ataque_botmaster.txt");
    while (!camino.empty()) {
        salida << camino.top() << endl;
        camino.pop();
    }
    salida.close();
}
```

Complejidad: El recorrido de distancias es O(n), y la reconstrucción del camino depende de la longitud del mismo, O(n) en el peor caso. Por tanto, esta parte del algoritmo tiene una complejidad de **O(n)** adicional a la ejecución de Dijkstra.