Tecnológico de Monterrey

Nombre: Rodrigo Castillo Francisco

Matricula: A01799191

Materia: Programación de estructuras de datos y algoritmos fundamentales

Profesor: Eduardo Arturo Rodríguez Tello

1.Importancia y eficiencia del uso de grafos en este problema

Bueno en casos así donde se dan los análisis de conexiones de red, como el mapeo de bots, un grafo es una estructura podría decirse más natural para representar las IPs como los nodos y conexiones como las aristas. Gracias a la lista de adyacencia se pudo iterar únicamente sobre los vecinos reales de cada IP, ignorando pares no conectados. Esta representación es eficiente para en espacio para grafos dispersos que en este caso en el problema fueron muchas IPs con pocas conexiones por IP), pues su complejidad temporal es de O(n+m) con n nodos y m aristas en lugar de los O(nˆ2) de una matriz de adyacencia. Por otra parte, las operaciones claves como recorrer todas las aristas sucede en tiempo O(n+m), por lo que permite escalar bitácoras con miles de registros manteniendo un rendimiento lineal en el tamaño de los datos.

2. Porque emplear un Binary Heap y no otra estructura jerárquica

Bueno para extraer todas las que son k IPs de mayor grado de salida o por ejemplo seleccionar el siguiente nodo más cercano en Dijkstra, se necesita una estructura que soporte dos operaciones de alta eficiencia por ejemplo inserción para agregar nuevos pares o extraer el máximo y mínimo para remover el elemento mayor o de menor prioridad.

Un Binary Heap ofrece opciones más rápidas por ejemplo un Push, pop, o top con complejidades de tiempo rápidos. Binary Heap es simple y tiene una eficiencia para las operaciones de prioridad.

3. Complejidad computacional de las operaciones básicas

Grafos

-addNode: O (1)

-addEdge: O (1)

Heap

-push: O (log n)

-top: O (1)

-pop: O (log n)

En getDeegree: O (1) por nodo

Para las top 5 inserciones más las 5 extracciones son O (n log n

Dijkstra tiene un total de O(nˆ2+m)

El desempeño de mayor nivel es el Dijkstra en grafos densos mientras que en grafos que están muy dispersos, la fase de grados y heap domina con O (n log n).

4. Reflexión

Bueno para esta entrega se eligió Dijkstra porque el trabajo esta basado en pesos positivos en donde se quiere conocer la distancia mínimo del bot máster a todas las IPs en una sola ejecución. Su complejidad O(nˆ2+m) sin heap externo es adecuada para n hasta para miles y miles. Puede que Dijkstra implementado con Heap propio pueda bajar un poco la complejidad temporal, pero en nuestro trabajo hacemos lo que es una versión lineal que simplifica la dependencia de otras estructuras y aun así sigue siendo viable.

5. Pseudocodigo y la complejidad de ataque

Reconstruir_camino(prev, farIdx) // este es el vector de los predecesores de Dijkstra

Entrada :

Prev[0…..n-1] // este es el vector de los predecesores de Dijkstra

farIdx   // índice de la IP de mayor esfuerzo

path <- lista vacia // O(1)

cur <- farIdx   // O(1)

while cur ≠ -1 do   // hasta llegar al origen

        path.insertFront(cur)   // insertar al frente, O(1)

        cur <-prev[cur]   // avanzar al predecesor, O (1)

end while

Salida:

return path   // lista de índices de IP en orden

El tota de la complejidad de todo es O(n)