Instituto Tecnológico y de Estudios Superiores de Monterrey

Tecnológico de Monterrey

Programación de Estructuras de Datos y Algoritmos Fundamentales

Ivan Reyez Amezcua

Act 4.3 - Actividad Integral De Grafos (Evidencia Competencia)

Santos Alejandro Arellano Olarte // A01643742
Carlos Iván Armenta Naranjo // A01643070
Arturo Ramos Martínez // A01643269
Adair Virgilio Figueroa Medina // A00572826
Leonardo Mario Alberto Guillén Soria // A00574110
Daniela Rocha Muñoz // A00573664

23 de Noviembre del 2023

extractIP	Descripción	Extrae la ip (incluyendo el puerto) y la retorna; esto lo hace por medio de los espacios que va encontrando en la línea de texto que recibe como argumento, de tal manera que cada que detecta un espacio, guarda en variables todo lo que estaba antes de este, como todas las líneas del archivo “bitacora.txt” tienen el mismo formato, la función ya sabe que parte de texto es la IP.
	Entrada	string
	Salida	string
	Precondición	Que se le pase un string, y que sea específicamente una linea del archivo “bitacora.txt”
	Postcondición	Se obtiene el string que representa la IP
	Complejidad	<sstream>, que es la que nos ayuda a extraer las partes de la línea de texto, tiene una complejidad O(n*log(n))

fanOut	Descripción	Esta función retorna el tamaño de la lista de adyacencia del vértice que se le pase como argumento.
	Entrada	int
	Salida	int
	Precondición	Que el argumento de entrada sea un entero
	Postcondición	Se obtiene el size de la lista de conexiones de ese vértice, el cual tomamos como fan-out.
	Complejidad	O(1)

maxFanOut	Descripción	
	Entrada	Número de segmentos con mayor FanOut que se desean obtener.
	Salida	Impresión en pantalla del número de segmentos con mayor Fan Out, ordenados de mayor a menor con respecto al FanOut
	Precondición	El grafo debe haber sido correctamente inicializado y definido.
	Postcondición	Nada
	Complejidad	La complejidad de este método es O(V) donde V es el número de vértices del grafo, porque hace un recorrido de la lista de adyacencia y compara su size para determinar el fan out de cada vértice y los va colocando en el arreglo que nos da al final para tener en él los que tengan un fan out más grande.

main	Descripción	El main es el programa principal del codigo, lee la bitácora del archivo de texto, extrae las direcciones IP de cada linea y las divide en segmentos.Luego, crea un grafo donde cada segmento de IP es un nodo y los segmentos están conectados. Después, calcula los segmentos con un Fan Out mayor, el número de segmentos lo determina el usuario y despliega estos segmentos con ayuda de la función maxFanOut, ademas en esta funcion a la hora de dividir la ip en segmentos

		utilizamos un arreglo que guarda los 4 segmentos de cada ip para al final recorrer este arreglo y determinar la IP cuya suma de fanOut de sus segmentos sea la mayor e indicar la IP Bootmaster
	Entrada	**Toma como entrada el archivo de texto “bitacora.txt”, y el número de segmentos con mayor fan out que desee el usuario**
	Salida	**Imprime el número de segmentos con el mayor Fan Out que el usuario desee y la dirección IP con el mayor número de conexiones de salida.**
	Precondición	**Las líneas en el archivo “bitacora.txt” tiene que tener líneas de bitácora válidas con direcciones IP, y debe estar en la misma carpeta que el programa.**
	Postcondición	**Se da información sobre el Fan-Out y la dirección IP BootMaster.**
	Complejidad	**La complejidad de esta función es O(N) donde N es el numero de lineas que tenga el archivo Bitacora.txt**

Investigacion y reflexion individual:

Investigación Y Reflexión De La Importancia Y Eficiencia Del Uso Grafos En Una Situación Problema De Esta Naturaleza.

Santos Arellano:

La visualización gráfica de las conexiones entre direcciones IP no solo proporciona una estructura, sino que también ayuda a identificar patrones, nodos críticos y evaluar de manera eficiente la actividad de la red. La capacidad de los gráficos para presentar datos en un formato visual estructurado es fundamental para acelerar y mejorar la comprensión de la topología de la red.

Esto es especialmente importante en un entorno donde se requiere una rápida atención para identificar posibles amenazas. Un método clave de análisis es el análisis del abanico, el cual muestra la cantidad de conexiones salientes que tiene cada nodo.

La eficiencia de este análisis juega un papel crucial a la hora de evaluar con precisión la actividad de la red e identificar nodos críticos, como el Boot Master. Además, esta herramienta es altamente efectiva para adaptarse a los cambios en el tamaño y complejidad de los datos, asegurando así que las capacidades de la red evolucionen adecuadamente.

Daniela Rocha:

Al trabajar y explorar el papel de los grafos en la resolución de problemas complejos, como el que se presentó y no solo en este sino que en diversas disciplinas, descubrí que se usa desde para encontrar la ruta más corta en un mapa hasta planificar proyectos. En la informática, los grafos son la clave en algoritmos eficientes para rutas y organización de datos, visualizando así interacciones en redes sociales o logísticas de entrega, facilitando decisiones informadas. También son utilizados en la planificación de proyectos, donde son clave en algoritmos para rutas y organización de datos.

Aunque los grafos son útiles en varias áreas, es importante saber manejar con cuidado la cantidad de datos y comprender los algoritmos. Profundizar en la estructura de un grafo, con nodos que representan cosas y conexiones entre ellos, es esencial. En programación, esto se implementa con estructuras como listas o matrices.

Las listas de adyacencia son idóneas para grafos dispersos, mientras que las matrices de adyacencia son más eficientes para grafos densos. En inteligencia artificial, los grafos son fundamentales para modelar conexiones y representar conocimientos. Conocer algoritmos como Dijkstra, Bellman-Ford y Floyd-Warshall amplía nuestras habilidades para resolver problemas específicos.

Es importante entender la diferencia entre grafos dirigidos y no dirigidos, así como la presencia de pesos en algunas conexiones. La comprensión de ciclos en grafos es básico, especialmente en grafos sin ciclos, importantes y presentes en estructuras como árboles, para evitar problemas de código infinito y mejorar el funcionamiento del código.

En los resultados obtenidos y presentados en este documento, es evidente que se ha cumplido con éxito todos los objetivos especificados.

Carlos Armenta:

El desarrollar esta actividad me ayudó a comprender de mejor manera qué cosa es un grafo y el cómo lo podemos llegar a aplicar en un problema de esta naturaleza, empezando por entender que los grafos son una estructura de datos en la cual cada vértice puede llegar a tener conexión con cualquier otro vértice dentro del grafo, además de que pues los grafos pueden llegar a ser dirigidos o no dirigidos y la navegación a través del grafo varía dependiendo de si es o no es dirigido.

Una de las aplicaciones de los grafos que me pareció muy interesante es que aplicaciones como Waze o Google maps utilizan grafos para poder decidir e indicar al conductor el camino o ruta más rápida entre un vértice u otro, y esto lo determinan asignando un valor a cada camino o conexión que puede depender de la distancia o la cantidad de tráfico en esa zona, por lo que mediante un recorrido de las diferentes rutas del grafo se logra indicar cual seria la mejor opción a tomar.

Otra característica de los grafos que me parece de suma importancia es que son muy versátiles en el sentido de que permiten añadir conexiones entre los diferentes vértices lo que permite representar una gran variedad de problemas a través de ellos además de que es muy fácil utilizar estructuras alternas auxiliares para poder facilitar alguna funcionalidad del grafo.

Para esta actividad en si utilizamos un grafo para añadir los segmentos de las IP sin repetir del archivo Bitácora.txt, para esto utilizamos un set como estructura auxiliar para poder almacenar los datos sin repetir en la lista, y poder llegar a indicar el número de segmentos que el usuario desee con un Fan Out o número de conexiones de salida más grande, de igual forma el sumar cada uno de los fan out de las 4 partes de las Ip para determinar aquella con un Fan Out mayor es decir la IP BootMaster.

Arturo Ramos Martínez:

Esta actividad me ayudó a entender un poco más la importancia e historia de los grafos, desde cómo surgió la teoría de grafos, las implementaciones que se le fueron dando, y la importancia que hoy día tiene en muchas ramas de las ciencias exactas, por ejemplo en los de los sistemas de software, como en redes sociales o en apps como uber.

Primeramente, el surgimiento de la teoría de grafos, me pareció interesante, y más aún lo que desencadenó que los científicos de la ciudad de Königsberg, trataran de resolver el conocido problema de los 7 puentes, el cual aunque podría parecer irrelevante, gracias a él, otros científicos se interesaron en el tema y claramente fue un gran acierto para las matemáticas y sus ramas, ya que día con día hacemos uso de sistemas, aplicaciones, redes sociales, juegos etc, que aplican la teoría de grafos.

Por otro lado, la problemática planteada me hizo darme cuenta de la capacidad que tienen los grafos de representar y modelar problemáticas y situaciones de la vida cotidiana, ya que su estructura como tal es más flexible que otras estructuras de datos, aunque esto también aumenta la complejidad al manejarlos.

En esta actividad usamos un grafo sin muchas limitaciones, un grafo (no árbol) representado en una lista de adyacencia, ya que nuestra cantidad de datos a manejar era considerable (alrededor de 16,000), y de esta manera manejamos mejor la memoria. De acuerdo a la problemática y los objetivos de la actividad, me di cuenta de la flexibilidad y dinamismo que ofrecen los grafos, en combinación con otras estructuras de datos, como lo son claramente las lista ligadas, o las matrices de adyacencia. Ya sabía que cada representación tenía sus ventajas y desventajas, pero siento que me hacía falta enfrentar una actividad como esta, para tener más presentes estas características; y es que siento que el usar una lista de adyacencia principalmente nos ahorró mucha memoria, pero nos presentó limitaciones por ejemplo al intentar evitar que se agregaran conexiones repetidas, proceso que sería mucho más fácil si se usará una matriz de adyacencia, por su acceso aleatorio.

Finalmente, considero que adquirí una mejor base de conocimientos sobre grafos, y aunque se que es un tema complejo, que tiene aplicaciones en muchas ciencias, me gustó poder ver un poco ese lado de aplicación en la computación y llevar a la práctica la mayoría de teoría que vimos, ya que me dió una idea más clara, y me ayudó a entender un poco mejor cómo funcionan mucha de la tecnología que uso a diario.

Adair Figueroa:

Se creó un código para esta tarea para abordar el desafío de analizar los registros, incluidas las conexiones IP. El programa generó un gráfico para representar estas relaciones y examinó las características del ventilador de cada nodo, identificando el más grande.

Al tratar cada IP como un nodo y registrar las conexiones como bordes, utilizamos la clase de gráfico para construir el diagrama. Pudimos proporcionar una topología de.

A través de esta actividad, descubrimos el valor de los gráficos como estructura de datos para representar relaciones complejas, como las conexiones entre IP. Como resultado, los gráficos resultantes nos ayudan a visualizar la topología de la red y resaltan los nodos activos en nuestras redes.

La identificación de puntos críticos en una red, como las conexiones IP, se vuelve crucial a la hora de buscar la IP con mayor despliegue. La conectividad general de la red se ve significativamente afectada por los nodos centrales que tienen una cantidad significativa de conexiones salientes. Estas configuraciones de nodos se conocen como puntos centrales. Además, la escalabilidad de los gráficos es beneficiosa para la eficiencia en el análisis, incluso cuando se consideran factores como el tamaño del registro y la complejidad de la red.

Los gráficos ofrecen una forma visualmente intuitiva de interpretar patrones y relaciones, ya que proporcionan representaciones granulares de la estructura de las redes. Al permitir que

el programa se modifique para otros problemas, como la detección de patrones y anomalías en la conectividad de la red, se destaca el enfoque gráfico.

Los gráficos son un medio eficaz para estudiar la complejidad de las conexiones IP, ofreciendo una representación realista de la topología de la red y ayudando en la identificación de nodos críticos, debido a su eficiencia y capacidad de modificación en este escenario.

Leonardo Guillén:

Los grafos son estructuras que nos ayudan a manejar de forma más eficiente cantidades de datos muy extensos, tales como bases de datos en las que se almacenan registros o cualquier tipo de dato necesario.
Un ejemplo de los grafos para uso de contener cualquier tipo de dato, se dio en 1884, cuando Alexander Crum Brown lo utilizó haciendo referencia a los enlaces que existían entre los átomos, con el tiempo esto se ha adaptado casi para representar datos y enlaces entre ellos.

Además con esto se dio la teoría de los grafos, la cual nació con un estudio realizado por Leonhard Euler, resolviendo el problema de los puentes de konisberg. Un problema que consiste en una ruta eficiente que permita cruzar todos los puentes de una ciudad, cruzandolos una sola vez pero el estudio demostró su improbabilidad y esto llevó a más estudios complejos.

Uno de ellos fue la teoría de grafos que está pensada en la posibilidad de llenar un mapa geográfico con cuatro colores, de forma que países vecinos no comparten el mismo color. Esta investigación fue concluida un siglo después.

Si bien existen distintos tipos de grafos, para diversos propósitos. En este caso nosotros aplicamos los grafos dirigidos y no dirigidos utilizando una lista de adyacencia para su manejo, el grafo es muy eficiente para manejar una gran cantidad de datos, que en este caso son aproximadamente 16,000, además de manejar más eficientemente el uso de memoria. Por ello note la flexibilidad, dinamismo y eficiencia que permiten los grafos, en el uso en conjunto ya sea con listas de adyacencia o matriz, note que es más eficiente el uso de la lista en cuanto al tamaño que se utiliza en memoria, sin embargo esto da limitaciones al momento en el que no se quiera repetir alguna conexión ya que en cuanto a complejidad al manejarlo con la lista es mas tardado, siendo más fácil de uso de una matriz de adyacencia por su accesibilidad y su estructura completa con la representación de todas las conexiones.

Comprendí que los grafos son muy importantes y aplicables en diversas áreas, siendo aplicados en la actualidad en muchos casos, comprendí y explore cómo funcionan los grafos en las tecnologías que utilizamos a diario a nivel de software, siendo aplicadas a diario en muchísimas de las cosas que utilizamos, además de que la historia de cómo surgieron.

Código:

```cpp
#include <iostream>
#include <list>
#include <sstream>
#include <fstream>
#include <stack>
#include <queue>
#include <set>
using namespace std;
class Graph {
private:
    int V; // número de vértices
    int E; // número de Edges
    set<int> *adj;
public:
    Graph(int V, int E);
    void addEdge(int v, int w);
    int fanOut(int v);
    void displayList();
    void maxFanOut(int n);
};

Graph::Graph(int V, int E) {
    this->V = V;
    this->E = E;
    adj = new set<int>[V];
}

void Graph::addEdge(int v, int w) {
        adj[v].insert(w);

}

int Graph::fanOut(int v) {
        return adj[v].size();
}

void Graph::displayList() {
    cout << "--------------------------------" << endl;
    cout << "Lista de adyacencia" << endl;
    for (int i = 0; i < V; i++) {
        cout << "vertice: " << i << " [";
```

```
        for (set<int>::iterator j = adj[i].begin(); j != adj[i].end();
j++) {
            cout << *j << " ";
        }
        cout << "]" << endl;
    }
    cout << "--------------------------------" << endl;
}

void Graph::maxFanOut(int n) {
    int *maxFanOut = new int[n];
    for (int i = 0; i < n; i++) {
        maxFanOut[i] = 0;
    }
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < n; j++) {
            if ( fanOut(i) > fanOut(maxFanOut[j])) {
                for (int k = n - 1; k > j; k--) {
                    maxFanOut[k] = maxFanOut[k - 1];
                }
                maxFanOut[j] = i;
                break;
            }
        }
    }
    for (int i = 0; i < n; i++) {
        cout << "Mayor Fan Out numero " << i + 1 << " : " <<
maxFanOut[i] << " FanOut: " << fanOut(maxFanOut[i]) << endl;
    }
}

string extractIp(std::string line) {
    string mes, dia, hora, ip;
    stringstream extract(line);
    extract >> mes >> dia >> hora >> ip;
    return ip;
}

int main() {
    fstream bitacora("bitacora.txt");
    string line, ip, puerto;
    Graph grafo(1000, 1000);
    int arr[16807][4];
```

```
    int cont=0;
    while (getline(bitacora, line)) {
        ip = extractIp(line);
        istringstream iss(ip);
        char dot;
        int parte1, parte2, parte3, parte4;
        iss >> parte1 >> dot >> parte2 >> dot >> parte3 >> dot >>
parte4;
        arr[cont][0] = parte1;
        arr[cont][1] = parte2;
        arr[cont][2] = parte3;
        arr[cont][3] = parte4;
        grafo.addEdge(parte1, parte2);
        grafo.addEdge(parte2, parte3);
        grafo.addEdge(parte3, parte4);
        cont=cont +1;
    }
    int fanOut=0;
    int fanOutTemp=0;
    string ipBootMaster="";
    string tempIp="";
    for(int i=0;i<cont;i++){
        for (int j=0;j<4;j++){
            fanOutTemp=fanOutTemp+grafo.fanOut(arr[i][j]);
            if(j<=3){
                tempIp=tempIp+to_string(arr[i][j])+".";
            }else{
                tempIp=tempIp+to_string(arr[i][j]);
            }
        }
        if (fanOutTemp>fanOut){
            fanOut=fanOutTemp;
            ipBootMaster=tempIp;
        }
        fanOutTemp=0;
        tempIp="";
    }
    int n;
    cout<<"Dime el numero de segmentos con mayor Fan Out que quieres
conocer "<<endl;
    cin>>n;
    grafo.maxFanOut(n);
```

```
    cout<<"Ip BootMaster : "<<ipBootMaster<<" con "<< fanOut<<"
conexiones de salida"<<endl;
    return 0;
}
```

Referencias:

Wikipedia contributors. (s/f). Teoría de grafos. Wikipedia, The Free Encyclopedia. https://es.wikipedia.org/w/index.php?title=Teor%C3%ADa_de_grafos&oldid=154969626

Kike [@Kike1303]. (2020, octubre 23). El origen de los Grafos: la historia de los puentes de Königsberg. Youtube. https://www.youtube.com/watch?v=AlEbJWq9i-Q

ParaDoppler [@paradoppler2132]. (2021, mayo 26). El maravilloso mundo de la Teoría de Grafos. Youtube. https://www.youtube.com/watch?v=mZMJJV6jDec