

1.-Introducción

A lo largo del semestre obtuvimos conocimientos sobre múltiples estructuras de datos y métodos de búsqueda de los mismos. Algunas de las estructuras utilizadas para resolver las problemáticas que se nos impusieron fueron las listas ligadas, los árboles BST, los grafos y las matrices hash tables.

Listas ligadas

Listas compuestas de uno o varios nodos los cuales contienen los datos correspondientes a cada nodo y un apuntador hacia el próximo nodo. Para la situación problema, se tomo como la información de cada ip es un nodo individual. Al ingresar todos los ips, se ordenaron con un método de ordenamiento, y usando una búsqueda se encuentran los resúmenes según el rango de ip inicial y final.

BST


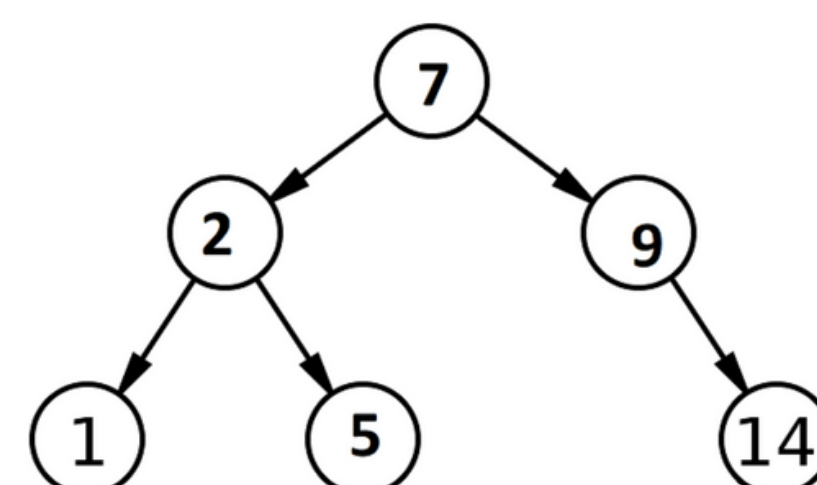
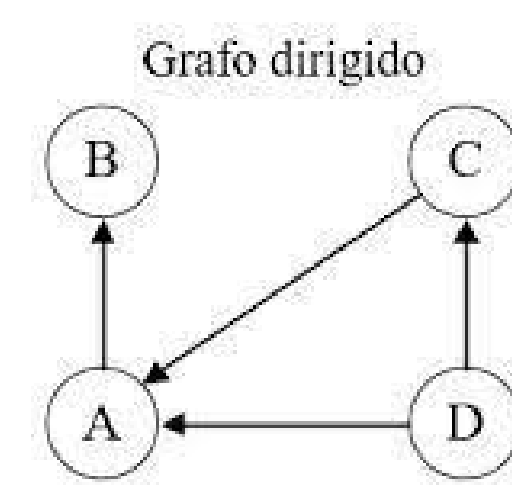
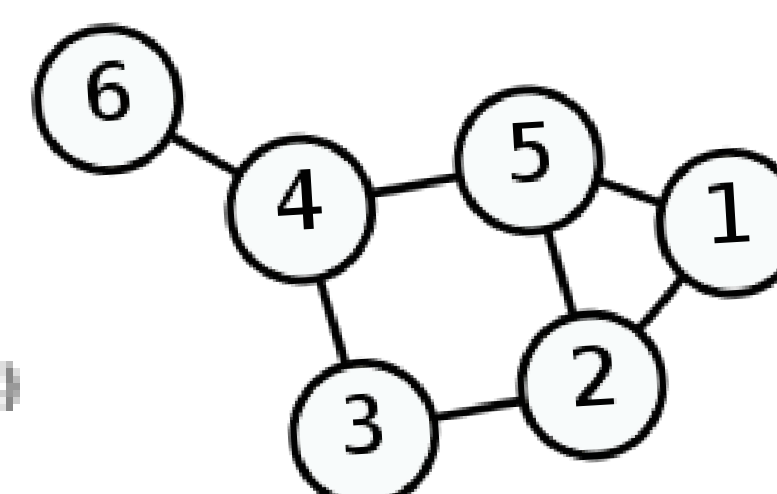
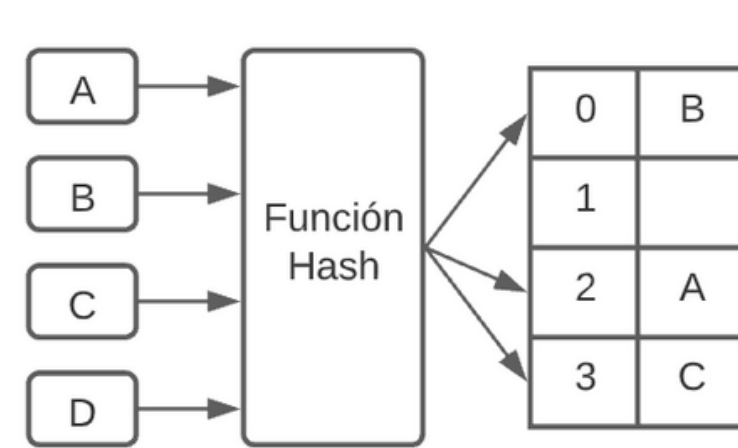
Los árboles binarios de búsqueda son estructuras de datos ordenadas jerárquicamente como padre-hijo. Cada nodo puede tener solo dos hijos, y el valor del hijo izquierdo debe ser menor mientras que el derecho debe ser mayor al nodo padre. En la situación problema, se creo un BST donde la llave era el número de accesos de cada ip, por lo que cada nodo tenía una llave y los datos. Se tiene también un vector con los accesos para imprimir los cinco ips con más accesos.

Grafos

Los grafos son conjuntos de objetos llamados nodos o vértices que están ligados entre ellos por aristas. Existen los grafos dirigidos y no dirigidos. Hacia la situación problema se cuenta con un grafo dirigido. Inicialmente se definen el número total de vértices (ips) que van a haber. Al continuar el desarrollo se toma cada arista con el ip de origen y el ip de destino, si el ip de origen ya existía, se añade el ip de destino dentro de la lista de adyacencia de el ip de origen existente. De esta manera se creo un grafo de ips conjuntos direccionalmente

Hash tables

Las tablas hash son listas en las que los elementos se almacenan mediante una función hash y una llave, localizándolos después por medio de esta misma función. Para la situación problema, las hash tables fueron desarrolladas similarmente a los grafos. En caso de que se encuentre un resumen de un ip que ya existe, este nuevo resumen se añade a una lista que contienen los resúmenes de ese ip.

<h3>Listas ligadas</h3> <pre> addFirst(dt,dy,h,i,r) //dt = data di = day h = hour i = ip r = razon size + 1 head = new Node(dt, dy,h,i,r) tail = head create(dt, dy, h,i,r) *current = head *currPrev = tail IF size==0 THEN addFirst(dt, dy,h,i,r) RETURN WHILE current->getNext != NULL currPrev = current current = current->getNext currPrev = current current->getNext(new Node(dt,dy,h,i,r)) size + 1 current = current->getNext current->setIdx(size) </pre> 	<h3>BST</h3> <pre> insert(d, ip) // d = data i = ip current = root father = NULL WHILE current != NULL father = current IF current->getData > d THEN current = current->getLeft ELSE current = current->getRight IF father = NULL THEN Root = new Node(d,i) topList.push_back(ipAccess(d,i,i)) ELSE IF father->getData > d father->setLeft(new Node(d,i)) THEN topList.push_back(ipAccess(d,i,i)) ELSE father->setRight(new Node(d,i)) topList.push_back(ipAccess(d,i,i)) </pre> 
<h3>Grafos</h3> <pre> Insert(a) // a=arista idxOrigen = findIdx(aris.origen) IF idxOrigen = -1 THEN Vértices.push_back(aris.origen) idxOrigen = numVertices numVertices + 1 vector<T> aris listaAdj.push_back(aris) idxDestino = findIdx(aris.destino) IF idxDestino = -1 THEN vertices.push_back(aris.destino) idxDestino = numVertices numVertices + 1 vector<T> aris listaAdj.push_back(aris) listaAdj[idxOrigen].push_back(aris) </pre>  	<h3>Hash tables</h3> <pre> vector<string> info unordered_map<string, vector<string>> umap; WHILE count != 0 in>>mes>>dia>>hora>>ip getline(in,razon) info = mes + "" + dia + "" + hora + "" + ip + "" + razon tempIp = ip tempIp = erasePort(tempIp) IF (umap.count(tempIp) == 0) THEN vector<string> pr = {info} pair<string, vector<string>> pr2= {tempIp,pr} umap.insert(pr2) ELSE umap[tempIp].push_back(info) count - 1 cout << count << endl </pre> 

Las complejidades se estan obteniendo por parte del metodo de busqueda en la situacion problema. Se toma en cuenta la complejidad computacional de los metodos de busqueda para cada estructura ya que es la parte mas comun dentro de estas.

Listas Ligadas	BST	Grafos	Hash tables
$O(n)$	$O(\log n)$	$O(V+E)$	$O(1)$

Merge: $O(n \log n)$

Cada estructura de datos tiene su propia especialidad, al trabajar en la situación problema podemos observar como alguna de estas realizaron el método de ordenamiento o de búsqueda significativamente más rápido que algunas otras. Pudimos observar como las hash tables funcionaban significativamente más rápidas en sus métodos de búsqueda a comparación a los grafos o listas, sin embargo se utilizan cuando el ordenamiento de la información no importa.