Tecnológico de Monterrey

Edgar Cruz Vázquez (A01730577)

TC1031.11: Programación de estructuras de datos y algoritmos fundamentales

Act 4.3 – Actividad Integral de Grafos (Evidencia Competencia)

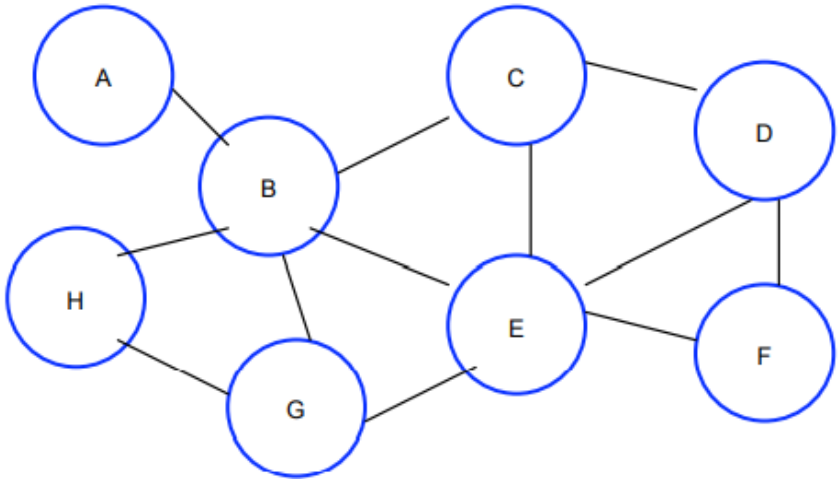
GRAFOS:

Definición:

Un grafo, G, es un par, compuesto por dos conjuntos V y A. Al conjunto V se le llama conjunto de vértices o nodos del grafo. A es un conjunto de pares de vértices, estos pares se conocen habitualmente con el nombre de arcos o ejes del grafo. Se suele utilizar la notación G = (V, A) para identificar un grafo.

Ejemplo:

G = {V, A}
V = {A, B, C, D, E, F, G, H}
A = { {A,B}, {B,C}, {B,E}, {B,G}, {B,H}, {C,D}, {C,E}, {D,F}, {D,E}, {E,G}, {G,H} }



Los grafos permiten presentar conjuntos de objetos arbitrariamente relacionados. Se puede asociar el conjunto de vértices con el conjunto de objetos y el conjunto de arcos con las relaciones que se establecen entre ellos.

Tipos de grafos:

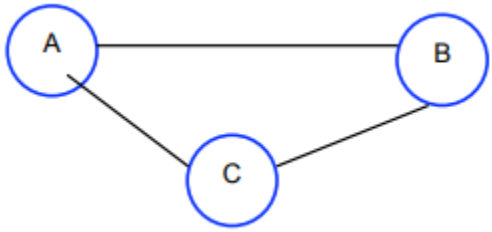
Grafo no dirigido (grafo):
En un grafo no dirigido el par de vértices que representa un arco no está ordenado. Por lo tanto, los pares (v1, v2) y (v2, v1) representan el mismo arco.

Tecnológico de Monterrey

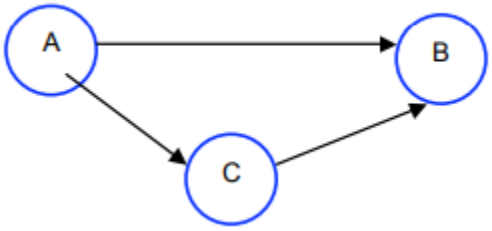
Edgar Cruz Vázquez (A01730577)

TC1031.11: Programación de estructuras de datos y algoritmos fundamentales

Act 4.3 – Actividad Integral de Grafos (Evidencia Competencia)

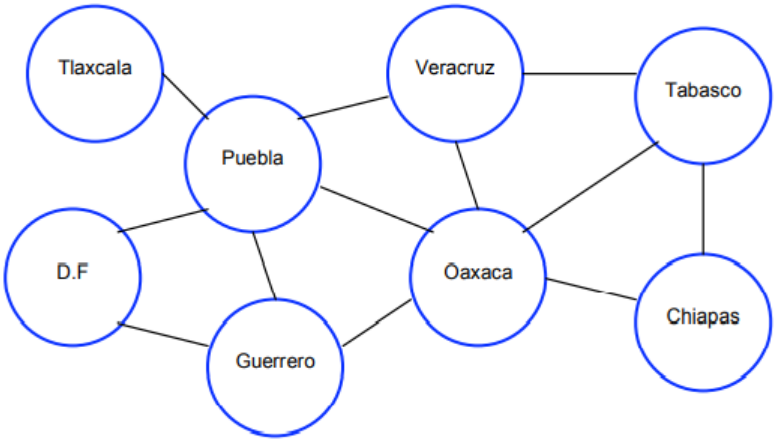


Grafo dirigido (digrafo):
En un grafo dirigido cada arco está representado por un par ordenado de vértices, de forma que y representan dos arcos diferentes.



Grafo etiquetado:
Un grafo cuyos nodos tienen asociada información. Un ejemplo común es que los nodos contengan los nombres de ciudades en un mapa aéreo.

Ejemplo: conexiones entre algunos estados de la república mexicana.



Tecnológico de Monterrey

Edgar Cruz Vázquez (A01730577)

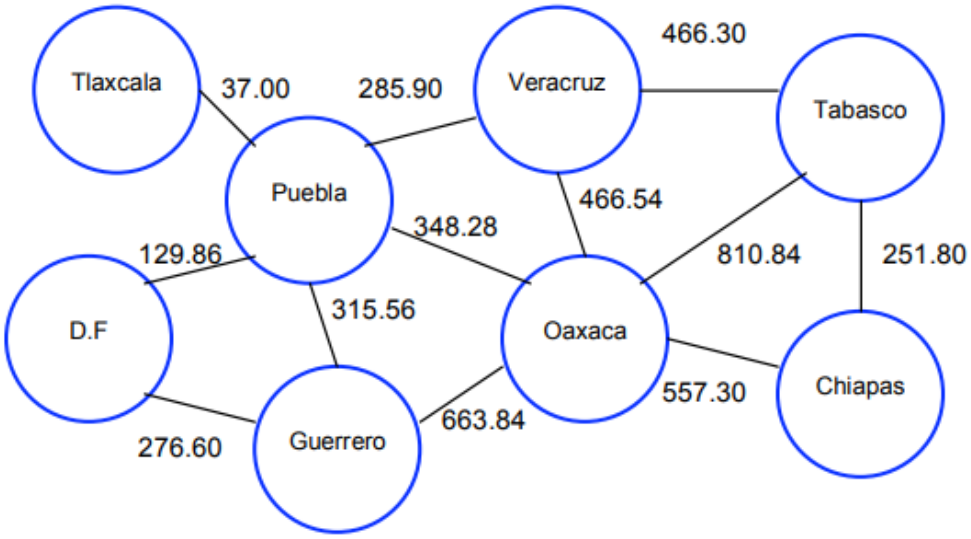
TC1031.11: Programación de estructuras de datos y algoritmos fundamentales

Act 4.3 – Actividad Integral de Grafos (Evidencia Competencia)

Grafo ponderado:
Un grafo donde cada arco tiene asociado un valor numérico o peso. Un grafo donde se indica las distancias de diferentes rutas en un mapa aéreo es un ejemplo típico.

Ejemplo:

Conexiones entre algunas ciudades de la república mexicana, donde el peso de cada arista significa la distancia en kilómetros desde la ciudad capital de un estado al otro.



Representación de los grafos:

Para representar grafos, existen 4 formas:

1. Matriz de adyacencia.
2. Listas de adyacencia.
3. Multilistas de adyacencia.
4. Matriz de incidencia.

Para la solución de esta actividad se integraron las listas de adyacencia, por lo cual va a ser el que se explicara.

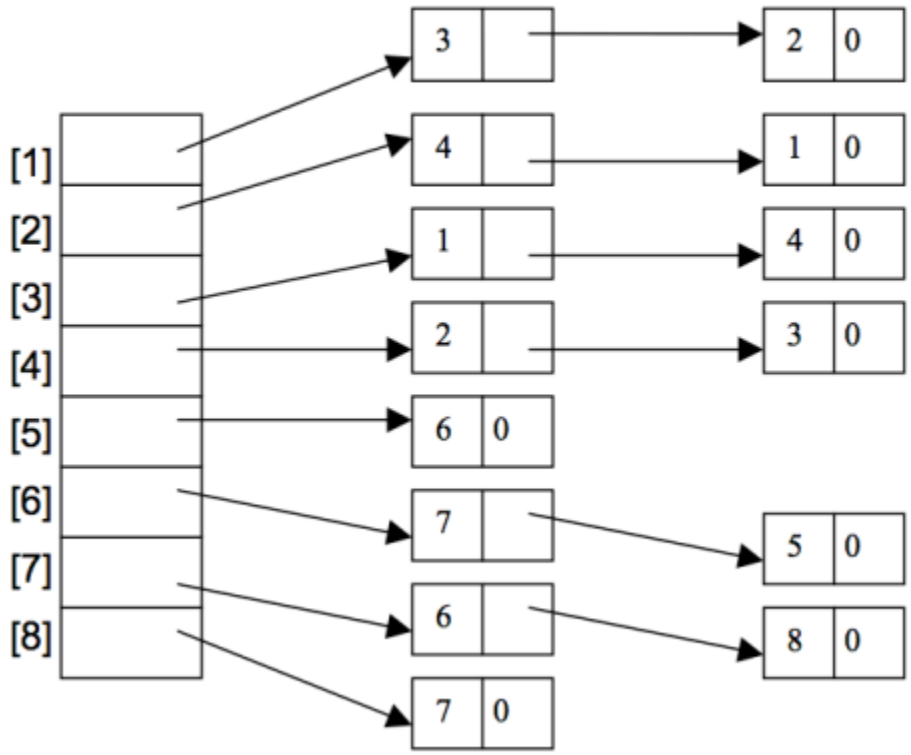
Lista de adyacencia:
Representa una lista de todos los vértices.
Cada objeto vértice guarda un alista de adyacencia con un objeto arista para cada vértice alcanzable desde él. Es una lista de listas.

Tecnológico de Monterrey

Edgar Cruz Vázquez (A01730577)

TC1031.11: Programación de estructuras de datos y algoritmos fundamentales

Act 4.3 – Actividad Integral de Grafos (Evidencia Competencia)



Los grafos son una estructura de datos que sirve para modelar una infinidad de problemas que se pueden expresar de manera computacional. A diferencia de la estructura de árbol, los grafos no son una estructura rígida, por lo cual permite utilizarlos en aplicaciones de este tipo:

- Mapas: está claro que las rutas de un mapa pueden ser modelado por grafos, para encontrar la ruta más corta o con menos congestión.

- Computación distribuida: Computadoras conectadas a una red, bueno internet es el mayor sistema distribuido del mundo, cada una de nuestras computadoras podría ser un nodo dentro de un enorme grafo. Por lo cual cuando estas usando internet indirectamente esta todo trabajando sobre grafos.

- Redes sociales: Un ejemplo es Facebook, usa grafos para manejar las relaciones de amistad entre personas.

- Inteligencia artificial: Las famosas redes neuronales, siendo algoritmos que son modelados a través de grafos

Tecnológico de Monterrey

Edgar Cruz Vázquez (A01730577)

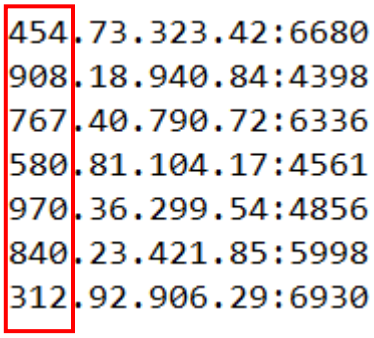
TC1031.11: Programación de estructuras de datos y algoritmos fundamentales

Act 4.3 – Actividad Integral de Grafos (Evidencia Competencia)

Los grafos tienen una importancia tremenda, muchas aplicaciones serían intratables de implementar y prácticamente imposibles de computar en tiempo óptimos.

Explicación:

En este ejercicio tomamos las ips del archivo y de ahí únicamente tomamos los primeros números antes del del primer punto:



para este proceso leímos el doc, guardamos las líneas en un vector, separamos los datos por cada espacio en un vector de vectores, extraimos del vector el índice de las ips insertándolas en un vector de strings, separamos las ips por cada punto y extraimos los primeros números y los insertamos en un vector de enteros, hicimos esa conversión por conveniencia para así aplicar con más facilidad el método de ordenamiento Quicksort y ordenar los números.

Funciones del Quicksort:

```
void Graph::quicksort(vector<int> &lista, int left, int right){
    if(left < right){
        int pivotIndex = partition(lista,left,right);
        quicksort(lista, left, pivotIndex - 1);
        quicksort(lista, pivotIndex, right);
    }
}
```

```
int Graph::partition(vector<int> &lista, int left, int right){
    int pivotIndex = left + (right - left) / 2;
    int pivotValue = lista[pivotIndex];
    int i = left, j = right;
    int temp;
    while(i <= j){
        while(lista[i] < pivotValue){
            i++;
        }

        while(lista[j] > pivotValue){
            j--;
        }

        if(i <= j){
            temp = lista[i];
            lista[i] = lista[j];
            lista[j] = temp;
            i++;
            j--;
        }
    }
    return i;
}
```

Una vez implementamos el ordenamiento, creamos dos funciones en las cuales nos ayudaran a contar los números repetidos, y también una vez contado ese número, borrar el dato

Para que no se repita, mediante un for.

Una vez creados esas funciones, inicializamos dos struct para ir creando los vértices y aristas, uno llamado nodo, que tendrá como dato la ip, las veces que se repite esa ip, el numero de conexiones que presenta y su lista de adyacencia.

Por su parte la arista tendrá como dato el vértice.

Tecnológico de Monterrey

Edgar Cruz Vázquez (A01730577)

TC1031.11: Programación de estructuras de datos y algoritmos fundamentales

Act 4.3 – Actividad Integral de Grafos (Evidencia Competencia)

```
Nodo{
    int numero;
    int conex;
    int repeat;
    Arista* adyacencia;
    Nodo* siguiente;
}*r;

Arista{
    int vrt;
    Arista* siguiente;
};
```

Lo que estamos haciendo en esta parte es crear dos linked list, uno siendo la de los nodos y otro siendo la lista de sus adyacentes de ese nodo.

Creamos una función que nos ayudara a crear la primera lista de nodos (inicialización), una vez creada la función juntamos las funciones de suma y elimina para que conforme vaya pasando el vector, nos vaya devolviendo el numero de la ip y las veces que se repiten.

```
int Graph::cuenta(vector<int> &lista, int t, int n){
    int x = 0;
    for(int i = 0; i < t; i++){
        if(n == lista[i]){
            x++;
        }
    }
    return x;
}
```

```
void Graph::Elimina(vector<int> &lista, int t, int n){
    for(int i = 0; i < t; i++){
        if(n == lista[i]){
            lista[i] = 0;
        }
    }
}
```

```
nodo* Graph::insertarNodo(nodo* r,int numero, int conex, int repeat){
    if(r == NULL){
        r = new nodo;
        r->numero = numero;
        r->conex = conex;
        r->repeat = repeat;
        r->adyacencia = NULL;
        r->siguiente = NULL;
    }

    else{
        r->siguiente = insertarNodo(r->siguiente, numero, conex, repeat);
    }

    return r;
}
```

Tecnológico de Monterrey

Edgar Cruz Vázquez (A01730577)

TC1031.11: Programación de estructuras de datos y algoritmos fundamentales

Act 4.3 – Actividad Integral de Grafos (Evidencia Competencia)

```
for(int i = 0; i < listaN.size(); i++){
    int n;
    int re;
    n = listaN[i];
    if(n != 0){
        re = Graph::cuenta(listaN,listaN.size(), n);
        r = g.insertarNodo(r,n,0,re);
        Graph::Elimina(listaN, listaN.size(), n);
    }
}
```

Una vez inicializado cada nodo, con el numero de la ip y las veces que se repite, lo que se hizo después crear una función insertarArista, que tendrá como parámetros dos números (el source y el destiny), primero verificando que ambos nodos existan para que después este se conecta con otra función llamada agregarArista que lo que hará será es añadir al nodo en su lista de adyacencia la ip de la cual se unirá (Edge) y le sumara uno al nodo en su número de conexiones.

```
void Graph::insertarArista(int src, int dest){
    if(r == NULL)
        return;

    Nodo *aux, *aux2;
    Arista *nuevo = new arista;
    aux = r;
    aux2 = r;

    while(aux2 != NULL){
        if(aux2->numero == dest)
            break;
        aux2 = aux2->siguiente;
    }

    if(aux2 == NULL){
        cout << "Error: vertice dest no encontrado.";
        return;
    }

    while(aux != NULL){
        if(aux->numero == src){
            agregarArista(aux,aux2,nuevo);
            return;
        }
        aux = aux->siguiente;
    }

    if(aux == NULL){
        cout << "Error: vertice src no encontrado.";
        return;
    }
```

```
void Graph::agregarArista(nodo* aux, nodo* aux2, arista* nuevo){
    Arista *a;
    if(aux->adyacencia == NULL){
        aux->adyacencia = nuevo;
        nuevo->vrt = aux2->numero;
        aux->adyacencia->siguiente = NULL;
        aux->conex++;
        return;
    }
    else{
        a = aux->adyacencia;
        while(a->siguiente != NULL)
            a = a->siguiente;
        nuevo->vrt = aux2->numero;
        a->siguiente = nuevo;
        a->siguiente->siguiente = NULL;
        aux->conex++;
        return;
    }
}
```

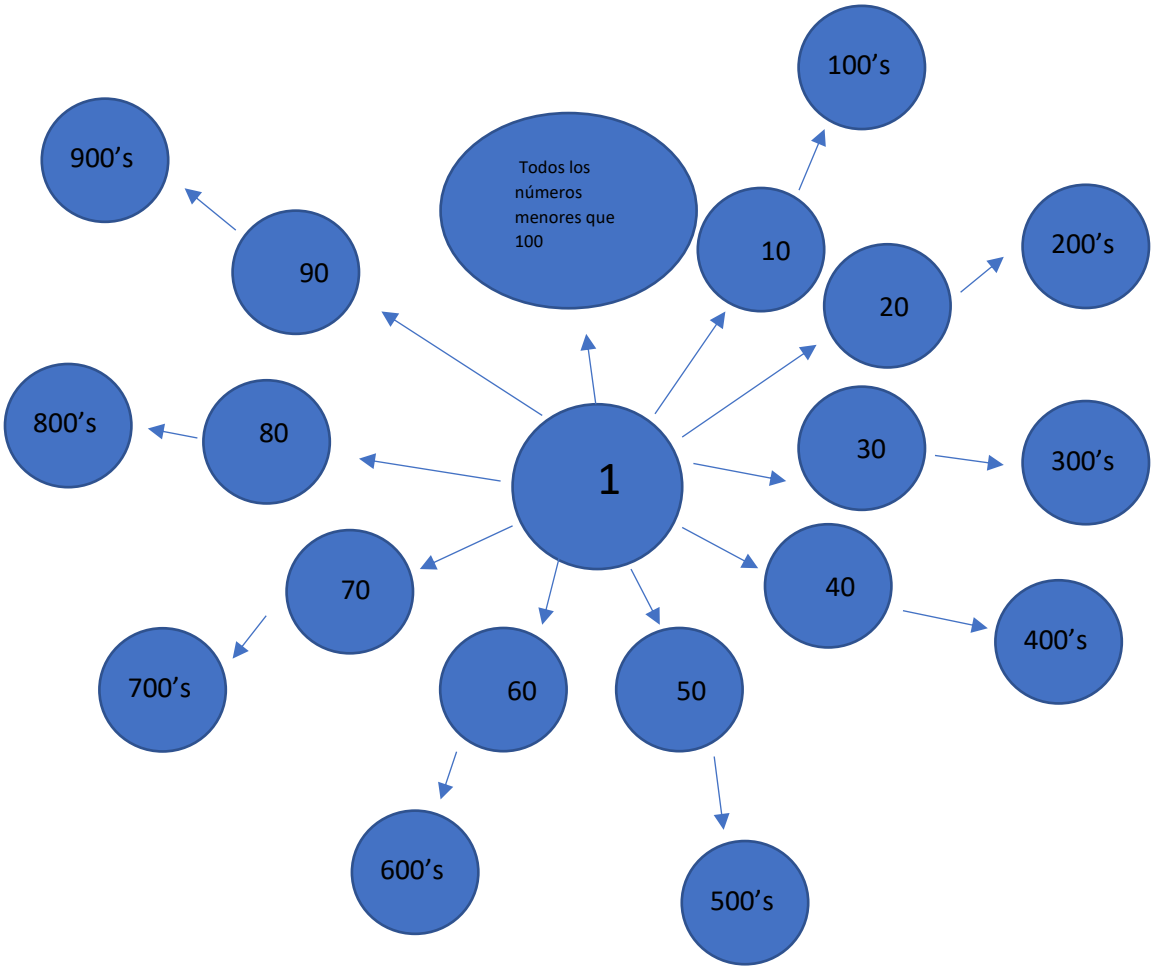
Una vez hecho ambas funciones, tenemos que visualizar el nodo en el cual queremos visualizarnos: Crear un nodo en el cual el centro sea la ip de 1 y este se conecte con todos los demás nodos que sean menores de 100, desúes que el nodo con la ip 10 se conecte con todos los 100, el 20 con todos los doscientes, 30 con los trescientes y así sucesivamente hasta el 90 que se conecte con todos los 900, creando un nodo parecido a esto:

Tecnológico de Monterrey

Edgar Cruz Vázquez (A01730577)

TC1031.11: Programación de estructuras de datos y algoritmos fundamentales

Act 4.3 – Actividad Integral de Grafos (Evidencia Competencia)



Una vez entendido la forma del grafo, creamos una función llamada insertConex, que lo que hace es hacer las conexiones mostradas en el ejemplo, recibiendo como parámetro todos los nodos y haciendo comparaciones conforme vaya pasando por la lista de los nodos, si en caso ese número de ip es menor a 100, se le aumente el número de conexiones y se le añada las ip en su lista de adyacencia, de igual forma se harán para las conexiones de 10, 20, 30, 40, 50, 60, 70, 80 y 90.

Tecnológico de Monterrey

Edgar Cruz Vázquez (A01730577)

TC1031.11: Programación de estructuras de datos y algoritmos fundamentales

Act 4.3 – Actividad Integral de Grafos (Evidencia Competencia)

```
void Graph::insertConex(nodo* r){
    Nodo *aux, *aux2;
    aux = r;
    aux2 = r;
    int u;

    while(aux != NULL){
        u = aux->numero;
        if(aux2->numero == u){
            aux = aux->siguiente;
            continue;
        }

        else if(u < 100){
            insertarArista(aux2->numero, u);
            insertarArista(u, aux2->numero);
            aux = aux->siguiente;
            continue;
        }

        else if((u >= 100) && (u < 200)){
            while(aux2->numero != 10){
                aux2 = aux2->siguiente;
            }
            insertarArista(aux2->numero, u);
            insertarArista(u, aux2->numero);
            aux = aux->siguiente;
            continue;
        }

        else if((u >= 200) && (u < 300)){
            while(aux2->numero != 20){
```

En esta imagen de la función insertConex ira haciendo dos iteraciones en donde, primero verificamos en caso que la ip en donde se presenta nuestro aux2 es igual a u, en donde u guarda el número de la ip de nuestro aux, cambia al siguiente nodo el aux y vuelve a reiniciar el ciclo y también actualizarse u con el numero ip del siguiente nodo, hacemos una serie de comparaciones para ver en que parámetro se encuentra u, ya sea entre los menores a 100, entre los 100, los 200 y así sucesivamente, en esta parte tomemos como ejemplo cuando la u es menor a 100, entonces insertamos la arista entre el aux2->numero = 1 y la u y luego volvemos a llamar la función pero invertimos los valores, haciendo que el grafo sea un grafo no dirigido, cambiamos el aux al siguiente nodo y volvemos a repetir el ciclo y así sucesivamente hasta que nuestro aux sea igual a NULL.

Con estas funciones podemos ver en cada vértice, la ip, el numero de repeticiones que tiene, su número de conexiones y también podemos ver a que ips son a los que se conecta:

```
IP: 1 Repeticiones: 23 Conexiones: 98
IPS adyacentes del vertice: 1
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
9 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
```

```
IP: 1  Repeticiones: 23  Conexiones: 98
IP: 2  Repeticiones: 14  Conexiones: 1
IP: 3  Repeticiones: 16  Conexiones: 1
IP: 4  Repeticiones: 12  Conexiones: 1
IP: 5  Repeticiones: 16  Conexiones: 1
IP: 6  Repeticiones: 9  Conexiones: 1
IP: 7  Repeticiones: 11  Conexiones: 1
IP: 8  Repeticiones: 21  Conexiones: 1
IP: 9  Repeticiones: 20  Conexiones: 1
IP: 10  Repeticiones: 16  Conexiones: 101
IP: 11  Repeticiones: 15  Conexiones: 1
IP: 12  Repeticiones: 17  Conexiones: 1
IP: 13  Repeticiones: 19  Conexiones: 1
IP: 14  Repeticiones: 15  Conexiones: 1
IP: 15  Repeticiones: 16  Conexiones: 1
IP: 16  Repeticiones: 26  Conexiones: 1
IP: 17  Repeticiones: 14  Conexiones: 1
IP: 18  Repeticiones: 5  Conexiones: 1
IP: 19  Repeticiones: 10  Conexiones: 1
IP: 20  Repeticiones: 14  Conexiones: 101
```

al imprimir todas las IP's pudimos darnos cuenta que las IP's que empiezan en 10, 20, 30, 40, 50, 60, 70 y 80, todos tienen 101 conexiones siendo los que más conexiones tienen.

Para poder encontrar el boot master, tenemos que ver cual es la IP que más veces se repite, para poder hacer eso creamos una función llamada maxRepeat el cual nos imprimirá el número de la IP qué más se repita y el número de sus repeticiones.

Tecnológico de Monterrey

Edgar Cruz Vázquez (A01730577)

TC1031.11: Programación de estructuras de datos y algoritmos fundamentales

Act 4.3 – Actividad Integral de Grafos (Evidencia Competencia)

```
void Graph::maxRepeat(nodo* r){
    int n, v;
    Nodo *aux;
    aux = r;
    n = aux->repeat;
    while(aux != NULL){
        if(aux->siguiente == NULL)
            break;

        else if(n < aux->siguiente->repeat){
            aux = aux->siguiente;
            n = aux->repeat;
            v = aux->numero;
        }

        else{
            aux = aux->siguiente;
        }
    }
    cout << "El vertice: " << v << " es el que mas se repite con: " << n << " veces." << endl;
    return;
}
```

```
La IP que mas se repite es el 708 con 32 veces.
```

Una vez encontrado la IP con más repeticiones, es ahí donde presumiblemente es donde se encuentre el boot master.

Con lo explicado en este doc, logramos determinar los siguientes puntos:

- Determinar el fan-out de cada nodo.
- Los nodos con mayores fan-out
- La posible dirección en donde se encuentra el boot master.

	CASOS DE PRUEBA ACT 4.3				
No.	Precondiciones	Resumen	Pasos	Resultado Esperado	Resultado obtenido
1	Tener el código y una linked list de nodos.	El usuario introduce un arco con IP invalidas	1. Iniciar el programa. 2. Llamar a la función e insertarArista con valores inválidos de IPS.	Se despliegue un mensaje de IPS no encontradas.	Se despliega en la pantalla el mensaje de IPS no encontradas.

Tecnológico de Monterrey

Edgar Cruz Vázquez (A01730577)

TC1031.11: Programación de estructuras de datos y algoritmos fundamentales

Act 4.3 – Actividad Integral de Grafos (Evidencia Competencia)

2	Tener el código y no tener vértices	El usuario llama la función insertConex sin la existencia de vértices.	1. Correr el programa. 2. Llamar la función insertConex.	El programa debería imprimir un mensaje diciendo que no puede hacer conexiones si no hay vértices.	Se despliega en pantalla el mensaje de no poder crear conexiones.

CASO 1:

```
Error: vertice dest no encontrado.
```

CASO 2:

```
No puedo crear conexiones si no existen vertices.
```