Lógica de programación I

Juan Pablo Restrepo Uribe

Ing. Biomedico

MSc. Automatización y Control Industrial

jprestrepo@correo.iue.edu.co

Institución Universitaria de Envigado

CARACTERIZACIÓN DE GRAFOS

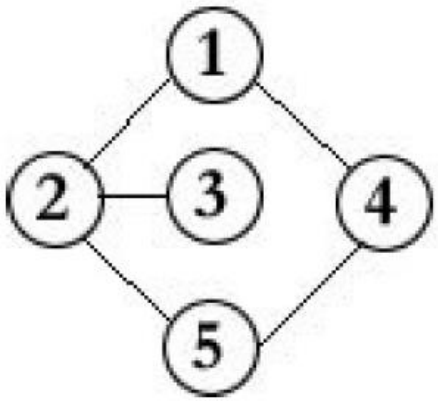
Grafos simples

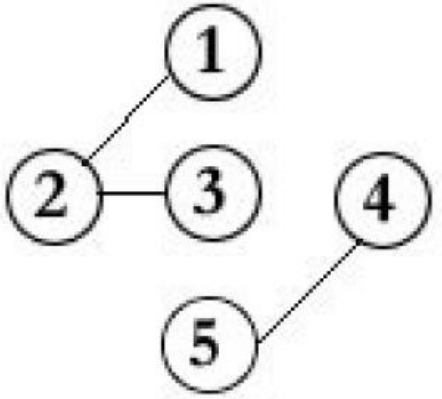
Un grafo es simple si a lo más existe una arista uniendo dos vértices cualesquiera. Esto es equivalente a decir que una arista cualquiera es la única que une dos vértices específicos. Un grafo que no es simple se denomina multigrafo.

Grafos conexos

Un grafo es conexo si cada par de vértices está conectado por un camino; es decir, si para cualquier par de vértices (a, b), existe al menos un camino posible desde a hacia b. Un grafo es doblemente conexo si cada par de vértices está conectado por al menos dos caminos disjuntos; es decir, es conexo y no existe un vértice tal que al sacarlo el grafo resultante sea disconexo.

CARACTERIZACIÓN DE GRAFOS





CARACTERIZACIÓN DE GRAFOS

Grafos completos

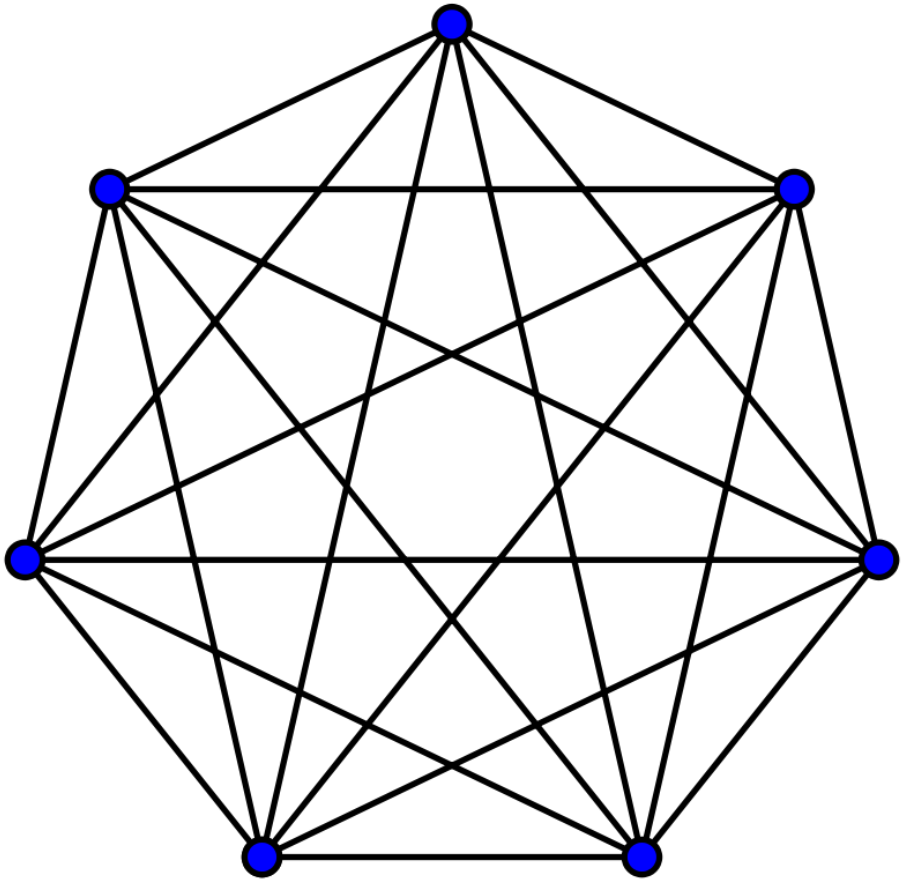
Un grafo es completo si existen aristas uniendo todos los pares posibles de vértices. Es decir, todo par de vértices (a,b) debe tener una arista e que los une.

El conjunto de los grafos completos es denominado usualmente $\mathbb{K}$, siendo $\mathbb{K}_n$ el grafo completo de n vértices.

Un $\mathbb{K}_n$, es decir, grafo completo de n vértices tiene exactamente $\frac{n(n-1)}{2}$ aristas.

La representación gráfica de los $\mathbb{K}_n$ como los vértices de un polígono regular da cuenta de su peculiar estructura.

CARACTERIZACIÓN DE GRAFOS



CARACTERIZACIÓN DE GRAFOS

Grafos bipartitos

Es un grafo cuyos vértices se pueden separar en dos conjuntos disjuntos, de manera que las aristas no pueden relacionar vértices de un mismo conjunto.

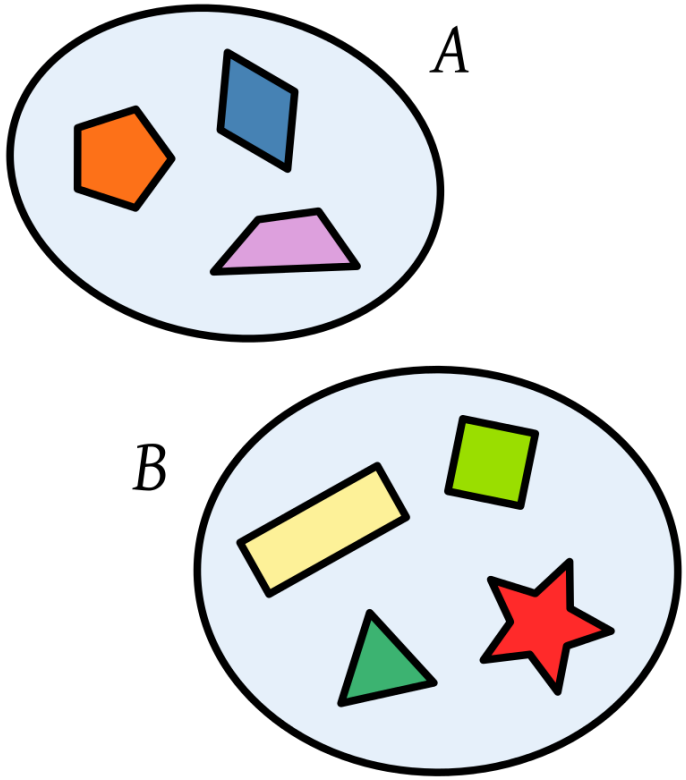
Un grafo G es bipartito si puede expresarse como $G = \{V_1 \cup V_2, A\}$ (es decir, sus vértices son la unión de dos grupos de vértices), bajo las siguientes condiciones:

- V_1 y V_2 son disjuntos y no vacíos.
- Cada arista de A une un vértice de V_1 con uno de V_2.
- No existen aristas uniendo dos elementos de V_1; análogamente para V_2.

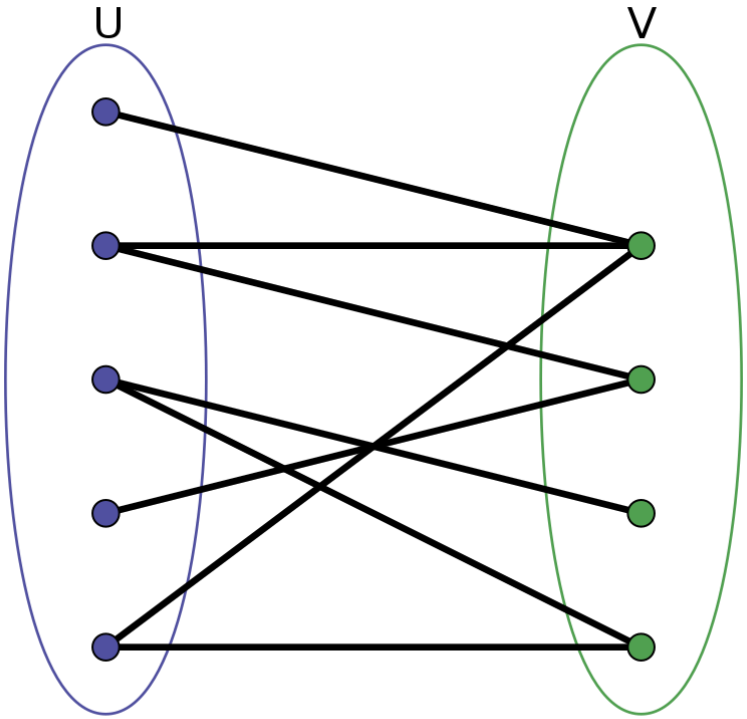
CARACTERIZACIÓN DE GRAFOS

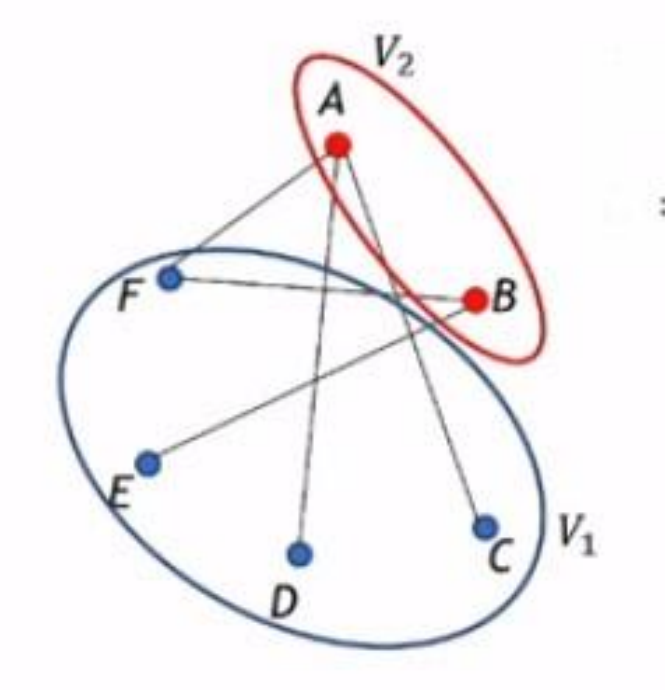
Grafos bipartitos

En teoría de conjuntos, dos conjuntos son disjuntos o ajenos si no tienen ningún elemento en común. En otras palabras, dos conjuntos son disjuntos si su intersección es vacía.



CARACTERIZACIÓN DE GRAFOS



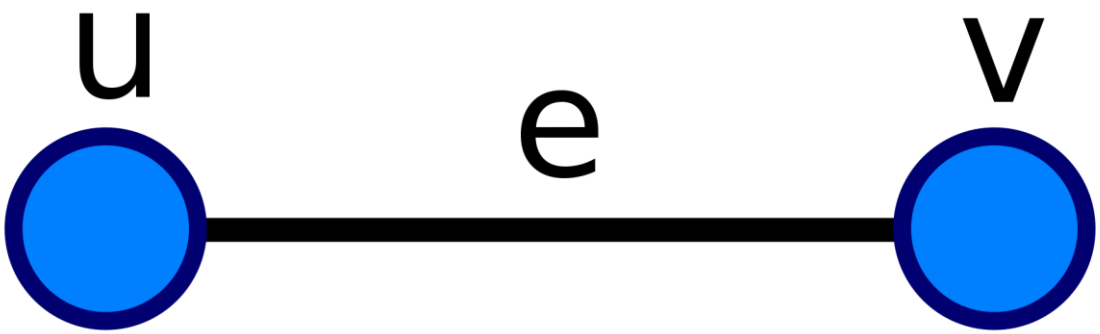


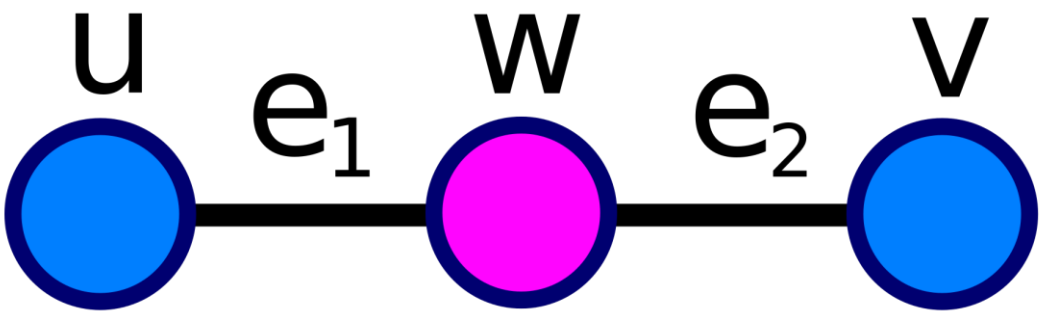
SUBDIVISIÓN ELEMENTAL DE UNA ARISTA

Es el proceso de dividir una arista en dos partes mediante la inserción de un nuevo nodo en medio de ella. Esto implica agregar un nuevo nodo (vértice) y dos nuevas aristas para mantener la conectividad del grafo.

Tenemos un grafo con dos nodos, u y v, y una arista e que los está uniendo, si queremos subdividir esta arista, podemos agregar un nuevo nodo w entre los nodos u y v.

SUBDIVISIÓN ELEMENTAL DE UNA ARISTA





ELIMINACIÓN DÉBIL DE UN VÉRTICE

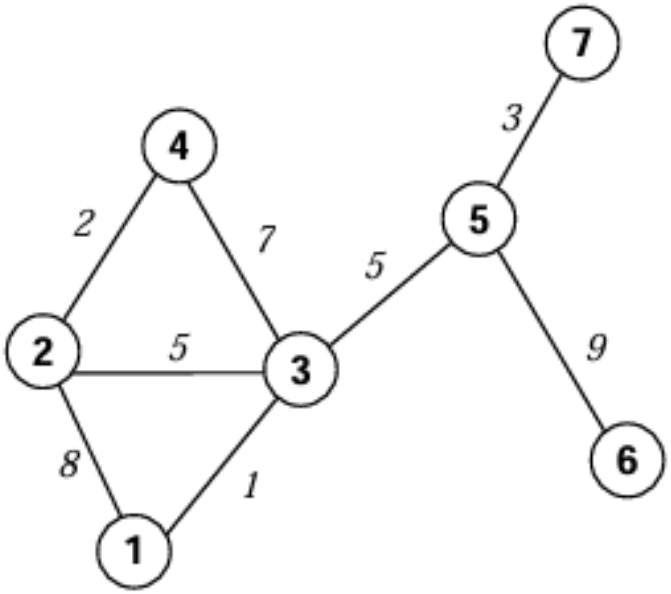
Consiste en eliminar un vértice del conjunto de vértices del grafo sin afectar las conexiones entre otros vértices. Es decir, las aristas que estaban conectadas al vértice eliminado permanecen en el grafo, pero ya no se considera ese vértice como parte del grafo.

Tenemos un grafo con vértices A, B y C, y las siguientes aristas: AB, BC y AC. Si eliminamos el vértice B de manera débil, las aristas AB y BC aún existirán en el grafo, pero el vértice B ya no será parte del conjunto de vértices del grafo. Entonces, el grafo resultante tendría dos vértices (A y C) y dos aristas (AB y AC), ya que la arista BC, que estaba conectada al vértice B, sigue existiendo, pero ahora se considera una conexión directa entre los vértices A y C.

OPERACIONES

- **Insertar vértice:** Consiste en añadir una nueva entrada en la tabla de vértices (estructura de datos que almacena los vértices) para el nuevo nodo.
- **Insertar arista:** Cuando se inserte una nueva arista en el grafo, habrá que añadir un nuevo nodo a la lista de adyacencia (lista que almacena los nodos a los que un vértice puede acceder mediante una arista)
- **Eliminar vértice:** Esta operación es inversa a la inserción de vértice. En este caso el procedimiento a realizar es la eliminación de la tabla de vértices del vértice en sí.
- **Eliminar arista:** Mediante esta operación se borra un arco del grafo.

OPERACIONES



vértice

vértice	1	2	3	4	5	6	7
1	0	8	1	∞	∞	∞	∞
2	8	0	5	2	∞	∞	∞
3	1	5	0	7	5	∞	∞
4	∞	2	7	0	∞	∞	∞
5	∞	∞	5	∞	0	9	3
6	∞	∞	∞	∞	9	0	∞
7	∞	∞	∞	∞	3	∞	0

vértice

vértice				
1	(2,8)	(3,1)		
2	(1,8)	(3,5)	(4,2)	
3	(1,1)	(4,7)	(2,5)	(5,5)
4	(2,2)	(3,7)		
5	(3,5)	(7,3)	(6,9)	
6	(5,9)			
7	(5,3)			

OPERACIONES

Operación	Matriz de Adyacencia	Lista de Adyacencia
Espacio ocupado	$\Theta(n^2)$	$\Theta(m+n)$
Longitud entre vértices / Existencia de aristas	$O(1)$	$O(\text{grado}(i))$
Recorrido vértices adyacentes a i	$\Theta(n)$	$\Theta(\text{grado}(i))$
Recorrido todas las aristas	$\Theta(n^2)$	$\Theta(m)$
Inserción/borrado arista	$O(1)$	$O(\text{grado}(i))$
Inserción/borrado vértice	$O(n^2)$	$O(n)$

Recorrido

- Un recorrido es un algoritmo que visita todos los nodos de un grafo
- Supondremos los grafos representados por matrices de adyacencia
- Los algoritmos más usados para recorrer grafos generalizan los recorridos de árboles
- Para el caso de grafo se necesita guardar información sobre los nodos que ya han sido visitados, de modo de no volver a visitarlos

Problema de la ruta óptima (shortest path)

Objetivo: Dada una red conectada, encontrar la ruta óptima (la de menor distancia) entre los vértices i y j.

Representación de la solución: La solución consistirá en proporcionar la secuencia de vértices, *v1…vk…vp* que forman la ruta óptima (*v1 = i, vp = j*)

Algoritmo Backtracking / Fuerza Bruta

El espacio de posibles soluciones son todos los posibles caminos simples que van desde el vértice i al vértice j. Una solución fuerza bruta consistiría en generar todos esos posibles caminos, calcular su coste (longitud), y obtener el mínimo. Para generar todos los posibles caminos simples lo más sencillo es utilizar un enfoque recursivo, basado en las ideas de la técnica de backtracking: Para ir del vértice i hasta el j se debe primero ir desde i hasta uno de sus vértices adyacentes, k (que puede ser el propio j).

Para generar todos los caminos basta entonces con un bucle en el que se recorren todos los vértices adyacentes de i y para cada uno de ellos (vértice k) se pide, recursivamente, obtener todos los caminos desde k hasta j. Como queremos los caminos simples, y en ellos no se debe pasar por cada vértice más de una vez, iremos marcando cada vértice ya visitado y no tomaremos en cuenta esos vértices si les encontramos en el proceso de generar un camino.

BFS (Breadth-First Search) - Búsqueda en Anchura

En la búsqueda en anchura, se exploran los nodos nivel por nivel, es decir, primero se visitan todos los nodos vecinos de un nodo antes de pasar a los nodos más alejados.
Proceso:

- Se empieza en un nodo inicial.
- Se utiliza una cola (queue) para almacenar los nodos que se deben visitar.
- Se marca el nodo inicial como visitado y se añaden todos sus nodos vecinos no visitados a la cola.
- Se extrae el primer nodo de la cola, se visita, y se añaden a la cola todos sus vecinos no visitados.
- El proceso se repite hasta que se hayan visitado todos los nodos.

DFS (Depth-First Search) - Búsqueda en Profundidad

En la búsqueda en profundidad, se explora lo más lejos posible a lo largo de una rama del grafo antes de retroceder.

Proceso:

- Se empieza en un nodo inicial.
- Se utiliza una pila (stack) o la recursión para visitar los nodos.
- Se marca el nodo inicial como visitado y se sigue visitando uno de sus vecinos no visitados hasta que no haya más nodos no visitados en esa rama.
- Luego, se retrocede (backtracking) y se exploran otras ramas no visitadas.

Programación Dinámica – Algoritmo de Floyd

Para aplicar programación dinámica al problema anterior primero resolveremos el problema restringido: En este caso lo normal es pedir la longitud de la ruta óptima entre i y j y renunciar a saber cuál es esa ruta optima. Es generalizar un poco el esquema backtracking visto anteriormente y hacer que k no tenga que ser un vértice adyacente a i sino que pueda ser cualquier vértice intermedio del camino que va desde i a j. El problema de calcular la función distancia entre i y j se plantea entonces recursivamente como el problema de calcular el mínimo de la distancia entre i y k más la distancia entre k y j para todo posible k:

$$dist(i,j) = \begin{cases} 0 & \text{si } i = j \\ \min\limits_{\forall k \neq i} \{ dist(i,k) + dist(k,j) \} & \text{si } i \neq j \end{cases}$$

Programación Dinámica – Algoritmo de Floyd

Resuelve el problema de encontrar todos los caminos más cortos entre cada par de vértices en un grafo dirigido ponderado con pesos no negativos o negativos. A diferencia del algoritmo de Dijkstra, que encuentra los caminos más cortos desde un vértice origen a todos los demás vértices, el algoritmo de Floyd encuentra los caminos más cortos entre todos los pares de vértices en el grafo.

Algoritmo Voraz – Algoritmo de Dijkstra

El algoritmo de Dijkstra calcula la ruta óptima desde un vértice (x, vértice origen) a todos los demás vértices del grafo. Tiene como requisito que no existan longitudes/costes negativos. Se basa en la idea de la relajación de aristas (edge relaxation), en la cual se mantiene información de cual es la mejor distancia conocida de cualquier vértice al origen, y a medida que se va conociendo una distancia mejor desde el vértice j al origen, se actualizan las mejores distancias conocidas a todos los vértices adyacentes a j.

Algoritmo Voraz – Algoritmo de Dijkstra

- Resuelve el problema del camino más corto desde un vértice de origen a todos los demás vértices en un grafo ponderado y dirigido.
- Utiliza un enfoque voraz al seleccionar el vértice no visitado con la distancia mínima conocida al vértice de origen en cada paso.
- A medida que avanza, actualiza las distancias mínimas conocidas de los vértices vecinos y continúa explorando el grafo hasta que se hayan visitado todos los vértices alcanzables.

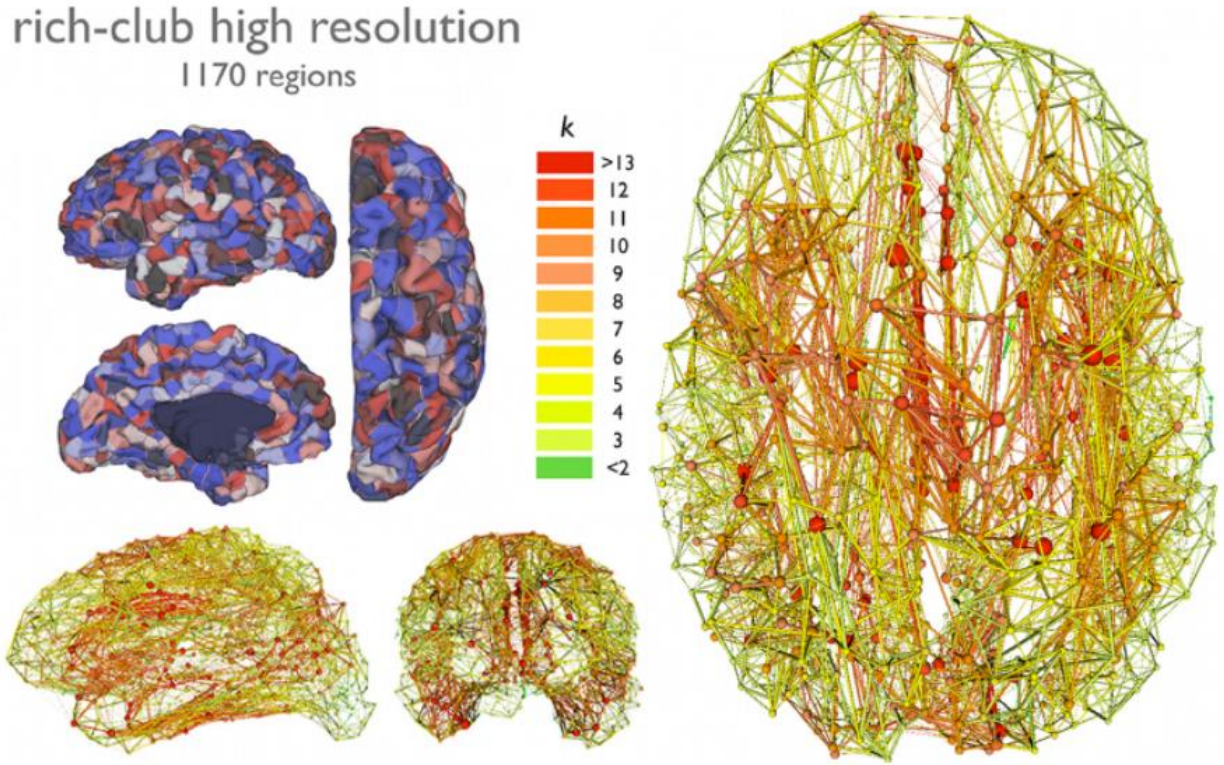
Recorrido

- **Búsqueda en profundidad (DFS):** Visita los nodos lo más profundamente posible a lo largo de cada rama antes de retroceder.
- **Búsqueda en amplitud (BFS):** Visita los nodos vecinos en el mismo nivel antes de visitar los nodos en niveles inferiores.
- **Recorrido en preorden:** Visita el nodo raíz, luego recursivamente realiza un recorrido en preorden de los subárboles izquierdo y derecho.
- **Recorrido en inorden:** Recorre el subárbol izquierdo, visita el nodo raíz y luego recorre el subárbol derecho.
- **Recorrido en postorden:** Recorre recursivamente los subárboles izquierdo y derecho y luego visita el nodo raíz.
- **Recorrido topológico:** Se utiliza en grafos dirigidos acíclicos (DAG) y visita los nodos de manera que para cada arista dirigida de u a v, el nodo u aparece antes que el nodo v.

Utilización de los grafos

Los campos de utilización de los grafos son muy variados, ya que los vértices pueden representar cualquier elemento (ciudades, aeropuertos, pc's...), y las aristas serían la conexión entre esos elementos (carreteras, pasillos aéreos, redes...). Por lo tanto, los grafos son muy usados en la modelización de sistemas reales.

Utilización de los grafos



Red de mundo pequeño