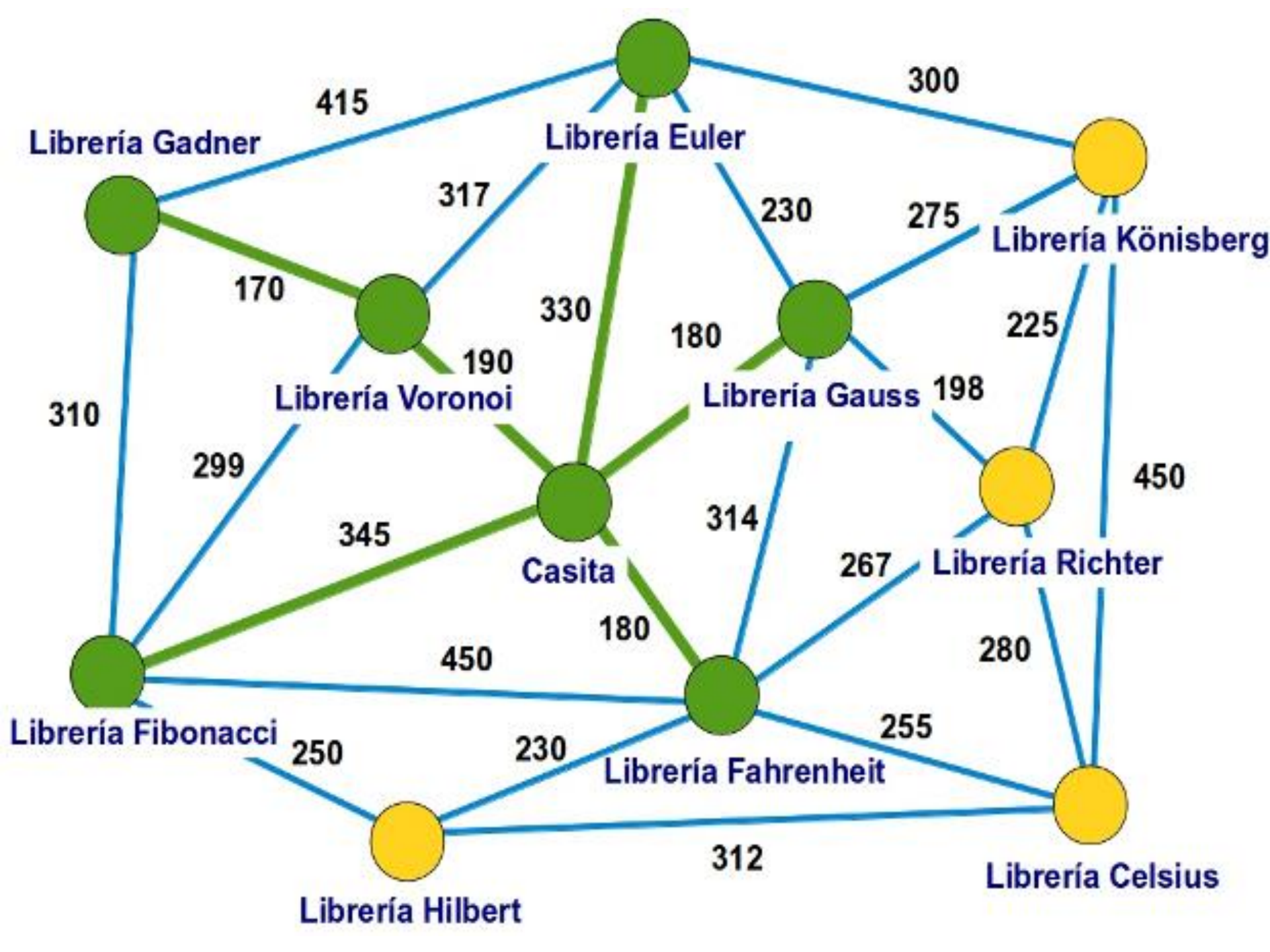
MATERIAL VISUAL CON RESPECTO A GRAFOS

PARTE I

ENTRENAMIENTO EN LÍNEA RUMBO A OVI 2020

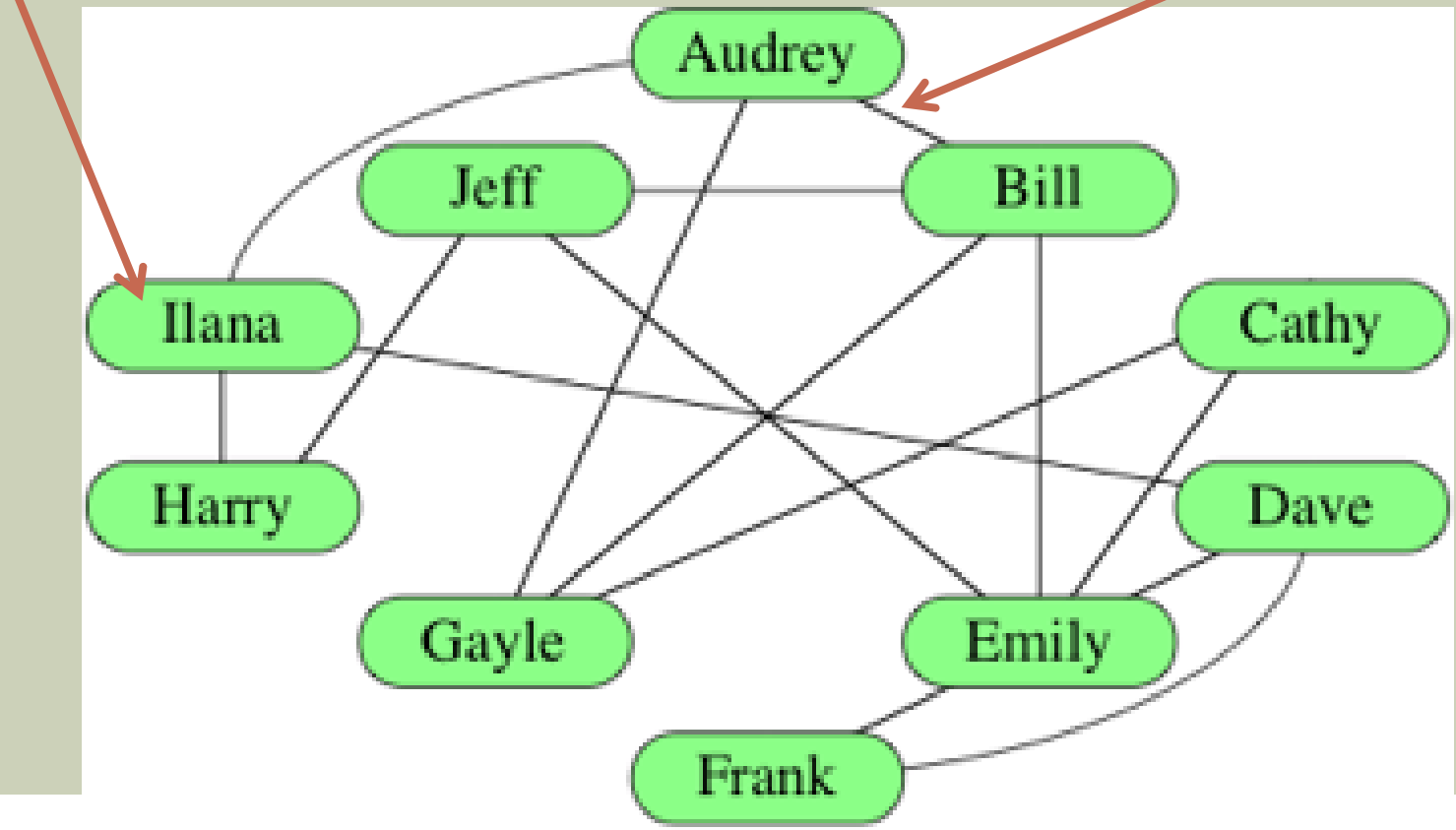
TEMARIO

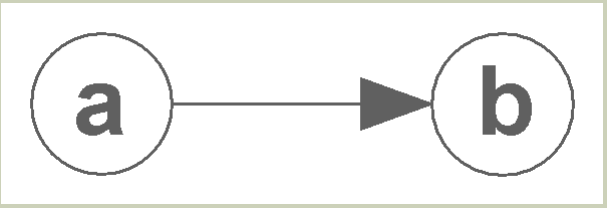
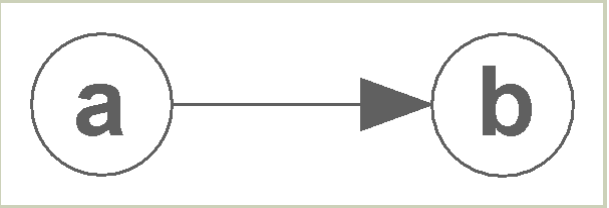
- Grafo
- Árbol
- ¿Cómo hacer un Árbol?
- ¿Cómo hacer un Grafo?
- Ventajas & Desventajas de la Matriz de Adyacencia
- Ventajas & Desventajas de la Lista de Adyacencia
- Leer e Imprimir un Grafo No Dirigido No Ponderado
- Leer e Imprimir un Grafo Dirigido No Ponderado
- Leer e Imprimir un Grafo No Dirigido Ponderado
- Leer e Imprimir un Grafo Dirigido Ponderado
- DFS en un Grafo
- BFS en un Grafo
  - DATOS CURIOSOS
- **EXTRAS**
  - Notación Polaca & Árboles en Pre-Orden
  - Recorrido Pre-Orden, In-Orden, Post-Orden

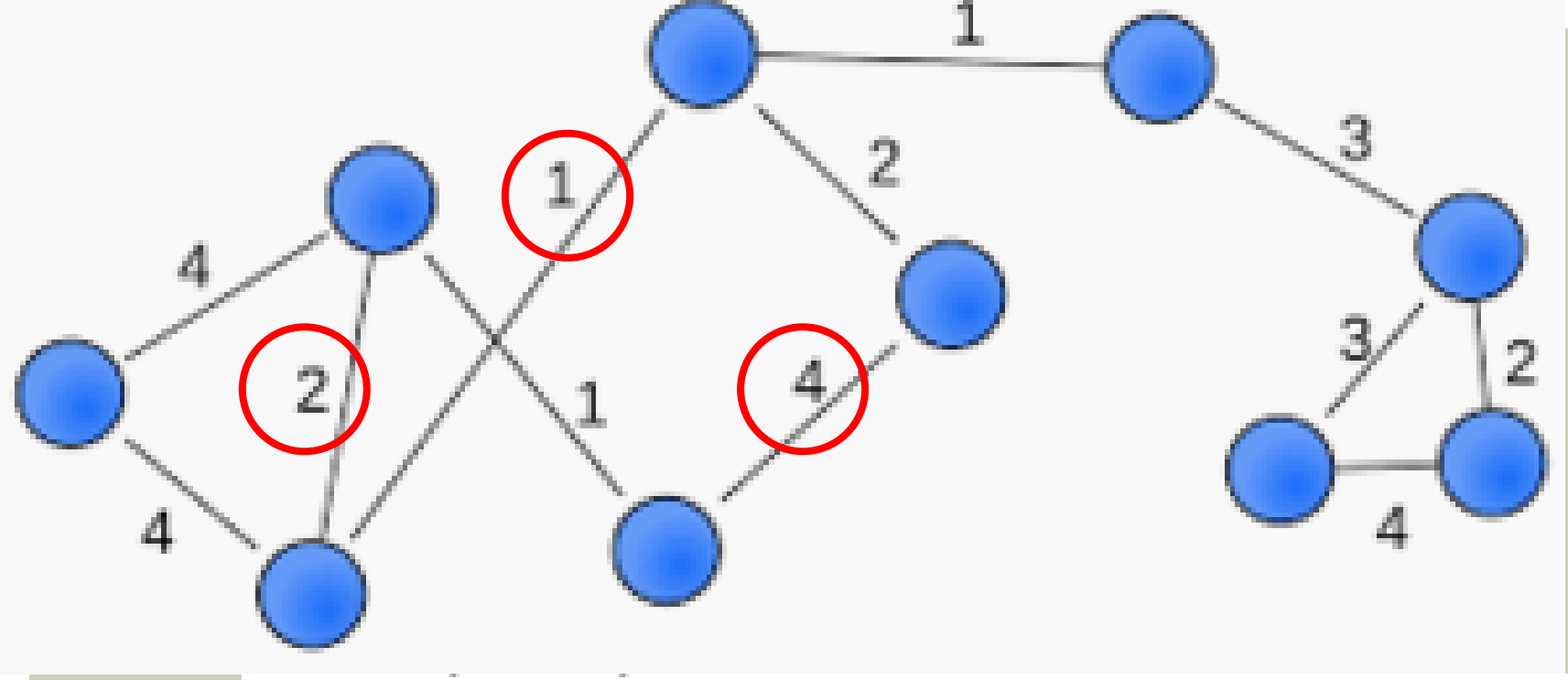

Librería Euler
Librería Gadner
Librería Könisberg
Librería Voronoi
Librería Gauss
Casita
Librería Richter
Librería Fibonacci
Librería Fahrenheit
Librería Hilbert
Librería Celsius
415
300
317
230
275
170
330
180
225
190
198
310
299
450
345
314
267
180
280
450
255
250
230
312

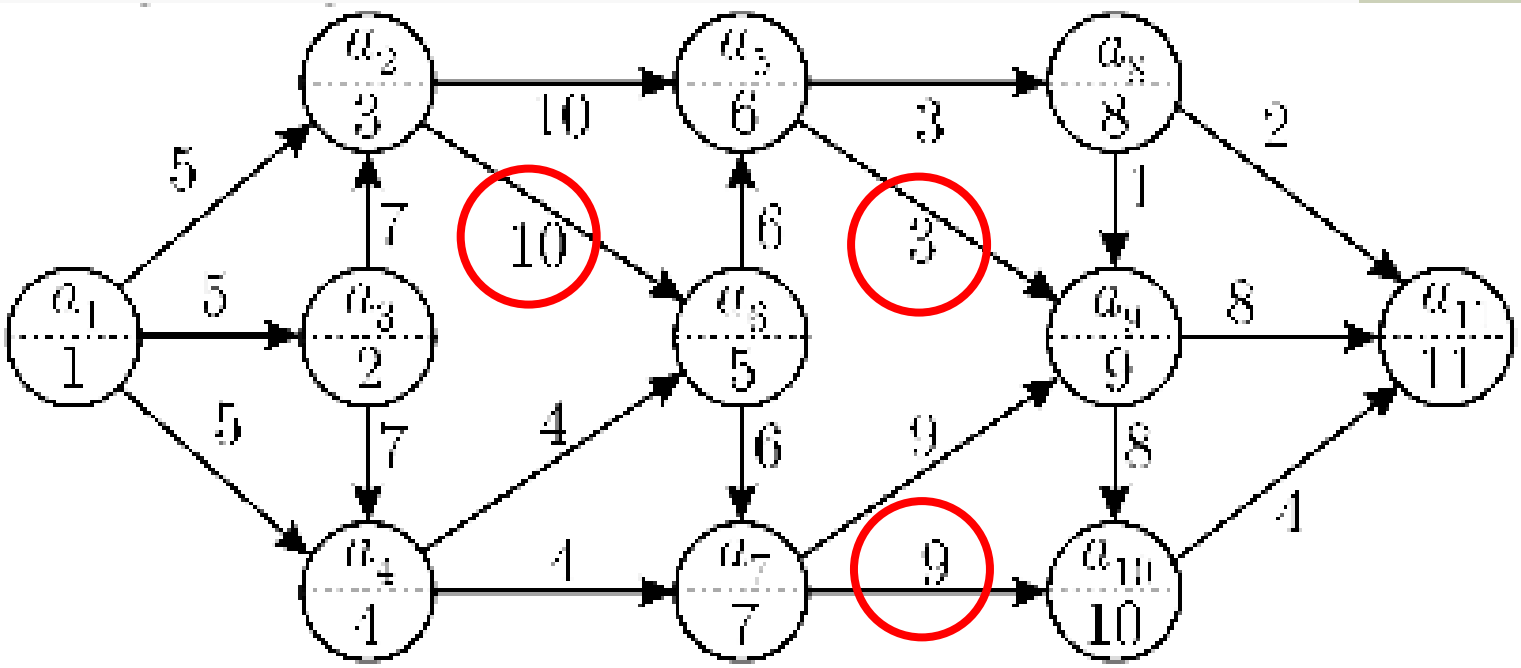
GRAFO

- Método por el cual podemos representar la relación de un conjunto de datos. Su unidad funcional es el nodo, el cual va conectado a través de aristas.



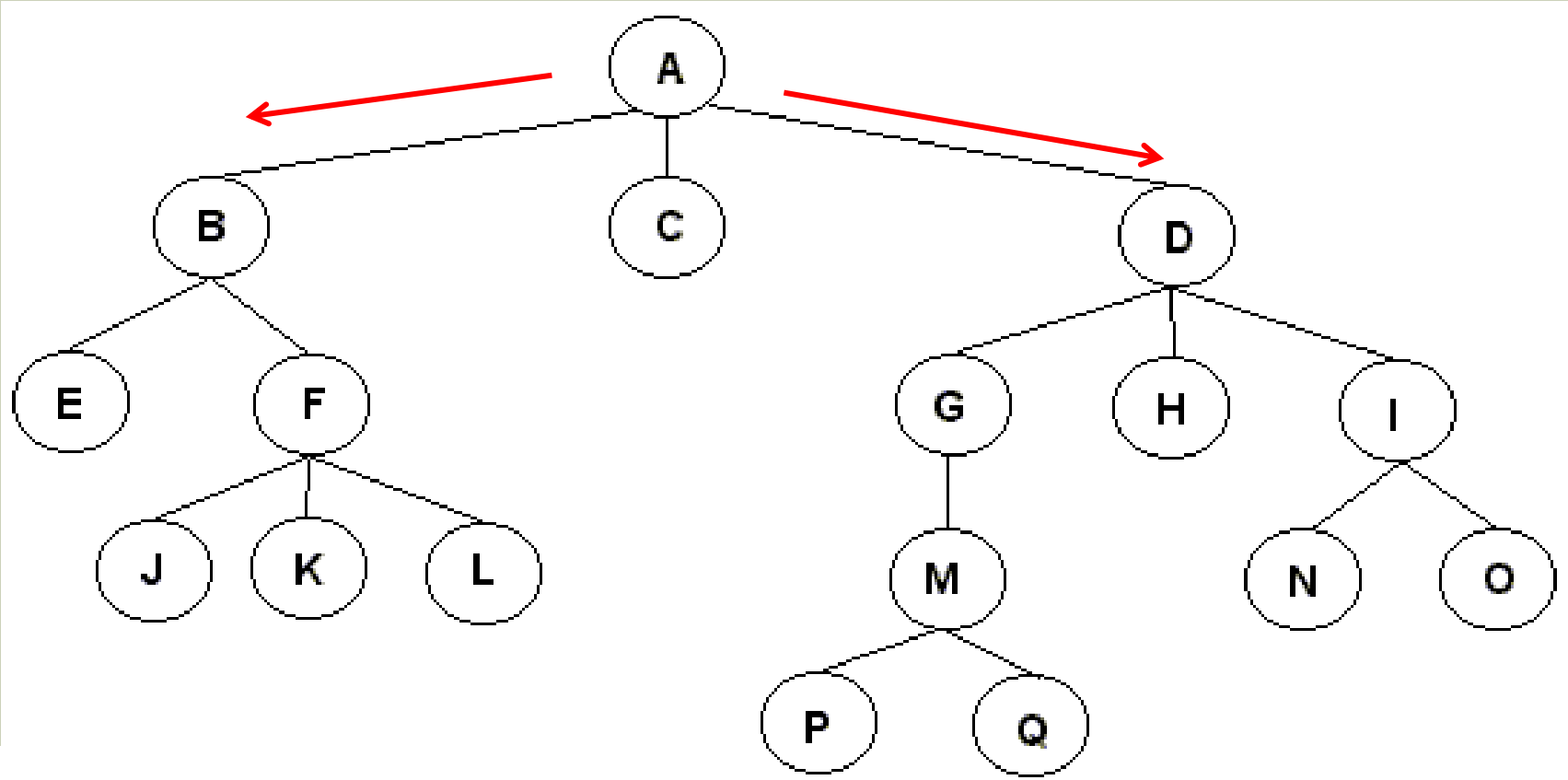
	DADO a, b	
GRAFO NO DIRIGIDO	‘a’ alberga a ‘b’ ‘b’ alberga a ‘a’	
GRAFO DIRIGIDO	‘a’ alberga a ‘b’	
GRAFO MIXTO	Puede presentar ambos casos anteriormente mencionados sin discriminación	 

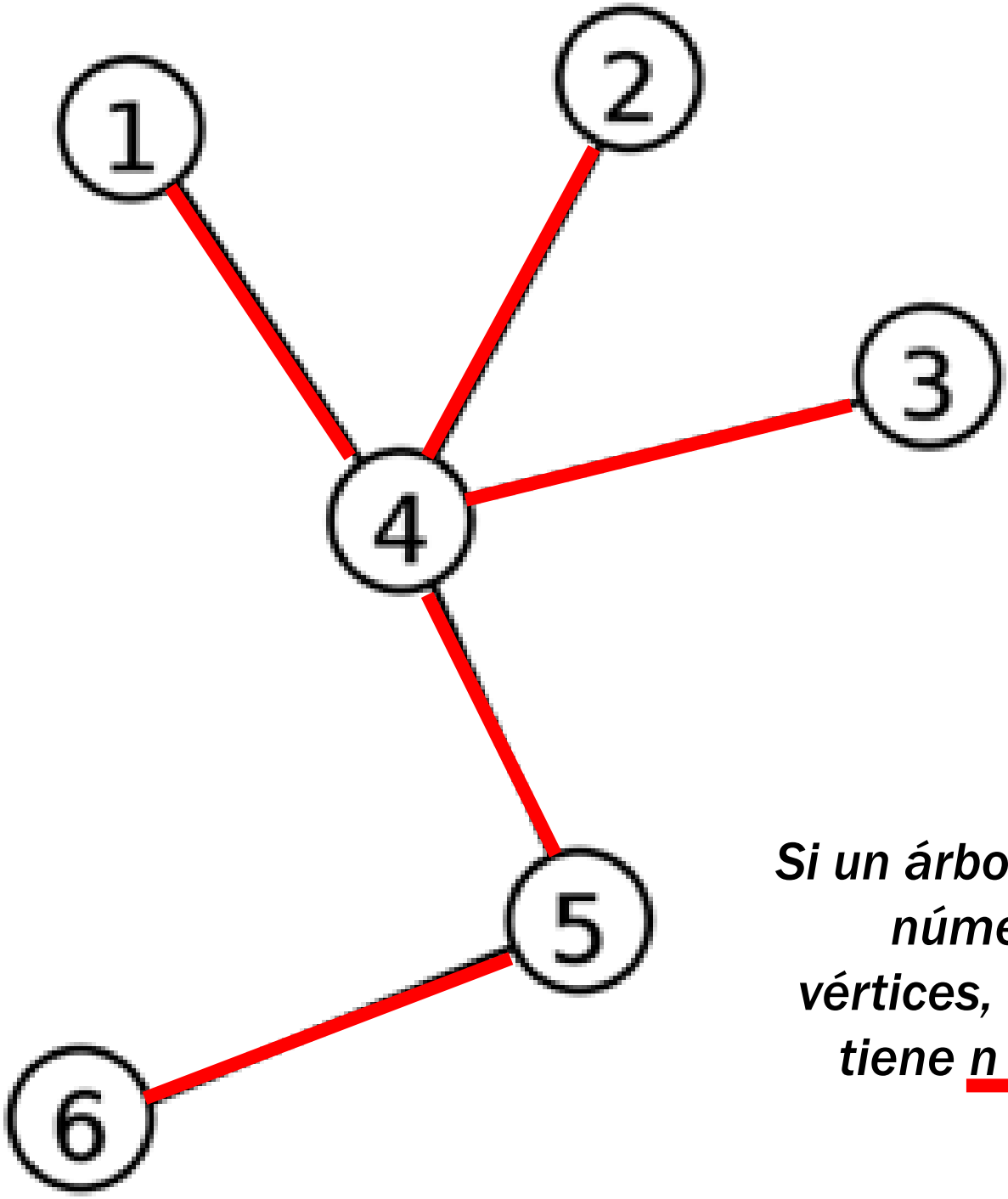

1
2
3
4
1
2
3
4
1
2
3
4


a_1 1
a_2 3
a_3 2
a_4 4
a_5 5
a_6 6
a_7 7
a_8 8
a_9 9
a_{10} 10
a_{11} 11
5
5
5
7
7
4
10
10
6
6
4
3
3
1
9
9
2
8
8
1

ÁRBOL

- Grafo donde dos vértices, A & B tienen solamente un camino de recorrido, además dichos vértices son adyacentes.

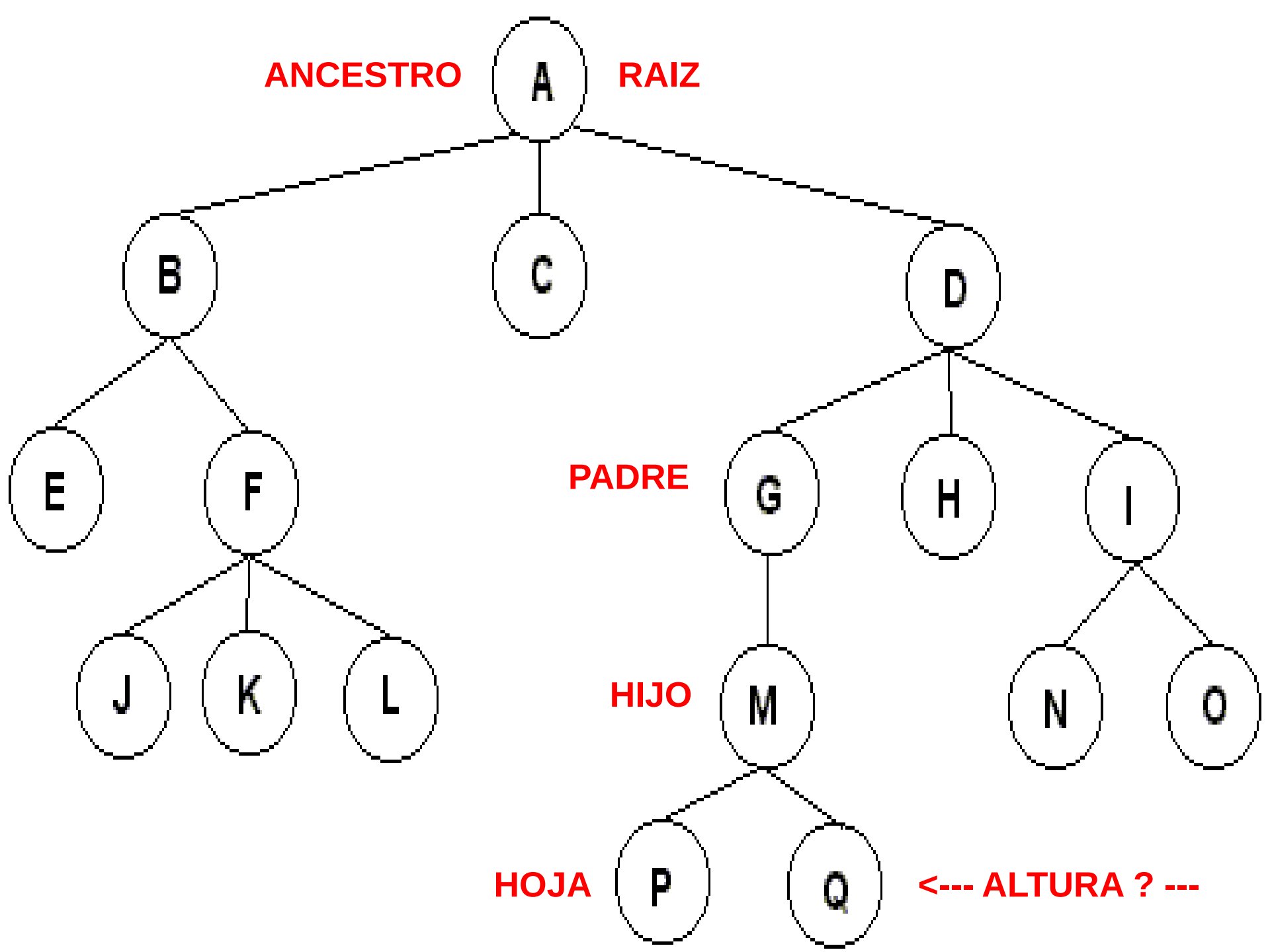




Si un árbol G tiene un número finito de vértices, n, entonces tiene n – 1 aristas.

- **El grafo debe ser conexo y carecer de ciclos**
- **Si al quitarle una arista deja de ser conexo**
- **Dos vértices están conectados por un único camino simple**

- **Raíz:** El nodo superior de un árbol.
- **Hijo:** Un nodo conectado directamente con otro cuando se aleja de la raíz.
- **Padre:** La noción inversa de *hijo*.
- **Ancestro:** Un nodo accesible por ascenso repetido de hijo a padre.
- **Hoja:** Un nodo sin hijos
- **Altura de un árbol:** La altura de un árbol es la altura de su nodo raíz.

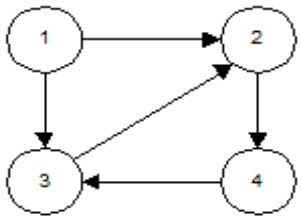

ANCESTRO
A
RAIZ
B
C
D
E
F
PADRE
G
H
I
J
K
L
HIJO
M
N
O
HOJA
P
Q
<--- ALTURA ? ---

¿CÓMO HACER UN ÁRBOL?

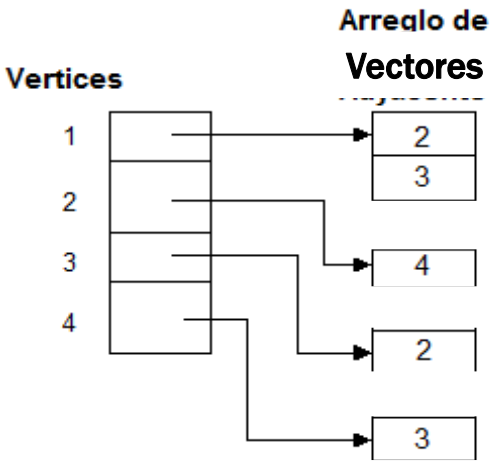
```
#include <vector>
struct dato
{
    int hijo_izquierdo, hijo_derecho;
};

vector <dato> arbol;
```

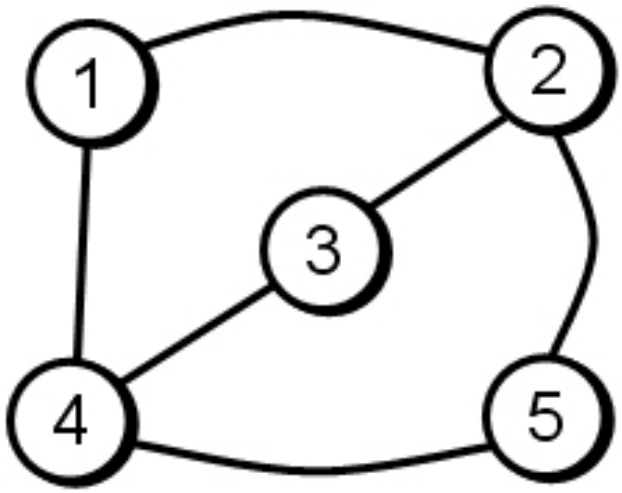
1		2		3		...		N	
Hijo_izquierdo	Hijo_derecho	Hijo_izquierdo	Hijo_derecho	Hijo_izquierdo	Hijo_derecho	Hijo_izquierdo	Hijo_derecho	Hijo_izquierdo	Hijo_derecho

¿CÓMO HACER UN GRAFO?

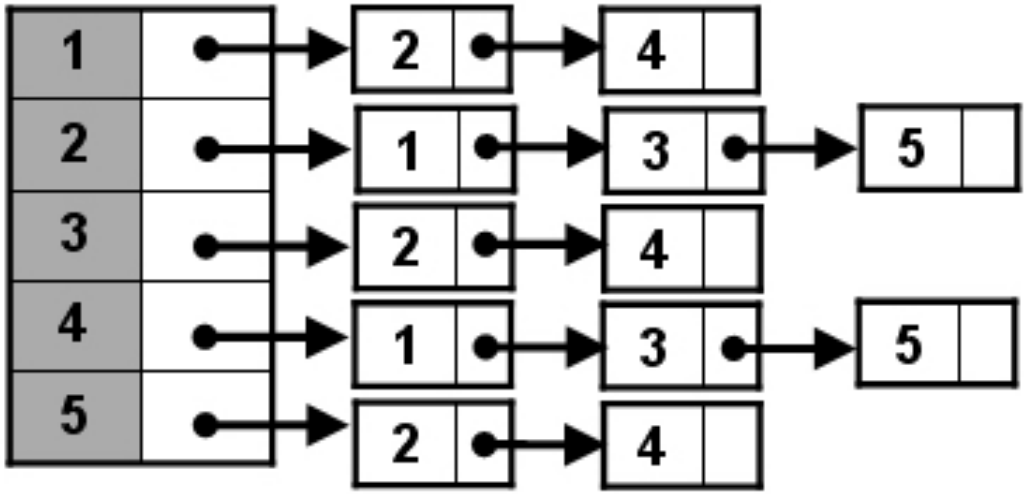


Grafos





M	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	1
3	0	1	0	1	0
4	1	0	1	0	1
5	0	1	0	1	0

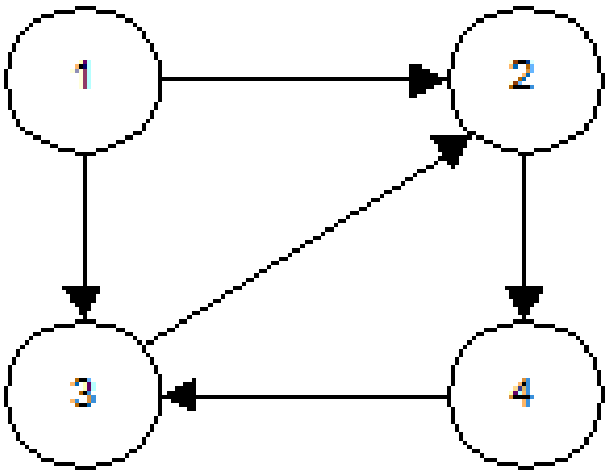


MATRIZ DE ADYACENCIA

VENTAJAS	DESVENTAJAS
Es fácil determinar si existe o no un arco o enlace, solo se debe posicionar en la matriz.	Se requiere un almacenamiento |v|*|v|. Es decir O(n2).
	Solo al leer o examinar la matriz puede llevar un tiempo de O(n2).

LISTA DE ADYACENCIA

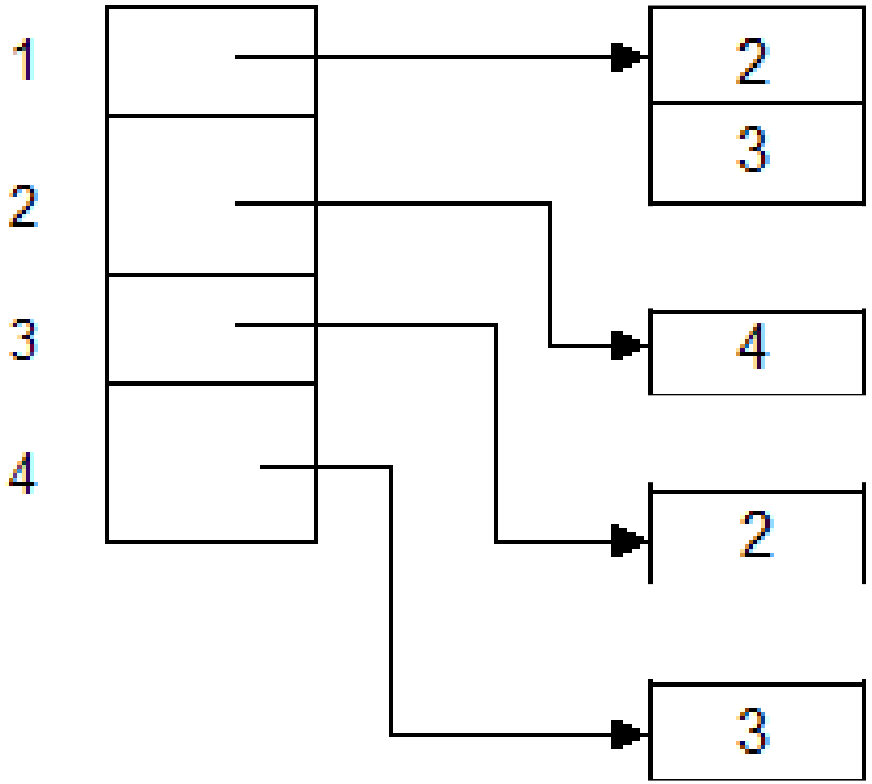
VENTAJAS	DESVENTAJAS
Requiere un espacio proporcional a la suma del número de vértices más el número de enlaces(arcos).	Puede llevar un tiempo O(n) determinar si existe un arco del vértice i al vértice j.



Grafos

Arreglo de Vectores

Vertices



LEER E IMPRIMIR UN GRAFO NO DIRIGIDO NO PONDERADO

```cpp
vector <int> grafo[1000];
int n, k, x, y;

int main()
{
    cin>>n>>k;
    for(int i=0; i<k; i++)
    {
        cin>>x>>y;
        grafo[x].push_back(y);
        grafo[y].push_back(x);
    }
    for(int i=1; i<=n; i++)
    {
        if(!grafo[i].empty())
        {
            cout<<i<<": ";
            for(int j=0; j<grafo[i].size(); j++)
                cout<<grafo[i][j]<<" ";
            cout<<"\n";
        }
    }
    return 0;
}
```

```
7 8
1 2
1 3
1 4
1 5
5 6
5 7
6 7
6 1
1: 2 3 4 5 6
2: 1
3: 1
4: 1
5: 1 6 7
6: 5 7 1
7: 5 6

Process returned
Press any key to
```

LEER E IMPRIMIR UN GRAFO DIRIGIDO NO PONDERADO

```
24  vector <int> grafo[1000];
25  int n, k, x, y;
26
27  int main()
28  {
29      cin>>n>>k;
30      for(int i=0; i<k; i++)
31      {
32          cin>>x>>y;
33          grafo[x].push_back(y);
34          ///grafo[y].push_back(x);
35      }
37      for(int i=1; i<=n; i++)
38      {
39          if(!grafo[i].empty())
40          {
41              cout<<i<<": ";
42              for(int j=0; j<grafo[i].size(); j++)
43                  cout<<grafo[i][j]<<" ";
44              cout<<"\n";
45          }
46      }
47      return 0;
48  }
```

```
7 8
1 2
1 3
1 4
1 5
5 6
5 7
6 7
6 1
1: 2 3 4 5
5: 6 7
6: 7 1

Process returned
Press any key to
_
```

LEER E IMPRIMIR UN GRAFO NO DIRIGIDO PONDERADO

```cpp
vector < pair <int, int> > grafo[1000];
int n, k, x, y, p;

int main()
{
    cin>>n>>k;
    for(int i=0; i<k; i++)
    {
        cin>>x>>y>>p;
        grafo[x].push_back(make_pair(y, p));
        grafo[y].push_back(make_pair(x, p));
    }
   for(int i=1; i<=n; i++)
   {
       if(!grafo[i].empty())
       {
           cout<<"Nodo "<<i<<":\n";
           for(int j=0; j<grafo[i].size(); j++)
               cout<<grafo[i][j].first<<" con peso de: "<<grafo[i][j].second<<"\n";
           cout<<"\n";
       }
   }
   return 0;
}
```

```
7 8
1 2 3
1 3 8
1 4 5
1 5 2
5 6 4
5 7 9
6 7 2
6 1 10
Nodo 1:
2 con peso de: 3
3 con peso de: 8
4 con peso de: 5
5 con peso de: 2
6 con peso de: 10

Nodo 2:
1 con peso de: 3

Nodo 3:
1 con peso de: 8

Nodo 4:
1 con peso de: 5

Nodo 5:
1 con peso de: 2
6 con peso de: 4
7 con peso de: 9

Nodo 6:
5 con peso de: 4
7 con peso de: 2
1 con peso de: 10

Nodo 7:
5 con peso de: 9
6 con peso de: 2


Process returned 0 (0x0)   execution time : 4.422 s
Press any key to continue.
```

LEER E IMPRIMIR UN GRAFO DIRIGIDO PONDERADO

```cpp
vector < pair <int, int> > grafo[1000];
int n, k, x, y, p;

int main()
{
    cin>>n>>k;
    for(int i=0; i<k; i++)
    {
        cin>>x>>y>>p;
        grafo[x].push_back(make_pair(y, p));
        ///grafo[y].push_back(make_pair(x, p));
    }
    for(int i=1; i<=n; i++)
    {
        if(!grafo[i].empty())
        {
            cout<<"Nodo "<<i<<":\n";
            for(int j=0; j<grafo[i].size(); j++)
                cout<<grafo[i][j].first<<" con peso de: "<<grafo[i][j].second<<"\n";
            cout<<"\n";
        }
    }
    return 0;
}
```

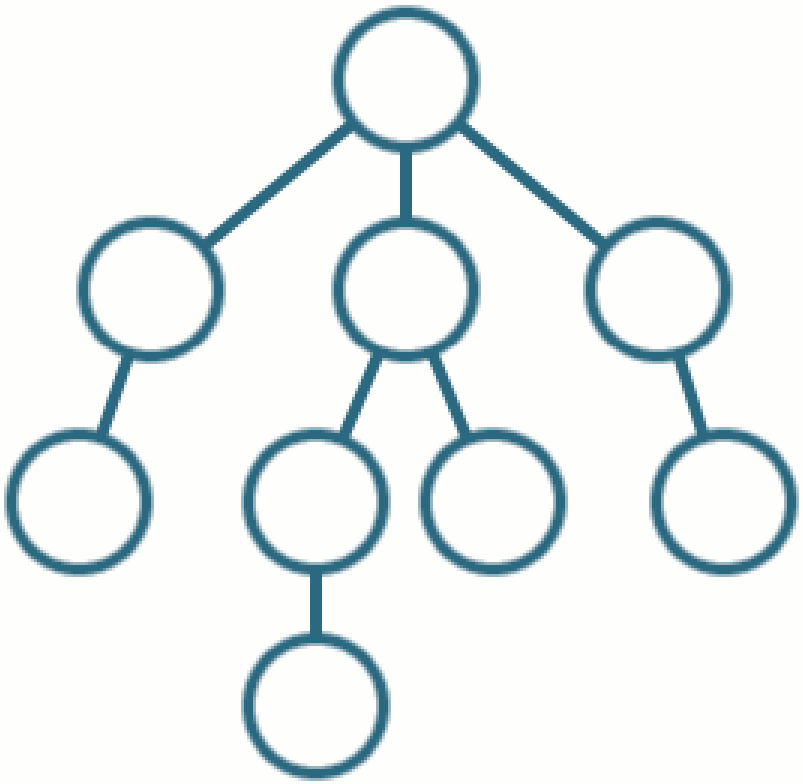
```
7 8
1 2 3
1 3 8
1 4 5
1 5 2
5 6 4
5 7 9
6 7 2
6 1 10
Nodo 1:
2 con peso de: 3
3 con peso de: 8
4 con peso de: 5
5 con peso de: 2

Nodo 5:
6 con peso de: 4
7 con peso de: 9

Nodo 6:
7 con peso de: 2
1 con peso de: 10


Process returned 0 (0x0
Press any key to contin
```

DFS EN UN GRAFO

```
 7  vector <int> vec[100001];
 8  //queue <int> cola;
 9  int n, v, h;
10  bool visitado[100001];
11
12  void dfs(int a)
13  {
14      printf("%d ", a);
15      visitado[a]=true;
16      for(int l=0; l<vec[a].size(); l++)
17      {
18          int h=vec[a][l];
19          if(visitado[h]==false)
20              dfs(h);
21      }
22  }
```

```
69      ///dfs
70      for(int i=0; i<n; i++)
71          if(!visitado[i])
72              dfs(i);
```

DFS

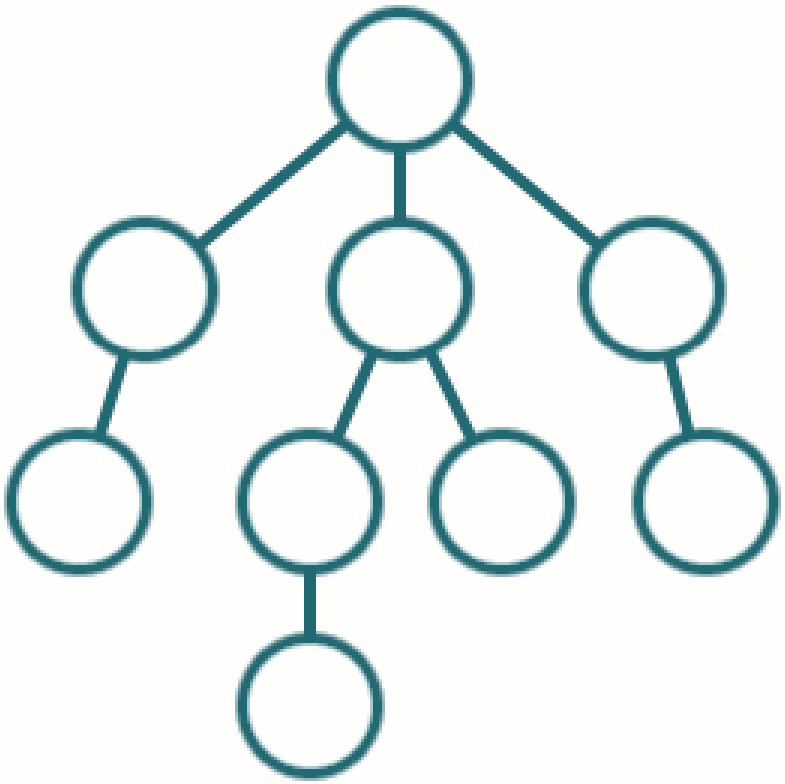


BFS EN UN GRAFO

```
24  void bfs(int a)
25  {
26      cola.push(a);
27      visitado[a]=true;
28      while(!cola.empty())
29      {
30          a=cola.front();
31          cola.pop();
32          printf("%d ", a);
33          for(int y=0; y<vec[a].size(); y++)
34          {
35              int u=vec[a][y];
36              if(!visitado[u])
37              {
38                  visitado[u]=true;
39                  cola.push(u);
40              }
41          }
42      }
43  }
```

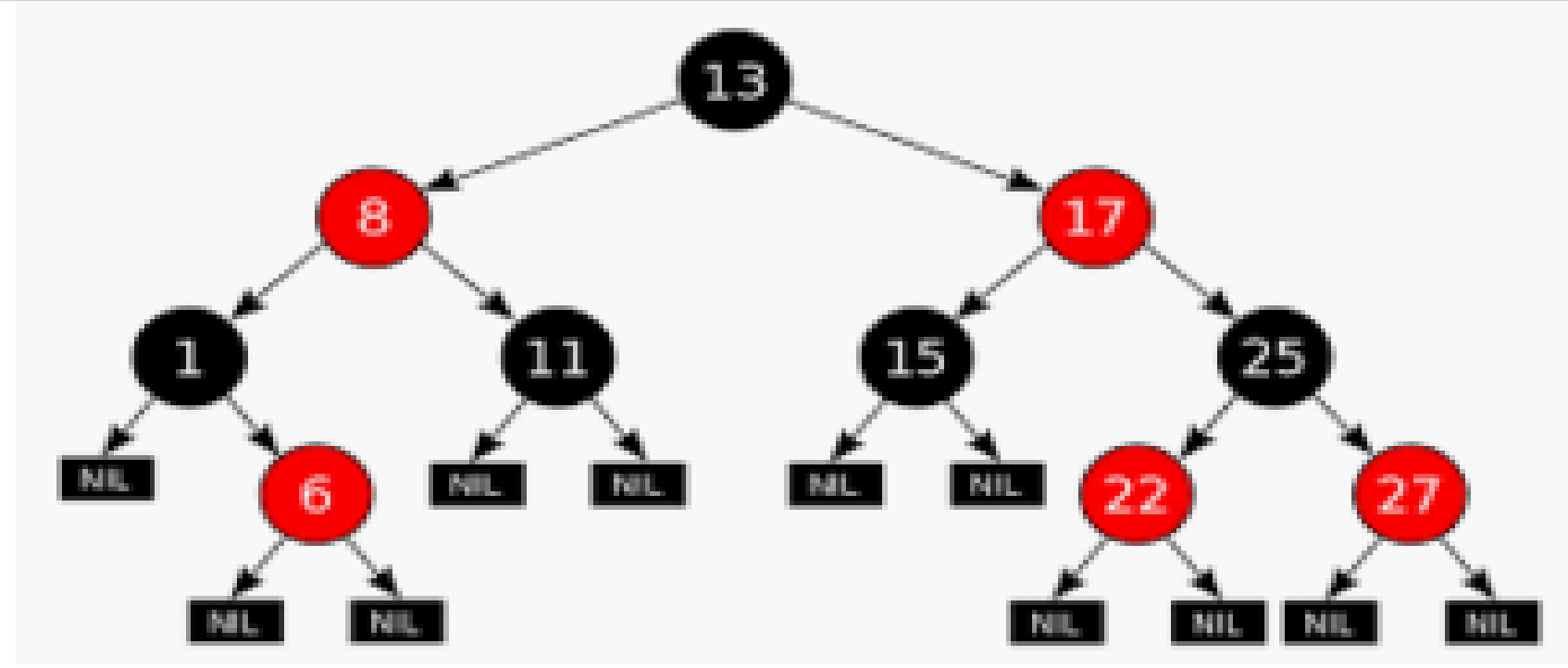
```
59      //bfs
60      for(int i=0; i<n; i++)
61          if(!visitado[i])
62              bfs(i);
```

BFS

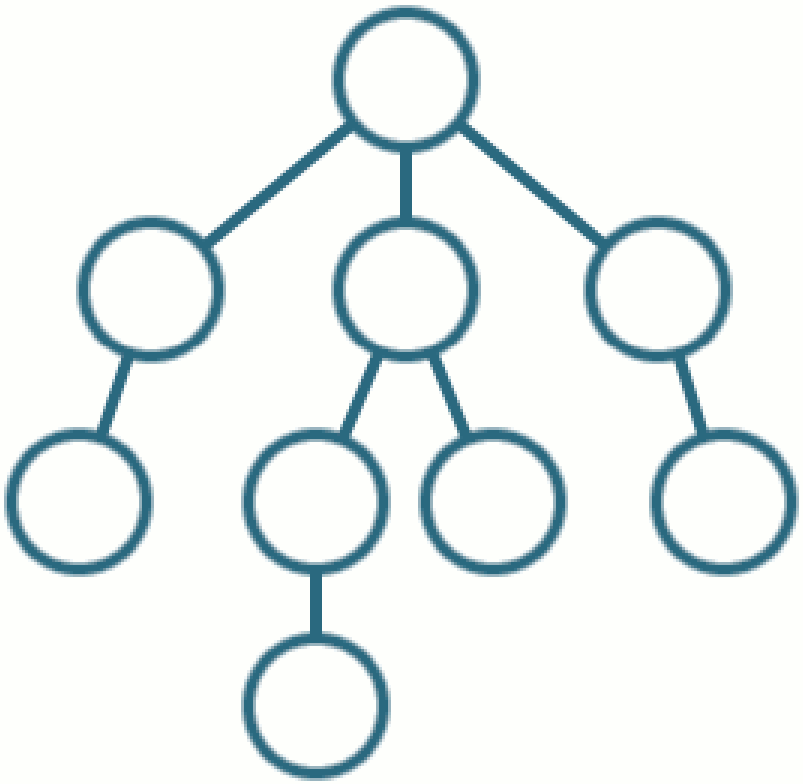


DATOS CURIOSOS DE BFS

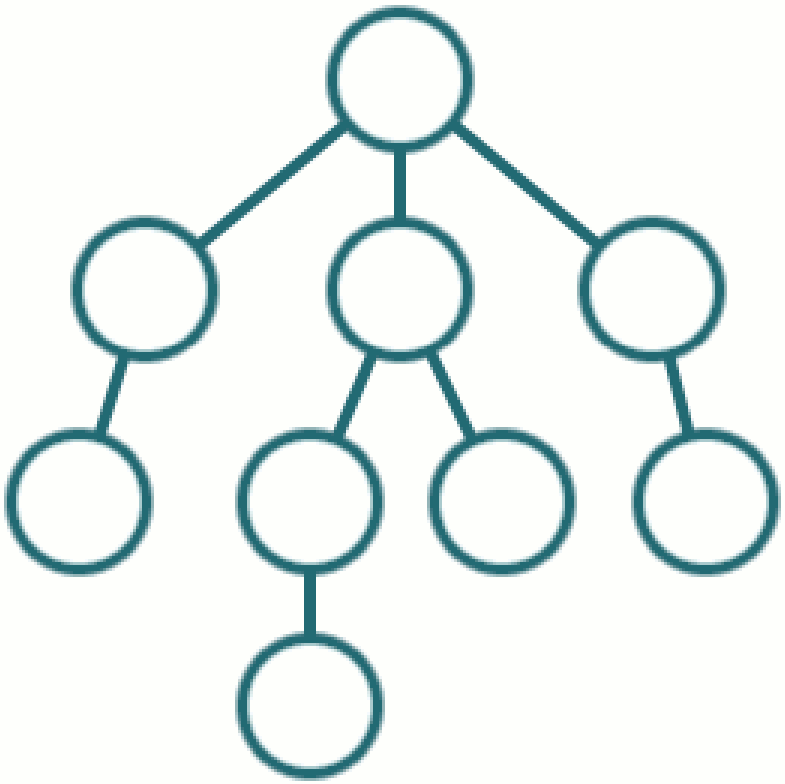
- Para encontrar los nodos que están en una distancia D de la raíz, la BFS requiere de $O(B^{D+1})$, donde B es el factor de ramificación del grafo.
- **El Factor de ramificación es el número de nodos hijos en cada nodo.**



DFS



BFS



MATERIAL CREADO PARA EL ENTRENAMIENTO EN LÍNEA P/OVI 2020

19/ABRIL

Este material puede usarse para explicar los temas ya establecidos.

*Fabricio Cruz López

@MrKristarIx07 – GitHub

@marbasz - Twitter

BUEN VIAJE!!!