

Asistencia 70% para aprobar CURSADA (no más de 3 faltas!)

Cronograma:

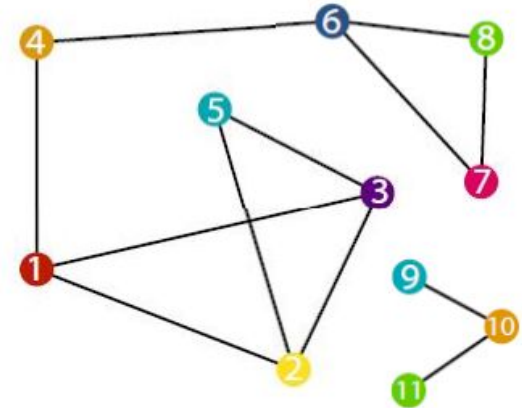
- **Semana 4/6: TP N°6 - Grafos (Semana 1 de 2)**
- **Semana 11/6: TP N°6 - Grafos (Semana 2 de 2)**

- **Sábado 23/6 Parcial Módulo 2**

Grafos

Relación con árboles

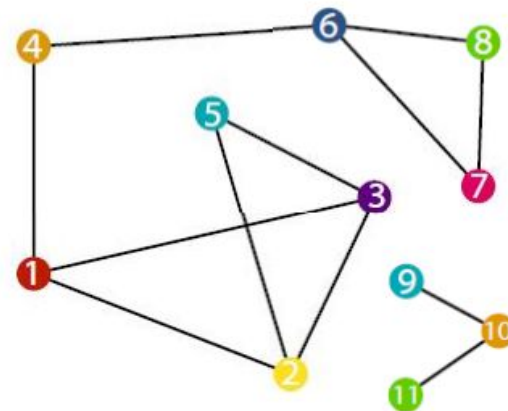
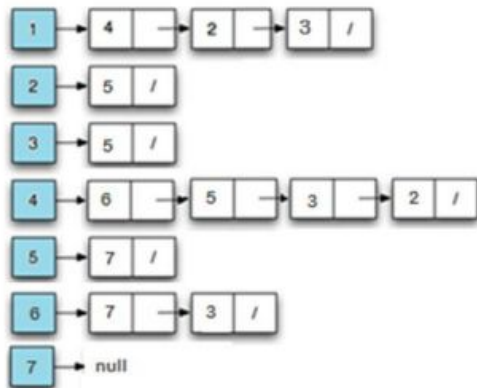
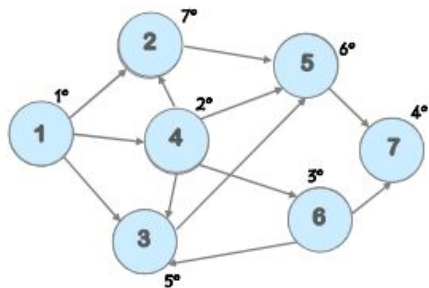
- no tiene raíz
- no hay nodos, en grafos tenemos vértices
- no hay hijos, vértices “adyacentes”
- Las relaciones entre vértices se llaman aristas y pueden tener un valor entero asociado llamado “costo”
- Las aristas pueden tener dirección o ser bidireccionales
- Los vértices no tienen que estar todos conectados (ver dibujo)
- ¿Un árbol es un grafo, un grafo no es un árbol ?



Recorridos en Grafos

DFS → recorrido en profundidad, recursivo
(similar a los recorridos en profundidad en árboles)

BFS → recorrido en amplitud, iterativo, se utiliza una Cola
(similar al recorrido por niveles en árboles)



DFS - Algoritmo

```
public class Recorridos<T> {  
    public void dfs(Grafo<T> grafo){  
        boolean[] marca = new boolean[grafo.listaDeVertices().tamanio()+1];  
        for(int i=1; i<=grafo.listaDeVertices().tamanio();i++){  
            if (!marca[i])  
                this.dfs(i, grafo, marca);  
        }  
    }  
    private void dfs(int i,Grafo<T> grafo, boolean[] marca){  
        marca[i] = true;  
        Vertice<T> v = grafo.listaDeVertices().elemento(i);  
        System.out.println(v);  
        ListaGenerica<Arista<T>> ady = grafo.listaDeAdyacentes(v);  
        ady.comenzar();  
        while(!ady.fin()){  
            int j = ady.proximo().getVerticeDestino().getPosicion();  
            if(!marca[j]){  
                this.dfs(j, grafo, marca);  
            }  
        }  
    }  
}
```

DFS - Algoritmo

```
public class Recorridos<T> {
```

```
    public void dfs(Grafo<T> grafo){
```

```
        boolean[] marca = new boolean[grafo.listaDeVertices().tamanio()+1];
```

```
        for(int i=1; i<=grafo.listaDeVertices().tamanio();i++){
```

```
            if (!marca[i])
```

```
                this.dfs(i, grafo, marca);
```

```
        }
```

```
    }
```

```
    private void dfs(int i,Grafo<T> grafo, boolean[] marca){
```

```
        marca[i] = true;
```

```
        Vertice<T> v = grafo.listaDeVertices().elemento(i);
```

```
        System.out.println(v);
```

```
        ListaGenerica<Arista<T>> ady = grafo.listaDeAdyacentes(v);
```

```
        ady.comenzar();
```

```
        while(!ady.fin()){
```

```
            int j = ady.proximo().getVerticeDestino().getPosicion();
```

```
            if(!marca[j]){
```

```
                this.dfs(j, grafo, marca);
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

Modificaciones



Disparar el recorrido para un determinado vértice dado o que se tenga que buscar el vértice dado el dato.

Se podría disparar con otros parámetros, ej vértice destino.

DFS - Algoritmo

```
public class Recorridos<T> {
```

```
    public void dfs(Grafo<T> grafo){
```

```
        boolean[] marca = new boolean[grafo.listaDeVertices().tamanio()+1];
```

```
        for(int i=1; i<=grafo.listaDeVertices().tamanio();i++){
```

```
            if (!marca[i])
```

```
                this.dfs(i, grafo, marca);
```

```
        }
```

```
    }
```

```
    private void dfs(int i,Grafo<T> grafo, boolean[] marca){
```

```
        marca[i] = true;
```

```
        Vertice<T> v = grafo.listaDeVertices().elemento(i);
```

```
        System.out.println(v);
```

```
        ListaGenerica<Arista<T>> ady = grafo.listaDeAdyacentes(v);
```

```
        ady.comenzar();
```

```
        while(!ady.fin()){
```

```
            int j = ady.proximo().getVerticeDestino().getPosicion();
```

```
            if(!marca[j]){
```

```
                this.dfs(j, grafo, marca);
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

Modificaciones

Disparar el recorrido para un determinado vértice dado o que se tenga que buscar el vértice dado el dato.

Se podría disparar con otros parámetros, ej vértice destino.

Se procesa el dato actual. Por ej, se podría guardar en una lista, sumar o contabilizar alguna variable

DFS - Algoritmo

```
public class Recorridos<T> {
```

```
    public void dfs(Grafo<T> grafo){
```

```
        boolean[] marca = new boolean[grafo.listaDeVertices().tamanio()+1];
```

```
        for(int i=1; i<=grafo.listaDeVertices().tamanio();i++){
```

```
            if (!marca[i])
```

```
                this.dfs(i, grafo, marca);
```

```
        }
```

```
    }
```

```
    private void dfs(int i,Grafo<T> grafo, boolean[] marca){
```

```
        marca[i] = true;
```

```
        Vertice<T> v = grafo.listaDeVertices().elemento(i);
```

```
        System.out.println(v);
```

```
        ListaGenerica<Arista<T>> ady = grafo.listaDeAdyacentes(v);
```

```
        ady.comenzar();
```

```
        while(!ady.fin()){
```

```
            int j = ady.proximo().getVerticeDestino().getPosicion();
```

```
            if(!marca[j]){
```

```
                this.dfs(j, grafo, marca);
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

Modificaciones

Disparar el recorrido para un determinado vértice dado o que se tenga que buscar el vértice dado el dato.

Se podría disparar con otros parámetros, ej vértice destino.

Se procesa el dato actual. Por ej, se podría guardar en una lista, sumar o contabilizar alguna variable

Antes de llamar recursivamente, se podría verificar que el vértice además de no estar visitado, ver si cumple con alguna/s condición/es

BFS - Algoritmo

```
public class Recorridos<T> {  
    public void bfs(Grafo<T> grafo) {  
        boolean[] marca = new boolean[grafo.listaDeVertices().tamanio()+1];  
        for(int i=1; i<=grafo.listaDeVertices().tamanio();i++){  
            if (!marca[i])  
                this.bfs(i, grafo, marca); //las listas empiezan en la pos 1  
        }  
    }  
    private void bfs (int i, Grafo<T> grafo, boolean[] marca) {  
        ...  
    }  
}
```


BFS - Algoritmo

```
public class Recorridos<T> {  
    ...  
    private void bfs(int i, Grafo<T> grafo, boolean[] marca) {  
        ListaGenerica<Arista<T>> ady = null;  
        ColaGenerica<Vertice<T>> q = new ColaGenerica<Vertice<T>>();  
        q.encolar(grafo.listaDeVertices().elemento(i));  
        marca[i] = true;  
        while (!q.esVacia()) {  
            Vertice<T> v = q.desencolar();  
            System.out.println(v);  
            ady = grafo.listaDeAdyacentes(v);  
            ady.comenzar();  
            while (!ady.fin()) {  
                Arista<T> arista = ady.proximo();  
                int j = arista.getVerticeDestino().getPosicion();  
                if (!marca[j]) {  
                    Vertice<T> w = arista.getVerticeDestino();  
                    marca[j] = true;  
                    q.encolar(w);  
                }  
            }  
        }  
    }  
}
```

BFS - Algoritmo

```
public class Recorridos<T> {
```

```
    ...
```

```
    private void bfs(int i, Grafo<T> grafo, boolean[] marca) {
```

```
        ListaGenerica<Arista<T>> ady = null;
```

```
        ColaGenerica<Vertice<T>> q = new ColaGenerica<Vertice<T>>();
```

```
        q.encolar(grafo.listaDeVertices().elemento(i));
```

```
        marca[i] = true;
```

```
        while (!q.esVacia()) {
```

```
            Vertice<T> v = q.desencolar();
```

```
            System.out.println(v);
```

```
            ady = grafo.listaDeAdyacentes(v);
```

```
            ady.comenzar();
```

```
            while (!ady.fin()) {
```

```
                Arista<T> arista = ady.proximo();
```

```
                int j = arista.getVerticeDestino().getPosicion();
```

```
                if (!marca[j]) {
```

```
                    Vertice<T> w = arista.getVerticeDestino();
```

```
                    marca[j] = true;
```

```
                    q.encolar(w);
```

```
                }
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

Modificaciones



Se procesa el dato actual. Por ej, se podría guardar en una lista, sumar o contabilizar alguna variable

BFS - Algoritmo

```
public class Recorridos<T> {
```

```
    ...
```

```
    private void bfs(int i, Grafo<T> grafo, boolean[] marca) {
```

```
        ListaGenerica<Arista<T>> ady = null;
```

```
        ColaGenerica<Vertice<T>> q = new ColaGenerica<Vertice<T>>();
```

```
        q.encolar(grafo.listaDeVertices().elemento(i));
```

```
        marca[i] = true;
```

```
        while (!q.esVacia()) {
```

```
            Vertice<T> v = q.desencolar();
```

```
            System.out.println(v);
```

```
            ady = grafo.listaDeAdyacentes(v);
```

```
            ady.comenzar();
```

```
            while (!ady.fin()) {
```

```
                Arista<T> arista = ady.proximo();
```

```
                int j = arista.getVerticeDestino().getPosicion();
```

```
                if (!marca[j]) {
```

```
                    Vertice<T> w = arista.getVerticeDestino();
```

```
                    marca[j] = true;
```

```
                    q.encolar(w);
```

```
                }
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

Modificaciones




Se procesa el dato actual. Por ej, se podría guardar en una lista, sumar o contabilizar alguna variable



Antes de encolar, se podría verificar que el vértice además de no estar visitado, ver si cumple con alguna/s condición/es

BFS - Algoritmo

```
public class Recorridos<T> {  
    public void bfs(Grafo<T> grafo) {  
        boolean[] marca = new boolean[grafo.listaDeVertices().tamanio()+1];  
        for(int i=1; i<=grafo.listaDeVertices().tamanio();i++){  
            if (!marca[i])  
                this.bfs(i, grafo, marca); //las listas empiezan en la pos 1  
        }  
    }  
    private void bfs (int i, Grafo<T> grafo, boolean[] marca) {  
        ...  
    }  
}
```



*Disparar el recorrido para un determinado vértice dado o que se tenga que buscar el vértice dado el dato.
Se podría disparar con otros parámetros, ej vértice destino.*