

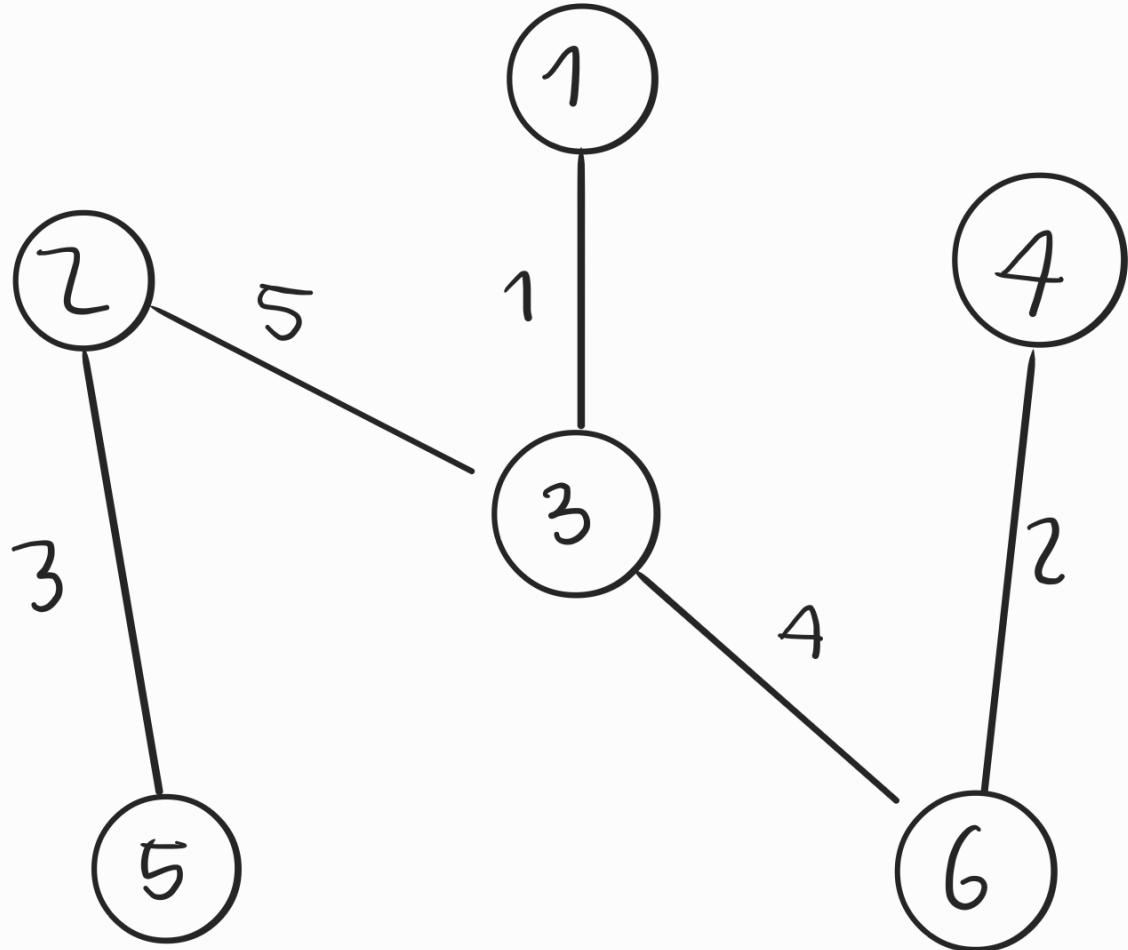
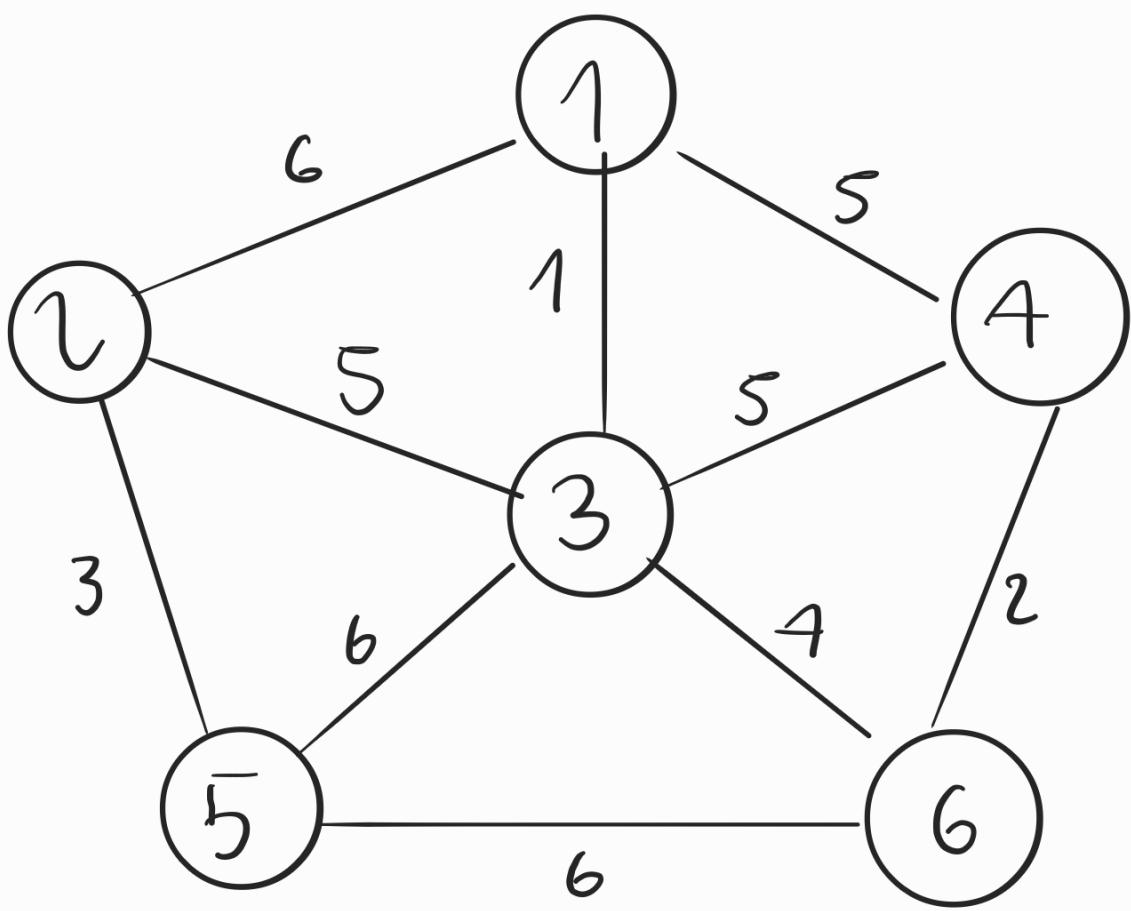
GRAFOS NO DIRIGIDOS

- Se utilizan para representar relaciones entre objetos de datos.
- Si las aristas NO son dirigidas $(V_1, V_2) = (V_2, V_1)$, el grafo es NO DIRIGIDO
- El Grafo CONEXO si todos sus pares de vértices están conectados
- Un grafo No dirigido, conexo y acíclico es considerado como un arbol libre, que si se elige un vértice como raíz se convierte en un arbol ordinario.
Si tiene $n \geq 1$ vértices , tiene $n-1$ aristas

- Para representar grafos dirigidos se utilizar matrices y listas.
La matriz siempre será simétrica.

ARBOL ABARCADOR DE COSTO MINIMO

- Es un arbol libre dentro de G que conecta todos los V
- El costo de ese árbol es la suma de los costos de todas las aristas



ALGORITMO PRIM

- Comienza cuando se le asigna a un conjunto U un valor inicial $\{1\}$. El arbol abarcador "crece" arista por arista.
- En cada paso, localiza la arista mas corta (u,v) que conecta al subconjunto U y $V-U$, y despues agrega v , el vertice en V , a U .
- Esto se repite hasta que $U=V$
↓
el subconjunto tenga todos los vertices

Método T Grafo. Prim (conjunto aristas T)

U: conjuntos de vértices

u, v : vértice;

COMIENZO

T. vaciar;

U. agregar (1);

Mientras $U <> V$ hacer

 elegir una arista (u, v) de costos

 mínimos tal que u está en U

 y v está en $V - U$;

 T. agregar (u, v)

 U. agregar (v)

FIN MIENTRAS

FIN

T GrafoNoDirigido.Prim(): T GrafoNoDirigido

- Ejecuta el algoritmo de Prim y devuelve un nuevo grafo, que es el AAM
- ✓ contiene las etiquetas del G. original
 - Aristas AAM del tipo TAristas es donde se agregan las aristas de costo minimo.
 - tempArista tipo TArista para ir llevando la minima.
 - Colección de vértices "U", "V"

TGrafoNoDirigido.Prim(); TGrafoNoDirigido

U. agregar (V. quitar primero)

Mientras V no vacio hacer

tempArista \leftarrow aristas.buscarMin(V, N)

AristasAAM.insertar(tempArista)

V. quitar (tempArista. etiquetaDestino)

U. agregar (tempArista. etiquetaDestino)

costoPrim \leftarrow costoPrim + tempArista.costo

FIN MIENTRAS

Devolver nuevos TGrafoNoDirigidos(V, AristasAAM)

buscar Min(U,V:colecciones de vértices): Tarista

- Tarista minArista = null
- int minCosto = maxInt
- Para cada U en \mathcal{U}
 - . Para cada V en \mathcal{V}
 - tempArista \leftarrow buscar (U, V)
 - , Si tempArista \neq null
 - Si tempArista. costo < minCosto
 - minArista \leftarrow tempArista
 - minCosto \leftarrow tempArista. costo
 - Fin Si
 - Devolver minArista

KRUSKAL

- Se empieza con un grafo $T = (V, E)$, construido solo por vértices.
- Cada vértice es un componente conexo en sí mismo.
- Para construir componentes cada vez mayores, se agrega la arista de costo mínimo que conecte dos componentes distintos
- Se descarta si conecta dos vértices del mismo componente, ya que formaría un ciclo
- La meta es llegar a que todos los vértices estén en el mismo componente

Método T Grafo . Kruskal;

F conjuntos de aristas;

COM

F. vaciar ;

Repetir

Elegir una arista de costo mínimo tal que no este en F ni haya sido elegida)

Si la arista no conecta dos vértices del mismo componente entonces

F. agregar (arista)

hasta que todos los vértices estén en el mismo componente

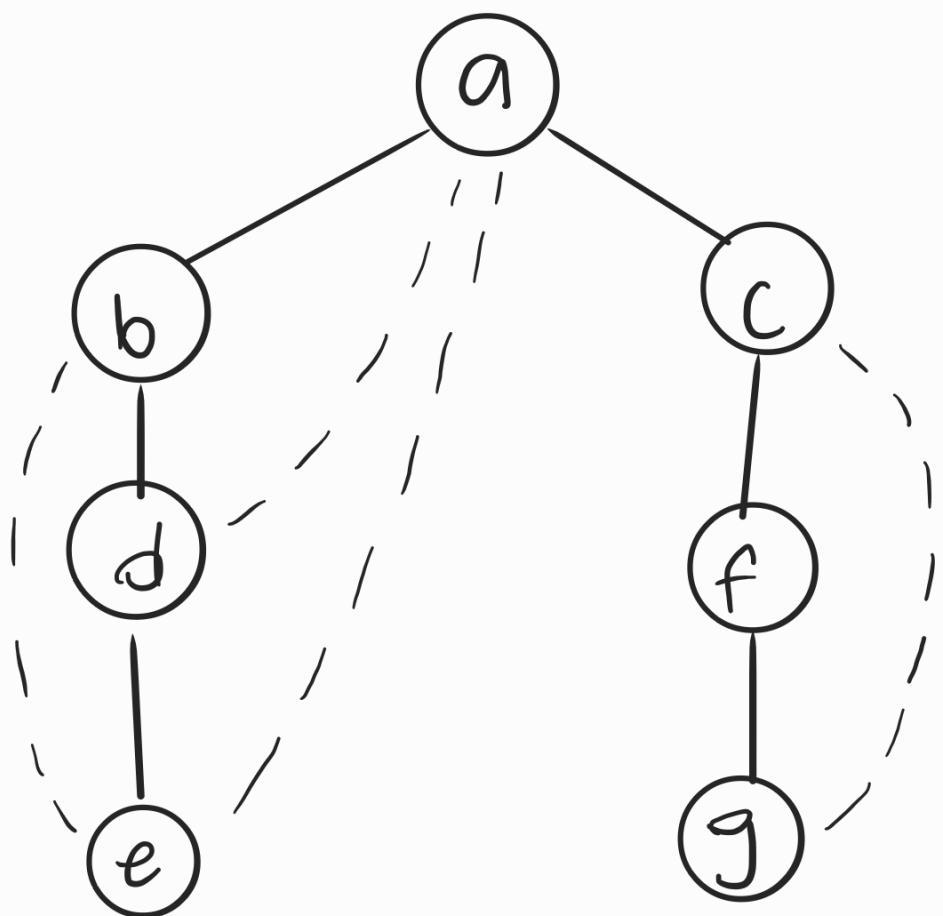
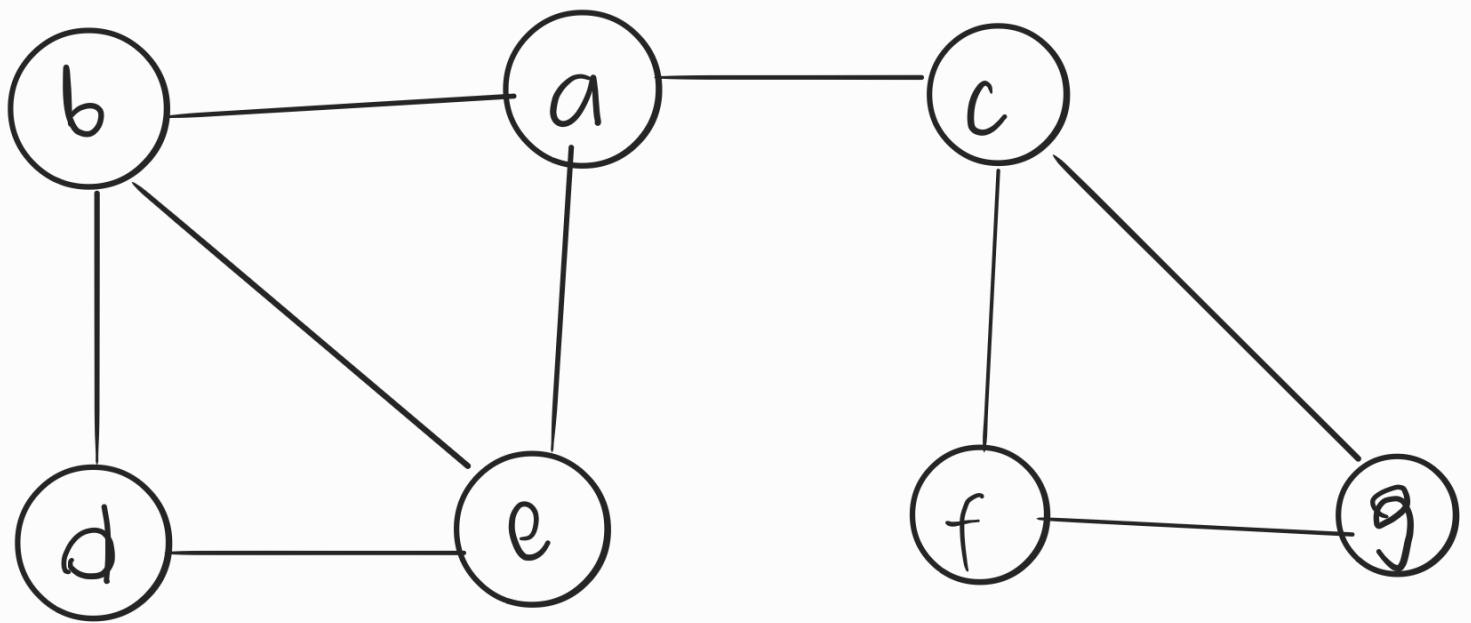
FIN

BUSQUEDA EN PROFUNDIDAD

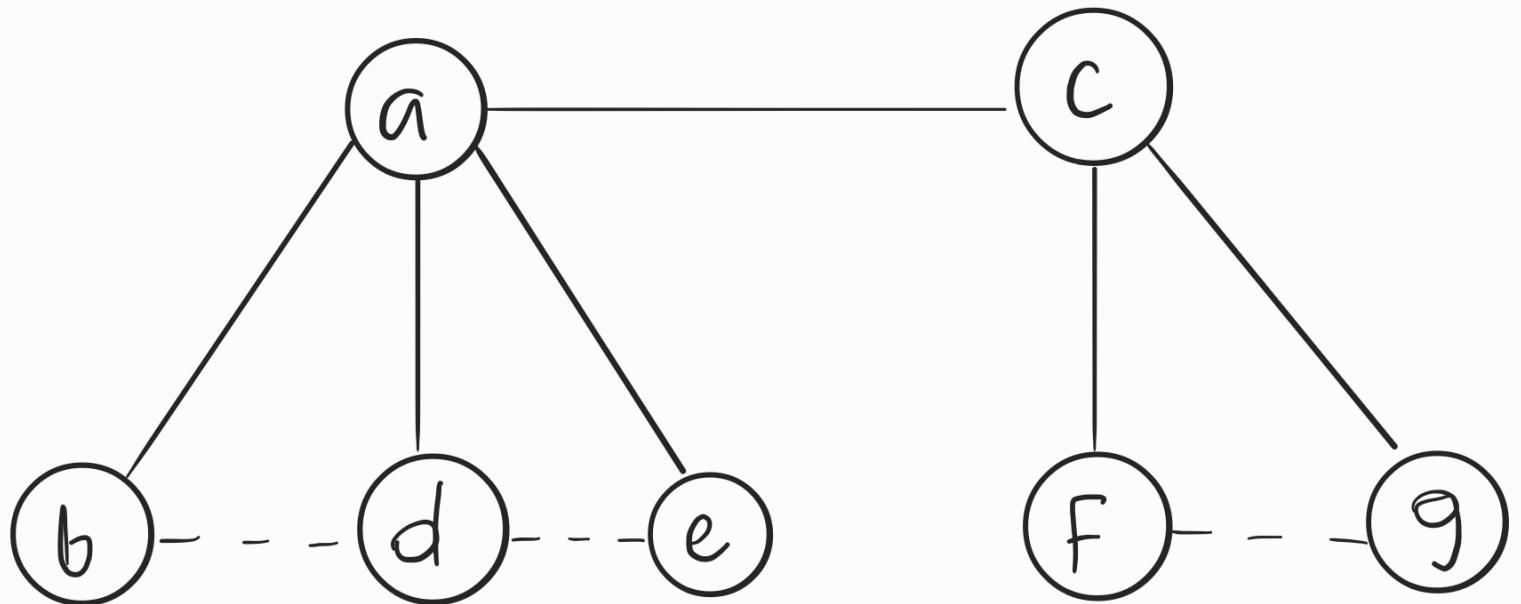
- Se puede emplear el mismo algoritmo que para grafos dirigidos
- Para grafos no dirigidos, solo hay arcos de arbol y retroceso

BUSQ EN AMPLITUD

- Desde cada vértice v se visitan todos los adyacentes, luego los descendientes
- También se puede construir un bosque abarcador
- Si G no es conexo, debe realizarse a partir de un vértice de cada componente.



RECORRIDO EN PROFUNDIDAD



RECORRIDO EN AMPLITUD

Método Tvertice **bea**: string
 // bea visita todos los vértices conectados
 a v usando la busq. en amplitud

C: Cola de vértices

x, y: vértice

tempString: String

COM

Visitar();

C. Insertar(this);

tempStr ← tempStr + etiqueta

mientras no vacia (C) hace

$x \leftarrow C.$ eliminar()

para cada vecino y adyacente a x hacer

Si no $y.$ visitado () entonces

$y.$ visitar()

$C.$ insertar (y);

tempStr ← tempStr + $y.$ etiqueta

fin Si

fin para cada;

fin mientras

Devolver tempStr

FIN

PUNTOS DE ARTICULACIÓN Y GRAFOS BICONEXOS

- Un punto de articulación es un vértice que si se elimina, se divide un componente conexo en dos o más partes
- A un grafo sin puntos de articulación se le llama grafo BICONEXO
- La bpf sirve para encontrar los componentes biconexos de un G

Para encontrar los puntos de Articulación

- Realizar la bpf numerando los vértices (numero_bp[v])
 - Por cada vértice obtener bays[v]
- ↓
- La raíz es un pto de articulación si tiene dos o más hijos
 - V es pto. de art. si hay un hijo w de V tal que bays[w] es mayor o igual a numero_bp(v)

• El bajo de un V es el numero mas pequeño de ese nodo v o cualquier otro nodo w accesible desde él.

• Se toma $\text{baj}(v)$ como min. de:

1. número_bp de v

2. número_bp de z para cualquier vértice z para el cual haya una arista de retroceso (v, z)

3. $\text{baj}(y)$ para cualquier hijo de v

