# iRAT #007 UT8

## Aho, $Hopcroft$, Ullman

## Grafos no dirigidos

Un grafo no dirigido $G = (V, A)$ consta de un conjunto finito de vértices V y de un conjunto de aristas A. Cada arista es un par no ordenado de vértices, es decir, si $(v, w)$ es una arista entonces $(v, w) = (w, v)$. Se utilizan para modelar relaciones simétricas entre objetos.

### 7.1 Definiciones

Los vértices w y v son adyacentes si $(v, w)$ o $(w, v)$ son aristas. **Se dice que la arista $(v, w)$ es incidente sobre los vértices v y w.**

Un camino es una secuencia de vértices $v_1, v_2, \ldots, v_n$ tal que $(v_i, v_{i+1})$ es una arista para $1 \leq i \leq n$. Un camino es simple si todos sus vértices son distintos, exceptuando $v_1$ y $v_n$, que pueden ser el mismo vértice. La longitud del camino es $n - 1$, siendo este el número de aristas en el camino. Se dice que $v_1, v_2, \ldots, v_n$ conecta a $v_1$ y $v_n$. Un grafo es conexo si todos sus pares de vértices están conectados.

**Subgrafo inducido de G**

Sea $G = (V, A)$ un grafo. Un subgrafo de G es un grafo $G' = (V', A')$ donde,

1. V' es un subconjunto de V
2. A' consta de las aristas (v, w) en A tales que w y v están en V'.

En este caso G' se conoce como un subgrafo inducido de G.

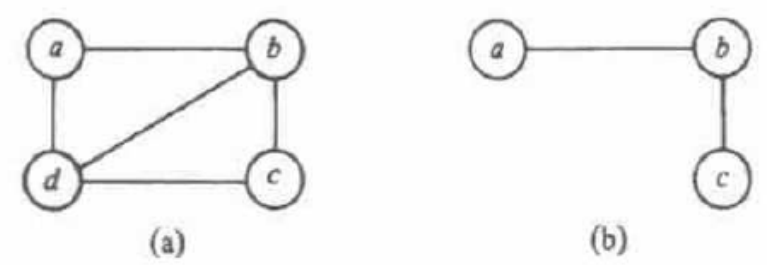


Fig. 7.1. Grafo con uno de sus subgrafos.

Un componente conexo de un grafo G es un subgrafo inducido conexo $maximal$.

**Ciclos**

Un ciclo simple de un grafo es un **camino simple que conecta a un vértice consigo mismo** en una longitud mayor o igual a tres. *Es decir bucles, caminos de longitud 1 o 2 no son considerados ciclos.*

Un grafo es cíclico si contiene **mínimo un ciclo.**

Un grafo conexo acíclico muchas veces se conoce como **árbol libre.**

**Árboles libres**

Los árboles libres tienen dos propiedades:

1. Todo árbol libre con $n \geq 1$ **vértices** contiene exactamente $n - 1$ **aristas.**
2. Si se agrega **cualquier arista** a un árbol libre, resulta un **ciclo**.

Se puede probar *(1)* por inducción o por el contraejemplo más pequeño. Es decir, si el contraejemplo de *(1)* es un grafo $G = (V, A)$ con un mínimo n de vértices, n no puede ser 1, porque un árbol de 1 vértice satisface *(1);* por lo tanto, n tiene que ser mayor que 1.

Ahora se pretende que en el árbol libre exista una arista con exactamente una arista incidente. Por definición, ningún vértice puede tener cero aristas incidentes, ya que sino G no sería conexo. Supóngase entonces que **todo vértice tiene por lo menos dos aristas incidentes**. Si nos paramos en un vértice y se sigue una de las dos aristas desde este, **en cada paso se abandona un vértice por una arista diferente a la que se utilizó para llegar**; formándose un camino $v_1, v_2, v_3, \ldots$.

## Métodos de representación

Una arista no dirigida entre v y w se representa simplemente con dos aristas dirigidas, una de v a w, y otra de w a v. Una **matriz de adyacencia** para un grafo no dirigido es **simétrica**. En una lista de adyacencia, si **(i, j)** es una arista, **i estará en la lista de j y vice versa**.

## 7.2 Árboles abarcadores de costo mínimo

Supóngase que $G = (V, A)$ es un grafo conexo donde cada arista tiene un costo asociado.

**Árbol abarcador**

Un árbol abarcador es un **árbol libre que conecta todos los vértices de V**; su costo es la suma de los costos de las aristas del árbol.

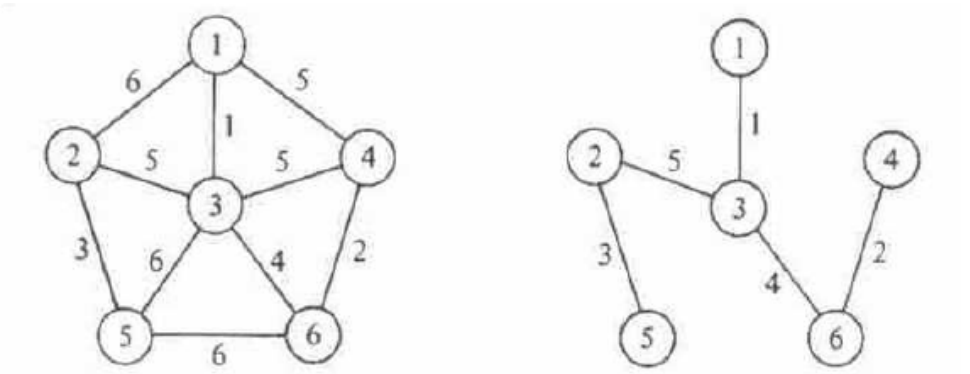


Fig. 7.4. Grafo y árbol abarcador.

## La propiedad AAM (Árbol Abarcador de costo Mínimo)

Sea G=(V,A) es grafo conexo con una función de costos definida en las aristas. Sea U algún subconjunto propio del conjunto V.

**Propiedad AAM**

Si $(u, v)$ es una arista de costo mínimo tal que $u \in U$ y $v \in V - U$, **existe un árbol abarcador de costo mínimo que incluye** $(u, v)$ **entre sus aristas**.

*Demostración por contradicción:* supóngase que no existe el árbol de costo mínimo para G que incluye (u, v). T es cualquier árbol abarcador de costo mínimo para G. Agregar (u, v) a T debe formar un ciclo, por la propiedad (2) de los árboles libres. Debe de existir otra arista (u',

v') en T tal que $u' \in U$ y $v' \in V - U$. Si eliminamos esta arista, se rompe el ciclo y se obtiene un árbol abarcador T' cuyo costo total no es mayor al de T, ya que por suposición $c(u, v) \leq c(u', v')$.

## Algoritmo de Prim

El algoritmo de Prim comienza cuando se asigna a un conjunto $U$ un valor inicial {1}, el el cual el árbol abarcador "crece" arista por arista. En cada paso se identifica la arista más corta (u, v) que conecta $U$ con $V - U$, y después se agrega v, que pertenece a $V - U$, a $U$. **Este paso se repite hasta que $U = V$.**

```
procedure Prim (G: grafo; var T: conjunto de aristas);
    {Prim construye un árbol abarcador de costo mínimo T para G}
    var
        U: conjunto de vértices;
        u, v: vértice;
    begin
        T := Ø;
        U := {1};
        while U ≠ V do begin
            sea (u, v) una arista de costo mínimo tal que u está en U y
                v en V–U;
            T := T ∪ {(u, v)};
            U := U ∪ {v}
        end
    end; {Prim}
```

**Fig. 7.6.** Esbozo del algoritmo de Prim.

Una forma de encontrar la arista de menor costo entre $U$ y $U - V$ es por medio de dos arreglos. $MasCercano[i]$ devuelve el vértice en $U$ más cercano a i en $V - U$. El otro, $MenorCosto[i]$, devuelve el costo de la arista $(i, MasCercano[i])$.

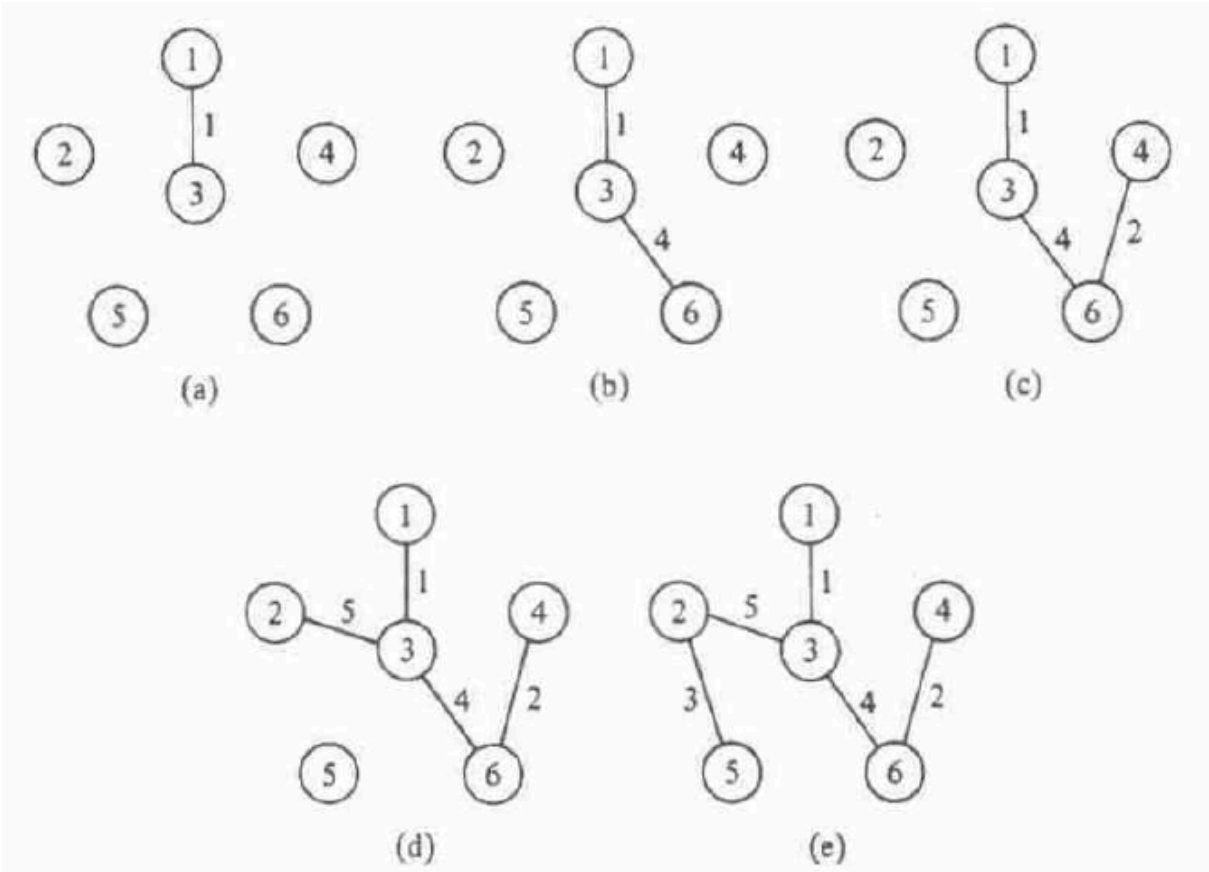


Fig. 7.7. Secuencias de aristas añadidas por el algoritmo de Prim.

En cada paso se revisa $MenorCosto$ para encontrar algún vértice k en V-U que esté más cercano a U. Se actualizan ambos arreglos tendiendo en cuenta que k ahora pertenece a U.

La complejidad del **algoritmo de Prim** es de $O(n^2)$.

La complejidad del **algoritmo de** $Kruksal$ **e**s máximo $O(a * log(a))$.

**Si el grafo es más denso, es decir, tiene muchas aristas, Prim es más rápido.*

Siendo $a$ el número de aristas en un grafo, si a es mucho menor que $n^2$ entonces $Kruksal$ es superior; si a es cercano a $n^2$ entonces se debe optar por el algoritmo de Prim.

## Algoritmo de $Kruksal$

Supóngase que se tiene un grafo conexo $G = (V, A)$ y una función de costos definida para cada arista. Otra forma de construir un árbol abarcador es comenzar con un grafo $G = (V, \emptyset)$ que solo tenga los vértices de G sin aristas. Por lo tanto, **cada vértice es un**

**componente conexo en sí mismo**. Conforme el algoritmo avanza, habrá siempre una colección de componentes conexos, y se elegirán las aristas que formen un árbol abarcador.

Para construirlo, **se examinarán las aristas en orden creciente de acuerdo al costo**. Si la arista **conecta dos vértices** que se encuentran **en dos componentes conexos distintos**, **se agregará dicha arista a T**. No se añadirá si conecta dos vértices que se encuentren en el mismo componente, puesto que esto puede generar un ciclo. **Una vez todos los vértices estén en el mismo componente, T es un árbol abarcador de costo mínimo para G**.

Las aristas pueden ordenarse a modo de cola de prioridad, donde se saca la arista de costo mínimo en cada iteración. También se requieren las siguientes operaciones:

1. $COMBINA(A, B, C)$; para combinar los componentes A y B en C y llamar al resultado A o B de forma arbitraria.
2. $ENCUENTRA(v, C)$; para devolver el nombre del componente de C, del cual el vértice v es miembro. Esta operación se utiliza para determinar si los dos vértices de una arista se encuentran en dos componentes distintos o en el mismo.
3. $INICIAL(A, v, C)$; para que A sea el nombre de un componente que pertenece a C, y que inicialmente solo contiene el vértice v.

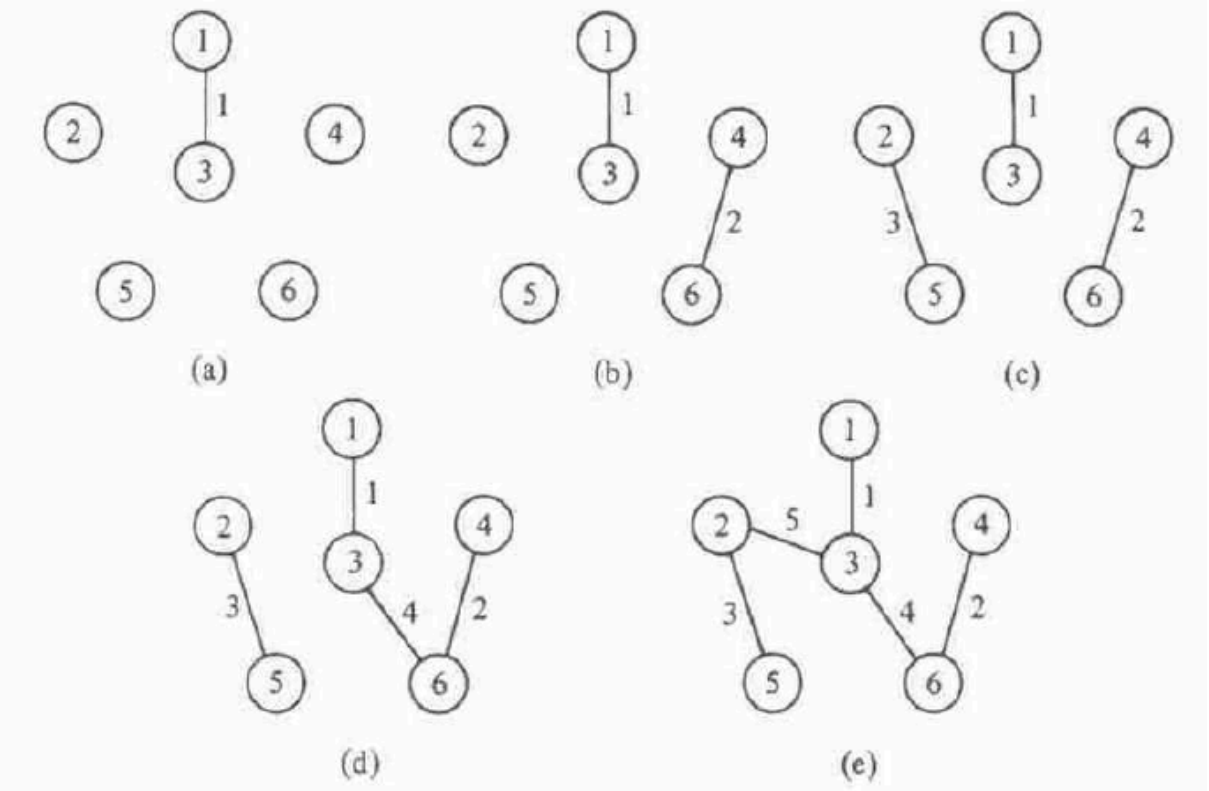


**Fig. 7.9.** Secuencia de aristas añadidas por el algoritmo de Kruskal.

## 7.3 Recorridos

## Búsqueda en profundidad

Para grafos dirigidos hay dos tipos de arcos: **de árbol y de retroceso**. Puesto que en los grafos dirigidos no existe distinción entre las aristas de retroceso y de avance, se llaman simplemente de retroceso.

1. **aristas de árbol**: aquellas aristas (v, w) tales que $bpf(v)$ llama directamente a $bpf(w)$ o viceversa
2. aristas de retroceso: aquellas aristas (v, w) tales que ni $bpf(v)$ ni $bpf(w)$ se llaman directamente, pero una llamó indirectamente a la otra (o sea,$bpf(w) \rightarrow bpf(x) \rightarrow bpf(v)$, de modo que w es antecesor de v).