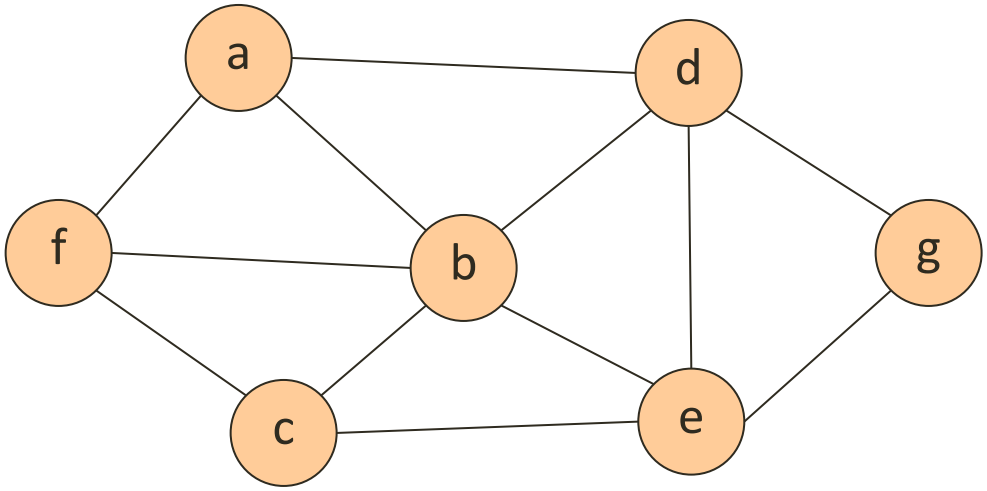
7. Árbol generador minimal

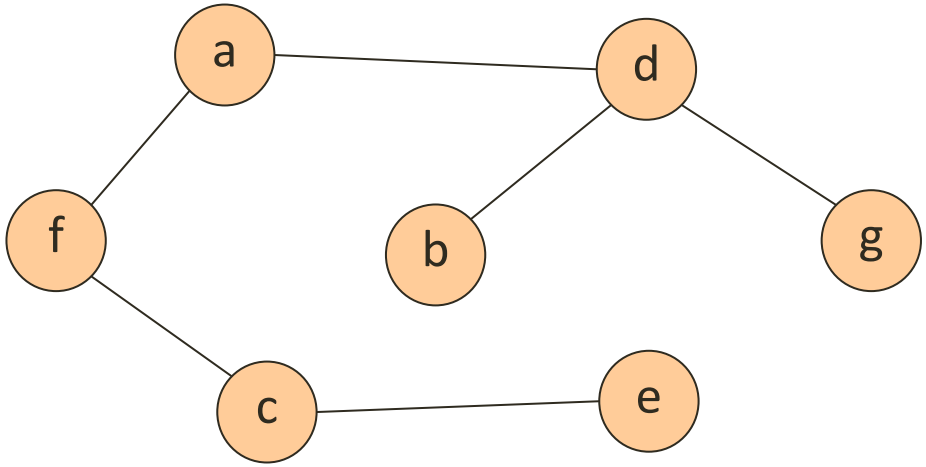
Introducción

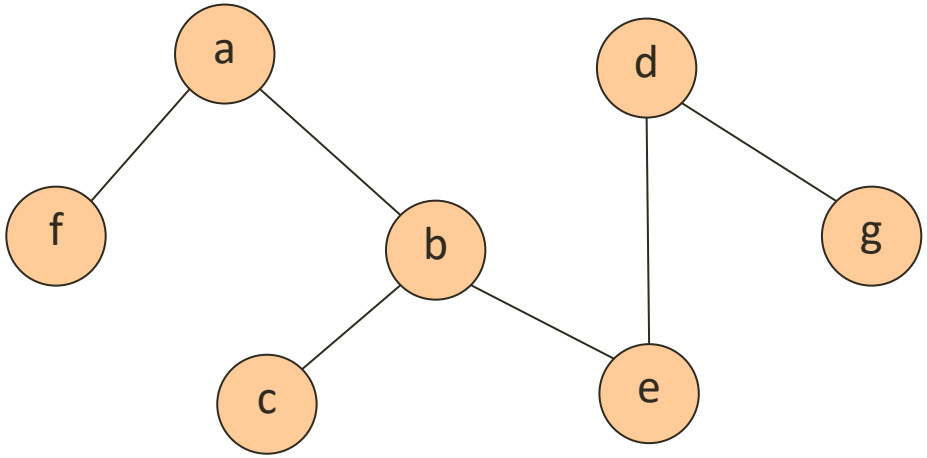
- Un grafo no dirigido es **conexo** si cualquier par de vértices está conectado por un camino
- Un grafo no dirigido acíclico y conexo es un **árbol**
- Un árbol generador (o árbol de recubrimiento) de un grafo (V, A) es un árbol (V', A') tal que:
  - $V' = V$
  - $A' \subseteq A$
- El problema de obtener el árbol generador minimal es muy importante por sus numerosas aplicaciones (diseño de redes, trazado de carreteras, astronomía, medicina, etc.)

7. Árbol generador minimal

Ejemplo



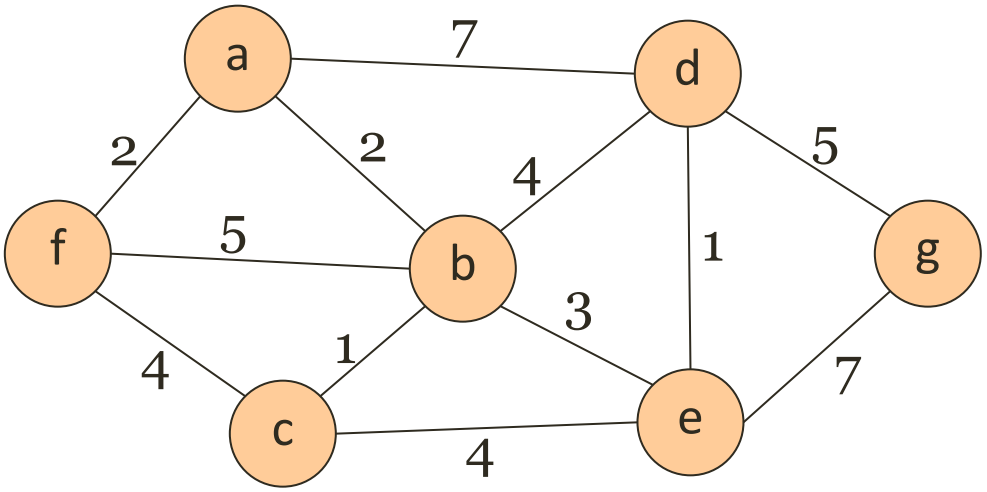




7. Árbol generador minimal

Ejemplo

- Los vértices del siguiente grafo representan los puntos de luz en una fábrica, y las aristas la longitud de cable necesaria para unir dos puntos de luz:

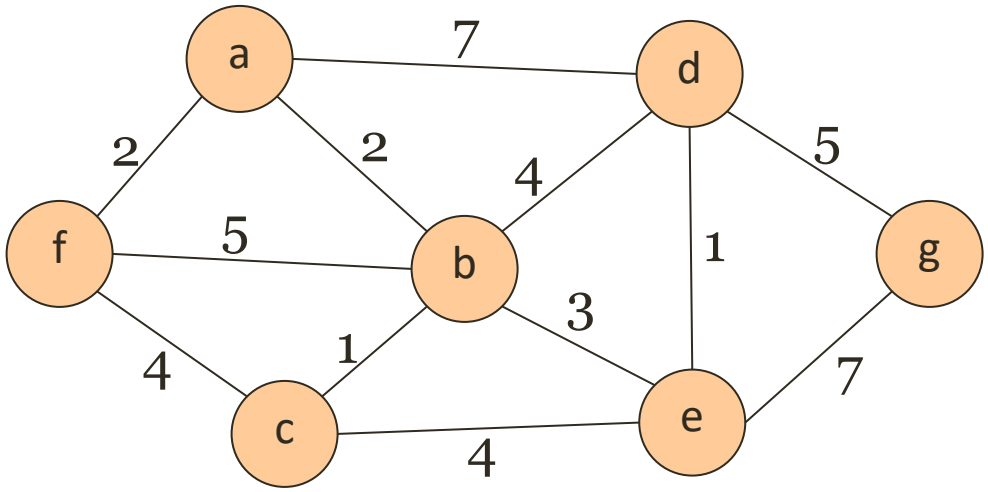


Problema:

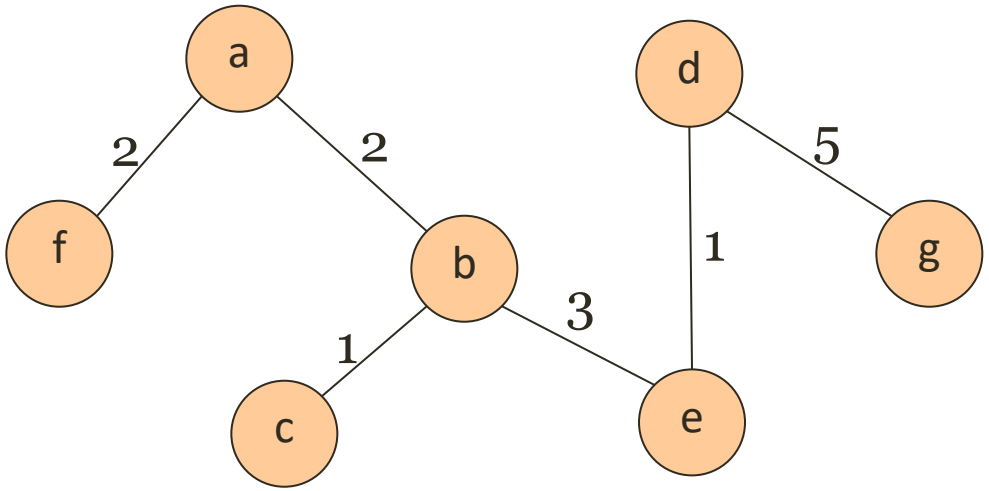
¿Cómo unir todos los puntos de luz utilizando la menor cantidad posible de cable?

7. Árbol generador minimal

Algoritmo de Kruskal



Resultado de aplicar *Kruskal*



7. Árbol generador minimal

Algoritmo de Kruskal

Paso 1: ordenar las aristas según su peso

Paso 2: partimos de un grafo sin aristas (sólo con los vértices)

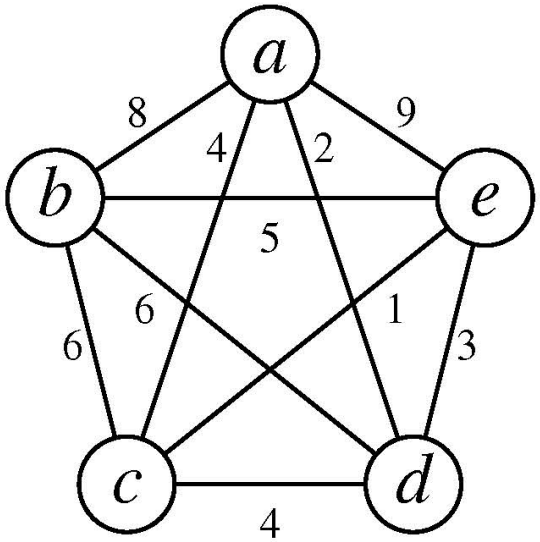
Paso 3: *mientras* $|A| < |V| - 1$ *hacer*:

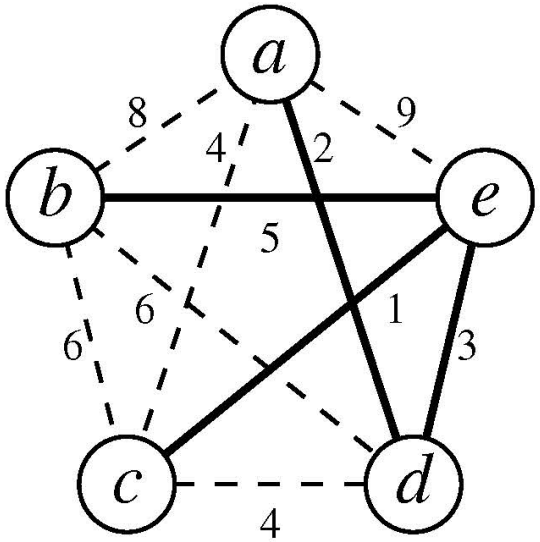
- Considerar la arista menor coste de las no utilizadas
- Incluir la arista en el grafo si no provoca ciclos

PROBLEMAS EN GRAFOS: ÁRBOL DE RECUBRIMIENTO DE COSTE MÍNIMO

Árbol de recubrimiento de un grafo no dirigido $G = (V, A)$**:** es un árbol libre $T = (V', A')$ tal que $V' = V$ y $A' \subseteq A$.

Problema: Dado un grafo conexo ponderado no dirigido $G = (V, A, p)$, encontrar un árbol de recubrimiento de G, $T = (V, A')$, tal que la suma de los pesos de las $|V| - 1$ aristas de T sea mínimo.





ALGORITMO DE KRUSKAL

Problema: ¿Cómo verificar eficientemente la condición de *"no crear ciclo"*?

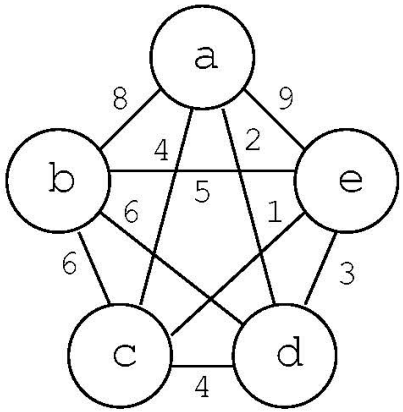
Solución: Mantener una colección de subconjuntos (disjuntos) con los vértices de cada árbol del bosque: *Una arista* (u, v) *no creará ciclo si* u *y* v *están en distintos subconjuntos* $\rightarrow$ componentes conexas $\rightarrow$ estructurar el conjunto de aristas seleccionadas como un MFset de vértices.

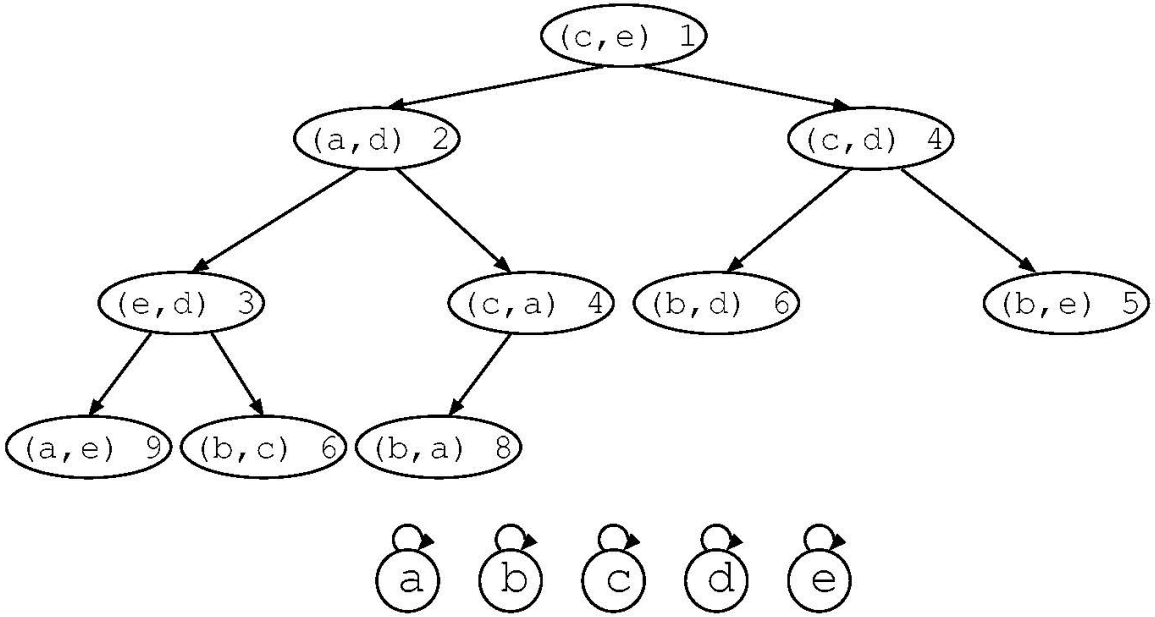
Problema: ¿Cómo seleccionar eficientemente la arista de menor peso en cada iteración?

Solución: Manteniendo las aristas en un MinHeap organizadas según su peso.

ALGORITMO DE KRUSKAL: UN EJEMPLO

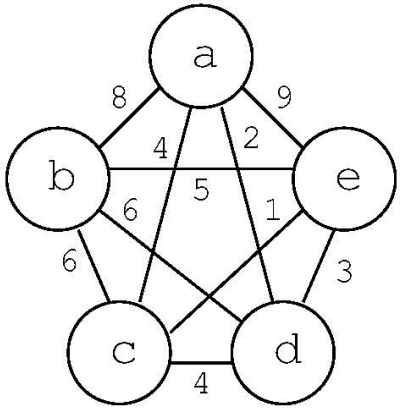
Inicialización

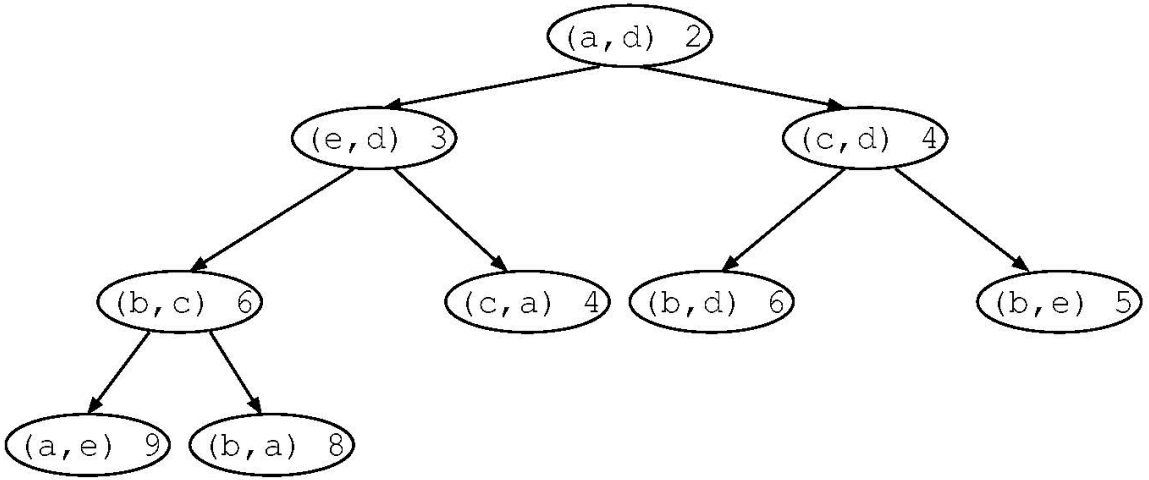


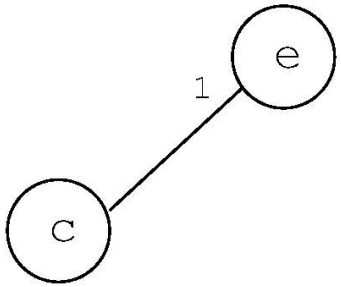


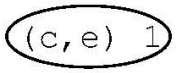
ALGORITMO DE KRUSKAL: UN EJEMPLO

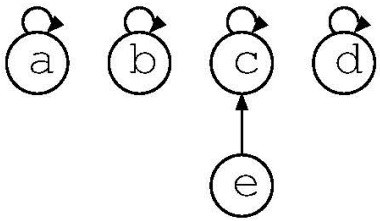
NumAristas = 1





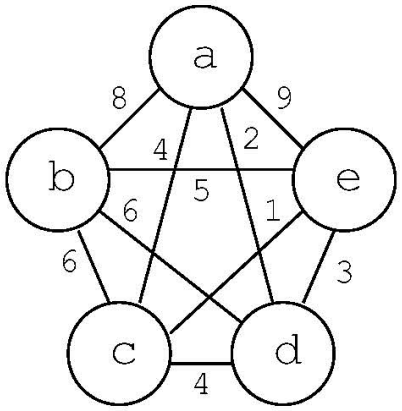


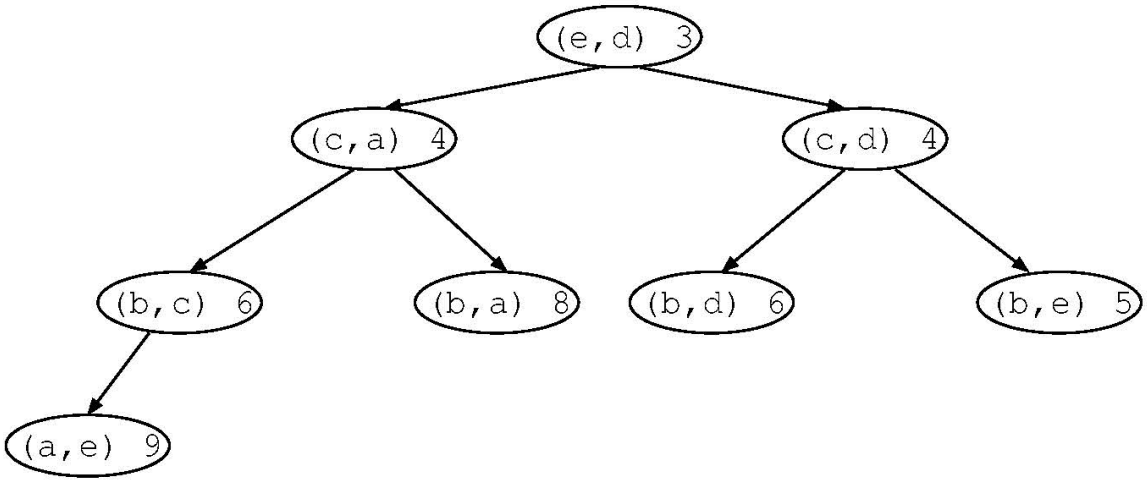


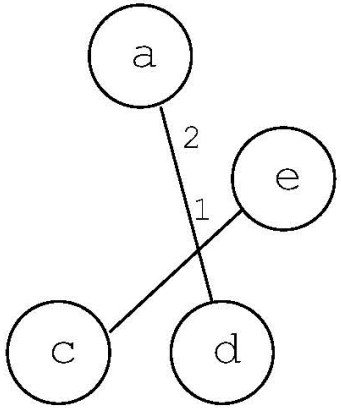


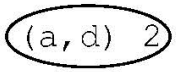
ALGORITMO DE KRUSKAL: UN EJEMPLO

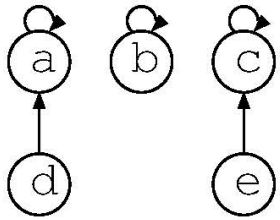
NumAristas = 2





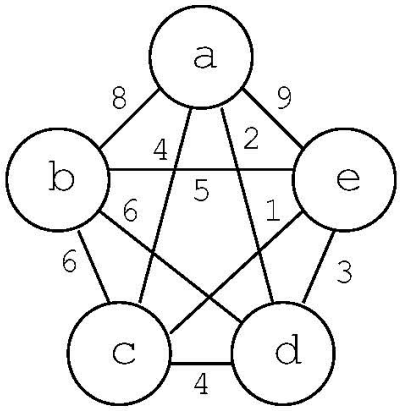


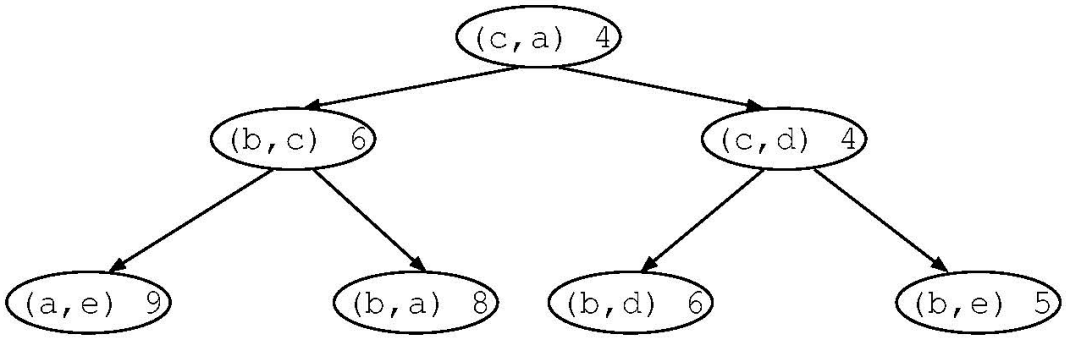


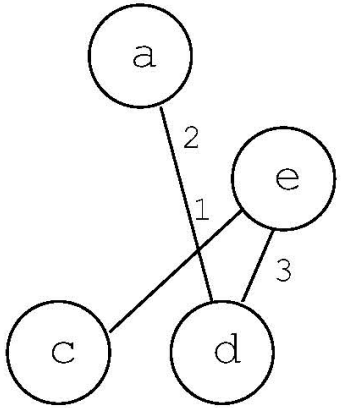


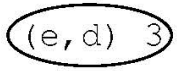
ALGORITMO DE KRUSKAL: UN EJEMPLO

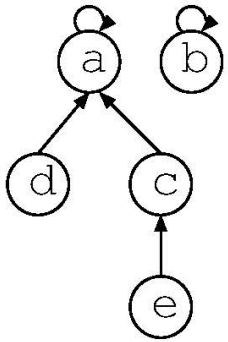
NumAristas = 3





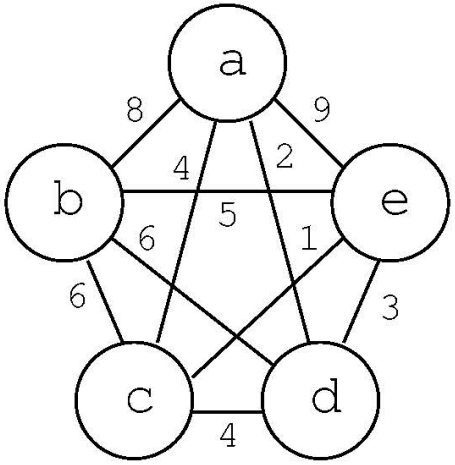


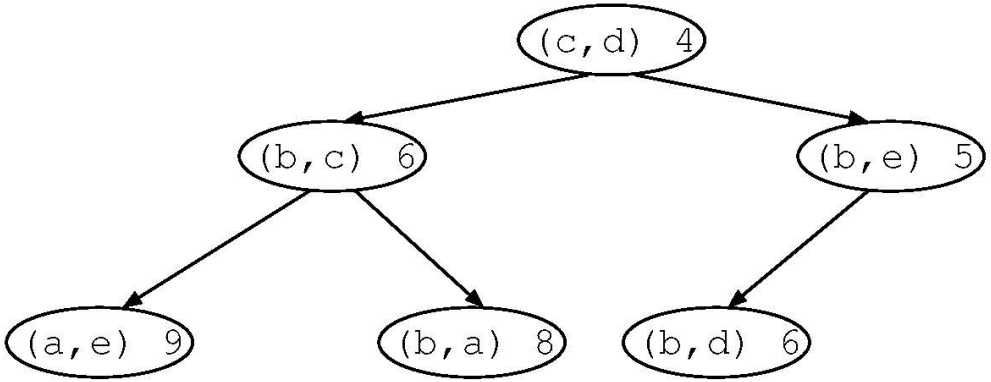


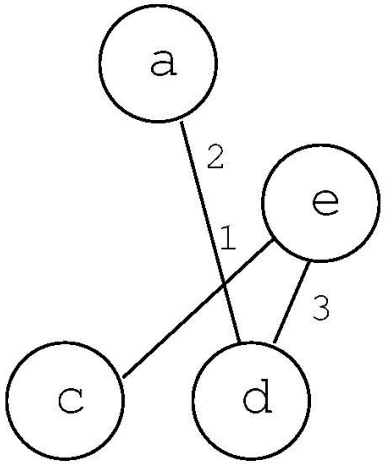


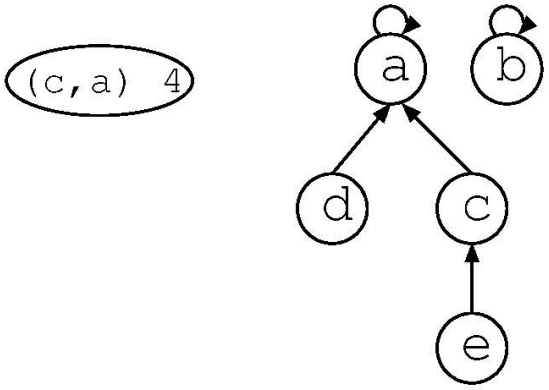
ALGORITMO DE KRUSKAL: UN EJEMPLO

NumAristas = 3



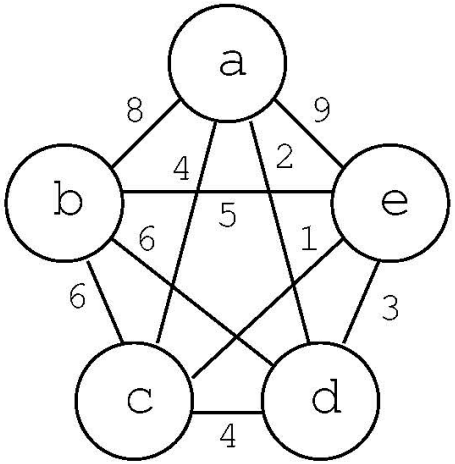


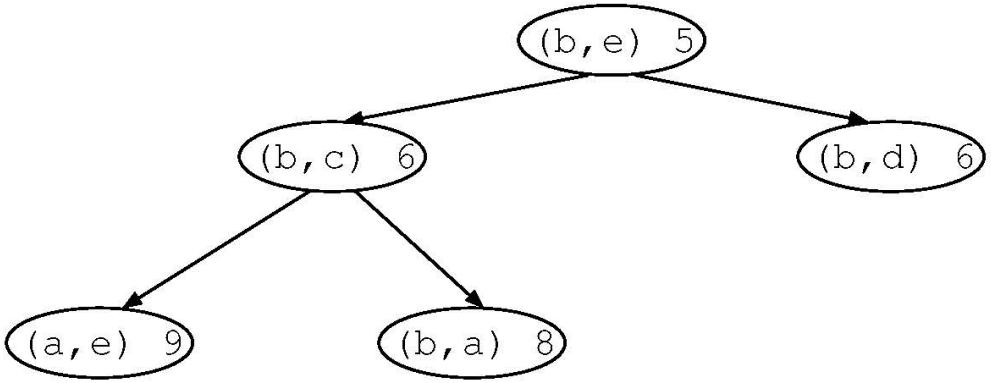


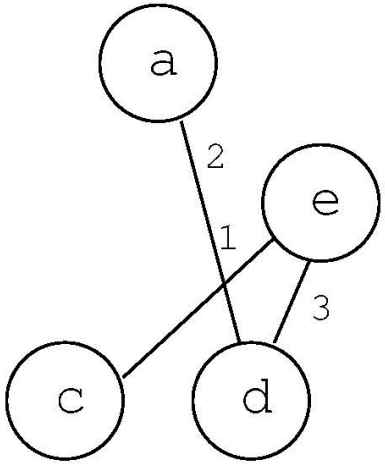


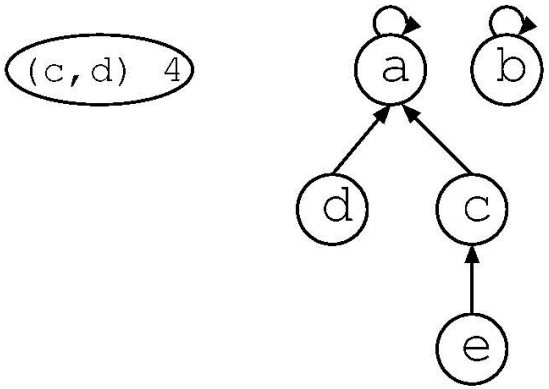
ALGORITMO DE KRUSKAL: UN EJEMPLO

NumAristas = 3



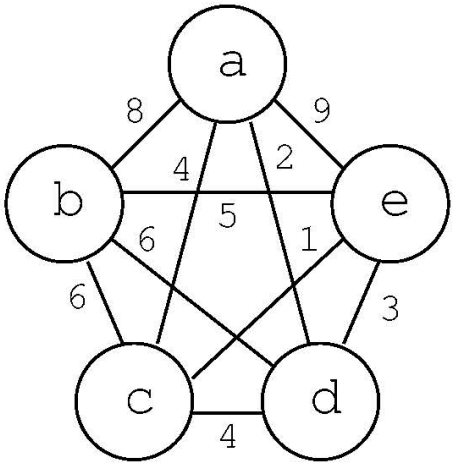


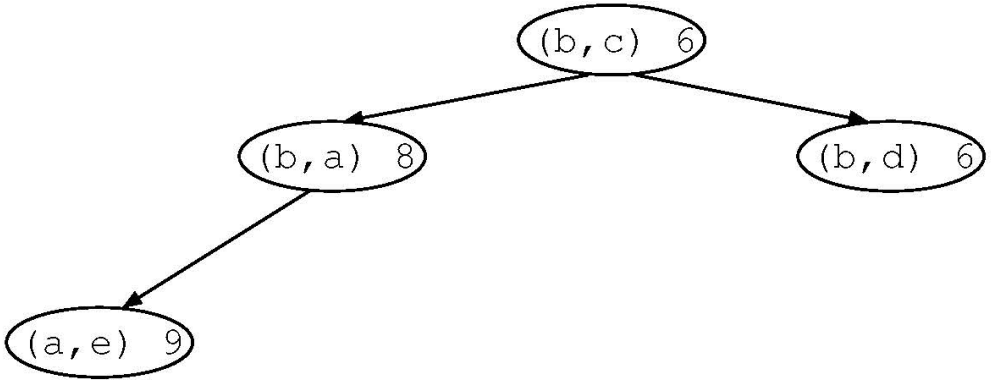


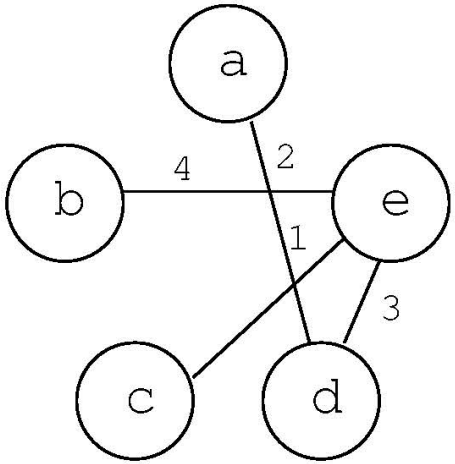


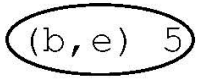
ALGORITMO DE KRUSKAL: UN EJEMPLO

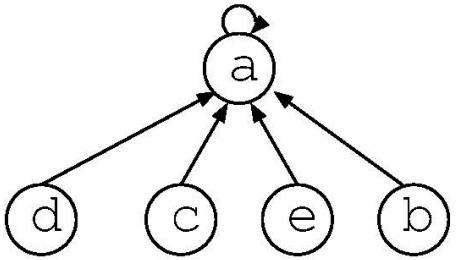
NumAristas = 4











7. Árbol generador minimal

Algoritmo de Kruskal

Paso 1: guardar las aristas en una cola de prioridad (build MinHeap)

Paso 2: partimos de un grafo sin aristas, sólo con los vértices (creamos UF-Set deV)

Paso 3: *mientras* $|A| < |V| - 1$ *hacer*:

- Recuperar y eliminar la arista menor coste del MinHeap

  Sea (i,j) esa arista

- Incluir la arista en el grafo si no provoca ciclos

  si find(i)<>find(j) /** Si la arista (i,j) no provoca ciclos

  entonces { Añadir (i,j) a la solución

  Hacer union(find(i), find(j)) }/** ahora ya están conectados

  /** Si la arista (i,j) provoca ciclos se ignora

$$T_{kruskal}(|V|,|A|) \in O(|A| \cdot \log|A|)$$

ALGORITMO DE KRUSKAL

El coste es $O(|A| \log |A|)$

construir $\|V\|$ MFsets	$O(\|V\|)$
construir MinHeap	$+O(\|A\|)$
$O(\|A\|)$ borrados en MinHeap	$+O(\|A\| \log \|A\|)$
$O(\|A\|)$ operaciones MFset	$+O(\|A\|)$

En general, el coste es bastante inferior a la cota $O(|A| \log |A|)$:

Si m es el número de iteraciones del bucle **while**, típicamente $m \approx |V|$ y si $|V| \ll |A| \leq |V|^2$, en la práctica, el coste está más cercano a

$$|A| + |V| \log |V|$$