Minimum Spanning Tree (Árbol de Expansión Mínima)

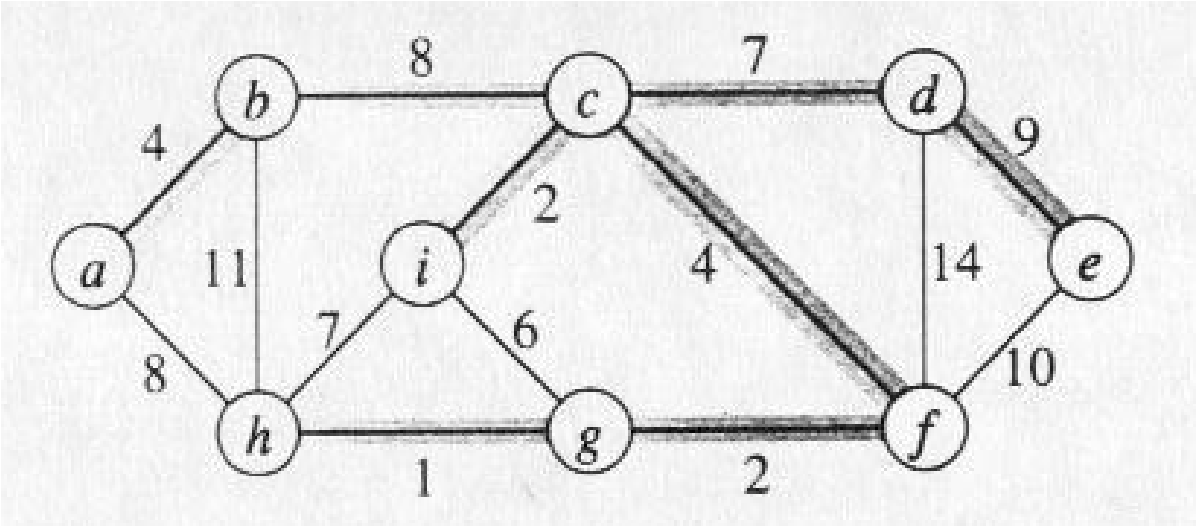
Agustín J. González

ELO320: Estructura de datos y Algoritmos

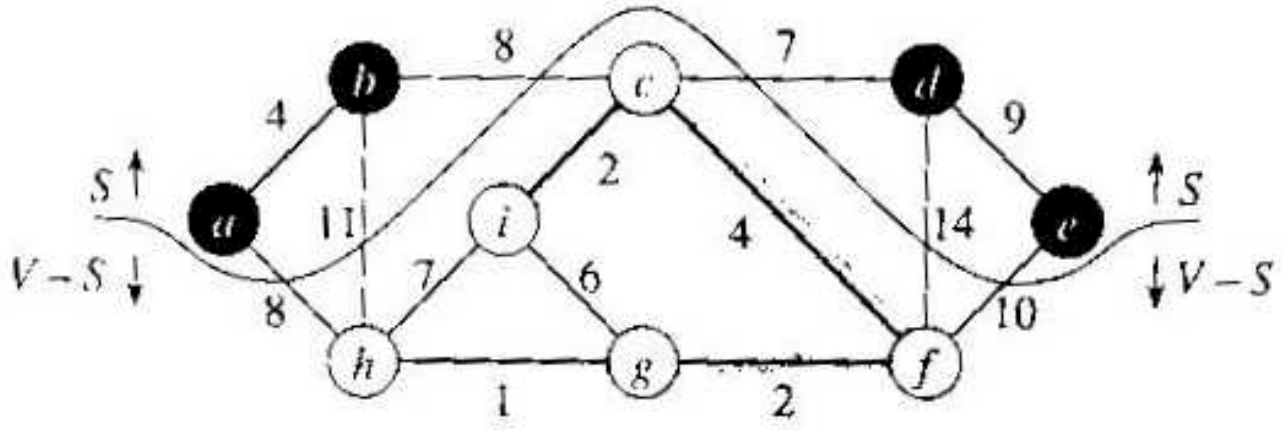
Introducción

- Lo que realmente se minimiza es el peso del árbol obtenido. No se minimiza el número de arcos, que se puede demostrar es igual a |V|-1.
- Hay varios problemas en los que se desea minimizar la interconexión de varios puntos. Por ejemplo en la confección de circuitos impresos.
- El problema consiste en minimizar la suma de todos los pesos de los arcos que inter-conectan todos los nodos de un grafo no dirigido.
- Hay dos algoritmos que resuelven este problema: Algoritmo de Kruskal y algoritmo de Prim (parecido al de Dijkstra - por verse).
- Ambos algoritmos corren en tiempo O(E lg V). Si se usa un heap especial (de Fibonacci) el algoritmo de Prim puede correr en O(E + V lg V) lo cual presenta ventajas cuando |V| << |E|
- Ambos algoritmos son ejemplos de la heurística de optimización llamada “greedy” (avaro, acaparador)

Ejemplo de árbol de expansión de mínimo peso



Una estrategia de avance



Algoritmo Genérico

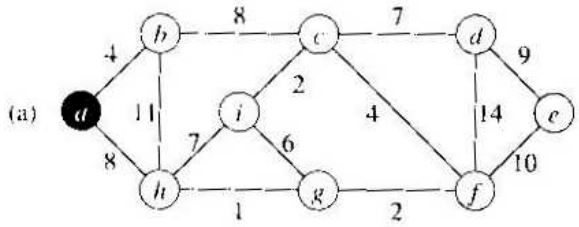
- La idea es ir haciendo crecer el número de nodos que pertenecen al árbol de peso mínimo.
- Debemos ir buscando nodos y arcos que puedan ser agregados y satisfagan la propiedad de mantener mínimo peso.
- Generic-MST(G, w) /* w es la función de peso */
  A = {};
  while(A no forma un “spanning Tree”)
      Encontrar un arco (u,v) que es seguro para A;
      A = A +{(u,v)};
  return A;

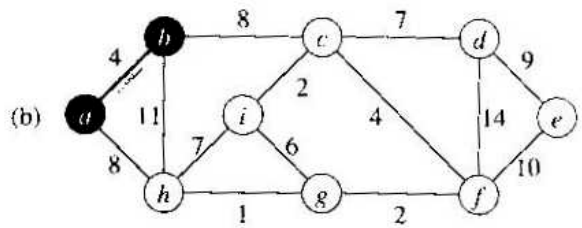
Algoritmo de Prim (1957 (Jarník 1930))

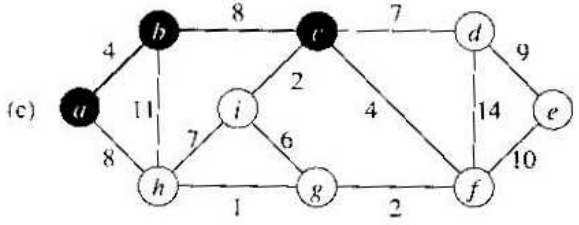
- MST-PRIM(G, w, r) /* G es el grafo, w es la función de peso, y r es el nodo por el cual empezamos (puede ser cualquiera del grafo) */

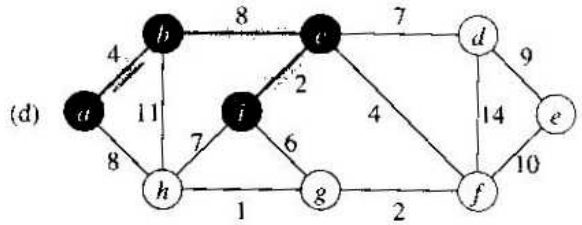
```
Q =  V[G];  /* Cola de prioridad con vértices fuera del árbol */
for (cada u en Q)
        key[u] = infinito; /* key es el peso mínimo para
                              conectar un nodo al árbol */
key [r] = 0;  /* raiz del árbol */
p [r] = NIL;
while (Q != {} )
        u = Extract_Min(Q);
        for (cada v en Adj [u] )
                if ( v está en Q && w(u,v) < key [v] )
                        p [v] = u;
                        key [v] = w(u,v);
```

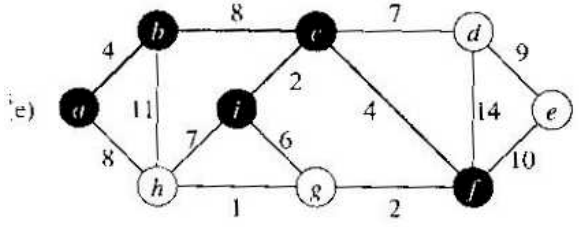
Ejemplo del algoritmo de Prim

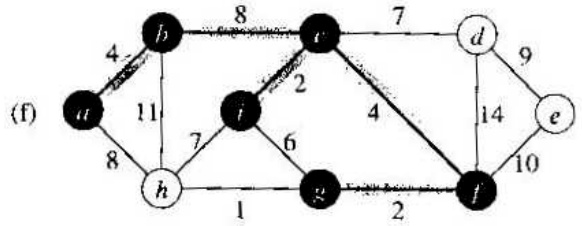


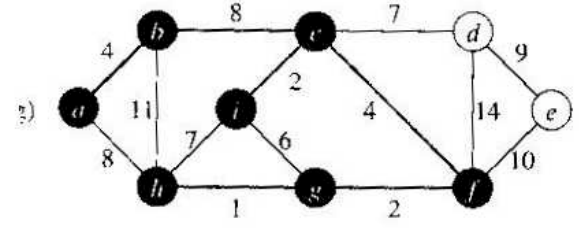


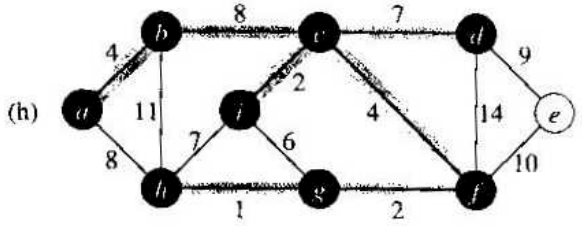


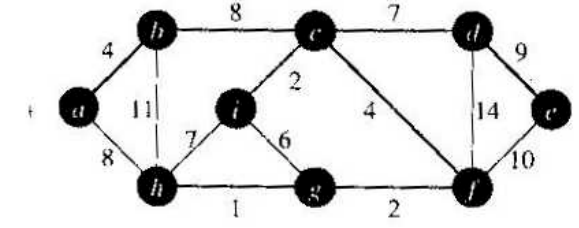












Comentarios sobre algoritmo de Prim

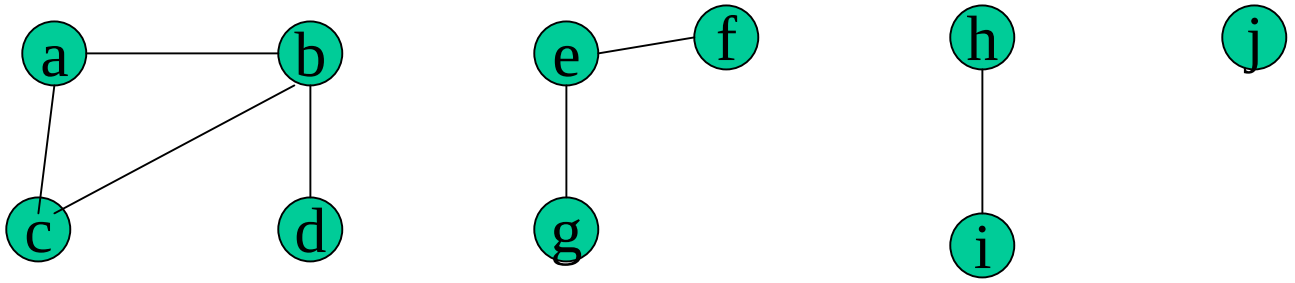
- La eficiencia de este algoritmo depende de cómo se implemente la cola de prioridad Q.
- Si se implementa con un heap binario se obtiene que ese algoritmo corre en tiempo $O(V \lg V + E \lg V) = O(E \lg V)$
- Si se usa un heap Fibonacci (no visto en el curso) el tiempo es $O(E+V \lg V)$, lo cual es una mejora cuando $|V| << |E|$

Algoritmo de Kruskal (1956)
Aspecto previo: Conjuntos disjuntos

- El algoritmo de Kruskal queda mejor definido si utilizamos conjuntos en su descripción. Por ello veremos la definición de un tipo de datos para conjuntos disjuntos.
- Consideremos una colección , S, de conjuntos disjuntos S = $\{S_1, S_2, S_3, .. S_k\}$.
- Cada conjunto será representado por un representante, el cual es un elemento cualquiera del conjunto.
- Deseamos disponer de las siguientes operaciones:
  Make_Set(x): crea un nuevo conjunto cuyo único elemento es apuntado por x (es así el representante).
  Union(x, y): Une a los conjuntos dinámicos que contienen a x e y en un nuevo conjunto. Como la colección de conjuntos es disjunta, esto destruye los conjuntos originales.
  Find_Set(x): retorna un puntero al representante del conjunto que contiene x.

Aplicación de Conjuntos disjuntos: Obtención de las componentes conexas en grafos no dirigidos

- Connected_Components(G)
  for (cada vertice v en V[G])
      Make_Set(v);
  for (cada arco (u,v) en E [G])
      Union(u,v);
- Ejemplo:



- Considerar que los arcos se procesan en el orden: (b,d); (e,g); (a,c); (h,i); (a,b); (e,f); (b,c)

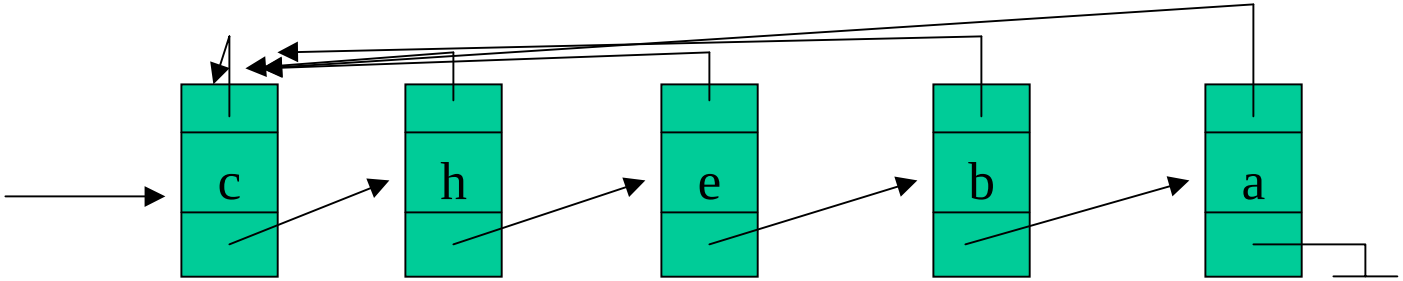
Posible implementación de Conjuntos disjuntos usando listas enlazadas

- Se puede tomar como base las listas simplemente enlazadas, e incorporando en cada nodo un puntero al primer nodo.
- La lista se crea con un elemento.
- No hay inserción y eliminación.
- La unión es juntar ambas listas.
- Find retorna el puntero al primer nodo.

- Typedef struct disjoint_set_node {
  struct disjoint_set_node * representante;
  struct disjoint_set_node * next;
  elementType element;
  } DISJOINT_SET_NODE;

Visualización gráfica del conjunto

- Typedef struct disjoint_set_node {
  struct disjoint_set_node * representante;
  struct disjoint_set_node * next;
  elementType element;
  } DISJOINT_SET_NODE;

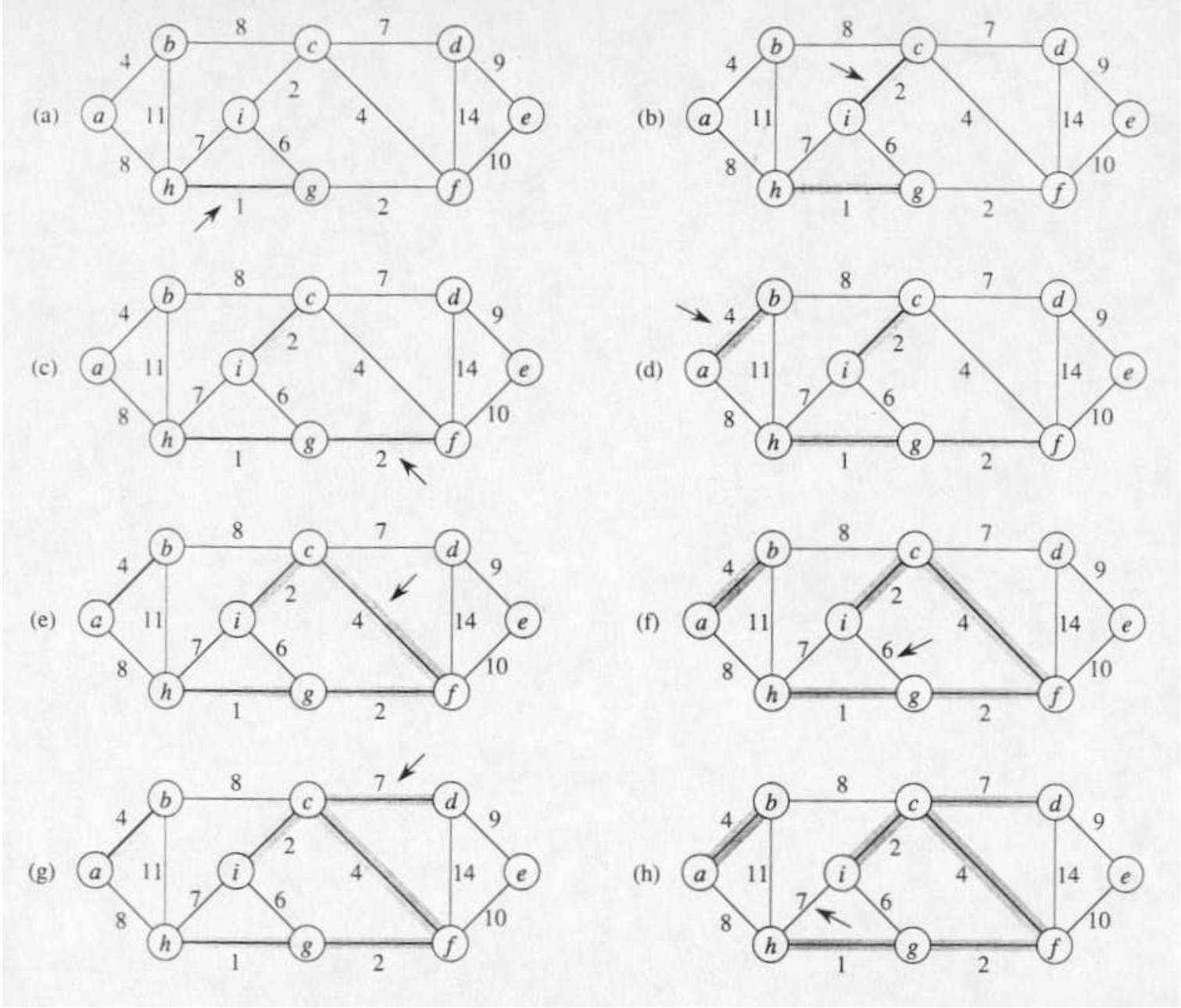


- Las operaciones Make_Set, Find_Set, y Union, pueden ser implementadas en forma mas o menos directa.

Algoritmo de Kruskal (1956)

```
MST_Kruskal (G,w) {
        A = {};
        for (cada vertice v en V[G] )
                Make_Set(v);
        Ordenar los arcos de E según peso w no decreciente;
        for ( cada arco (u,v) en E, en orden no decreciente de peso)
                if ( Find_Set(u) != Find_Set(v) ) {
                        A = A ∪ {(u,v)};
                        Union(u,v);
                }
}
```

Algoritmo de Kruskal :Ejemplo



Algoritmo de Kruskal :Ejemplo (continuación)