Algoritmos y estructuras de datos

TAD. Árbol de Expansión.

CEIS

Escuela Colombiana de Ingeniería

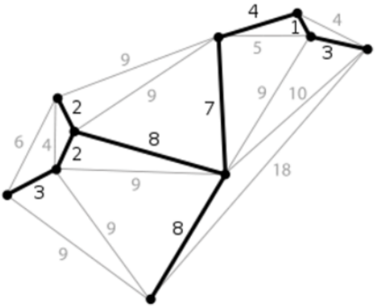
2024-2

Agenda

Árbol de Expansión

Un **árbol de expansión** de un grafo $G(E, V)$ es un árbol que:

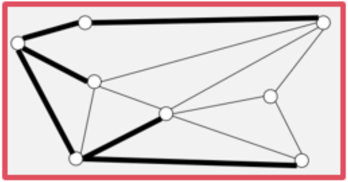
- Incluye todos los vértices de G
- Es un subgrafo de G
- Está conectado
- Es acíclico

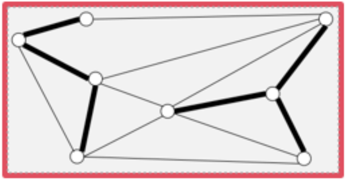


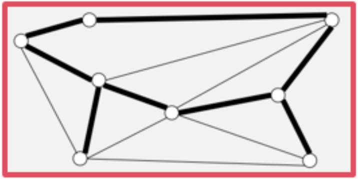
Árbol de Expansión

Un **árbol de expansión** de un grafo $G(E, V)$ es un árbol que:

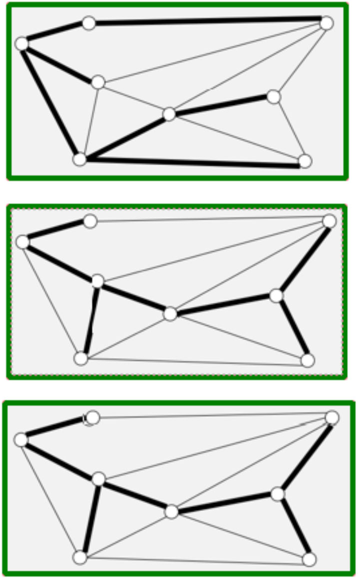
- Incluye todos los vértices de G
- Es un subgrafo de G
- Está conectado
- Es acíclico







Árbol de Expansión

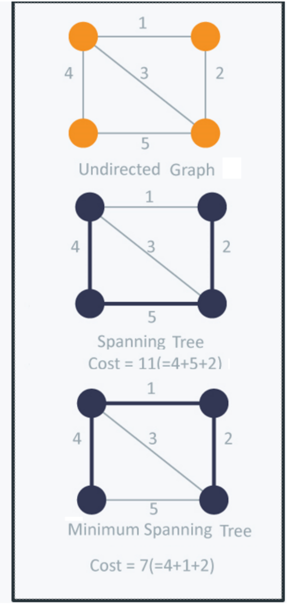


Un **árbol de expansión** de un grafo $G(E, V)$ es un árbol que:

- Incluye todos los vértices de G
- Es un subgrafo de G
- Está conectado
- Es acíclico

Árbol de Expansión Mínimo

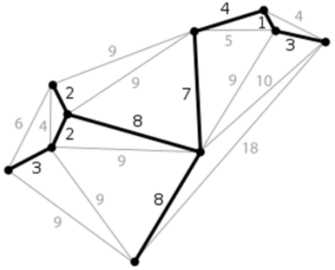
Un **árbol de expansión mínimo** de un grafo $G(E, V)$ es un **árbol de expansión** cuyo costo es el menor entre todos los demás árboles de expansión



Árbol de Expansión Mínimo

Problema

Dado un grafo no dirigido $G = (V, E)$ con una función de peso $w : E \longrightarrow R$ Deseamos encontrar un subconjunto acíclico $T \subseteq E$ que conecte todos los vértices y cuyo costo sea mínimo.



$$w(T) = \sum_{(u,v) \in T} w(u, v)$$

T es el **MST** (*Minimum Spanning Tree*)

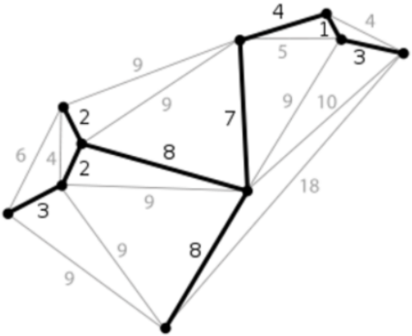
MST:Árbol de Expansión Mínimo

Problema

Dado un grafo no dirigido $G = (V, E)$ con una función de peso $w : E \longrightarrow R$, deseamos hallar un **MST** para G.

Solución voraz

Idea:
Hacer crecer el **MST** un arco a la vez.
Estrategia:
Mantener un conjunto de arcos A que crece cumpliendo el siguiente invariante.
INV: A es un subconjunto de algún **MST**



MST: Árbol de Expansión Mínimo

Problema

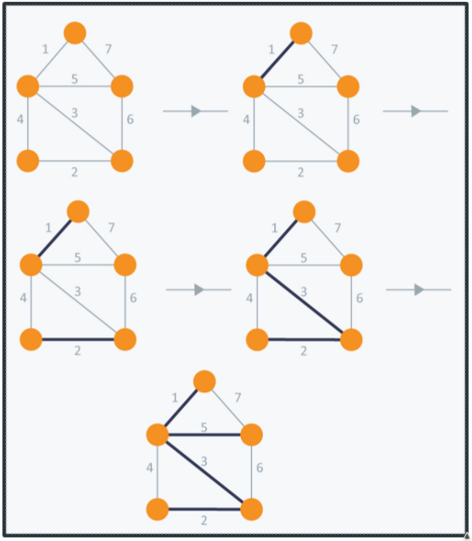
Dado un grafo no dirigido $G = (V, E)$ con una función de peso $w : E \longrightarrow R$, deseamos hallar un **MST** para G.

Solución voraz

Idea:
Hacer crecer el **MST** un arco a la vez.
Estrategia:
Mantener un conjunto de arcos A que crece cumpliendo el siguiente invariante.
INV: A es un subconjunto de algún **MST**

```
GENERIC-MST(G, w)
A = ∅
while A does not form a spanning tree
    find an edge (u, v) that is safe for A
    A = A ∪ {(u, v)}
return A
```

Árbol de Expansión Mínimo



Algoritmo de Kruskal's

1. Ordenar los arcos del grafo con relación a su peso
2. Agregar el arco con el menor costo al árbol de expansión, que conecte componentes que no estén conectados

   (para evitar ciclos)
3. Repetir el paso 2 hasta cubrir todos los vértices

Algoritmo de aproximación voraz

Árbol de Expansión Mínimo

Algoritmo de Kruskal's

1. Ordenar los arcos del grafo con relación a su peso
2. Agregar el arco con el menor costo al árbol de expansión, que conecte componentes que no estén conectados aún (para evitar ciclos)
3. Repetir el paso 2 hasta cubrir todos los vértices

Algoritmo de aproximación voraz

$\Theta(E\ logV)$

```
MST-KRUSKAL(G, w)
A = Ø
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into
    nondecreasing order by weight w
for each edge (u, v) ∈ G.E,
    taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

Árbol de Expansión Mínimo

Algoritmo de Kruskal's

Este algoritmo encuentra un 'arco seguro' para añadir al bosque creciente.
De todos los arcos que conectan dos arboles cualquiera en el bosque selecciona un arco (u, v) de menor peso.

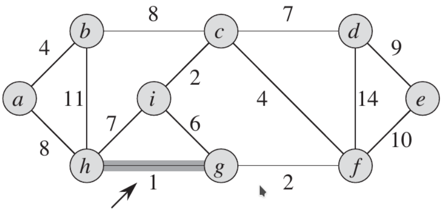
```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into
    nondecreasing order by weight w
for each edge (u, v) ∈ G.E,
    taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

Árbol de Expansión Mínimo

Algoritmo de Kruskal's

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into
    nondecreasing order by weight w
for each edge (u, v) ∈ G.E,
    taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

(a)

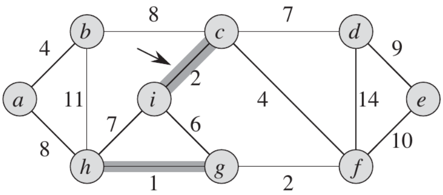


Árbol de Expansión Mínimo

Algoritmo de Kruskal's

```
MST-KRUSKAL(G, w)
A = Ø
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into
    nondecreasing order by weight w
for each edge (u, v) ∈ G.E,
    taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

(b)

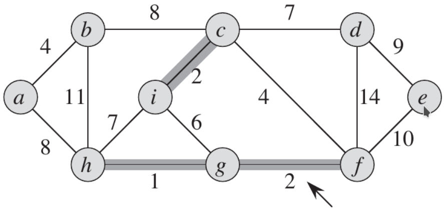


Algoritmo de Kruskal's

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into
    nondecreasing order by weight w
for each edge (u, v) ∈ G.E,
    taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

(c)

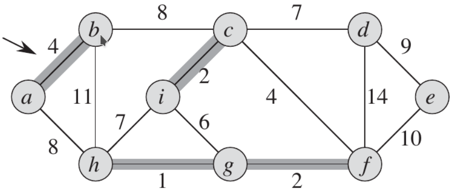


Árbol de Expansión Mínimo

Algoritmo de Kruskal's

```
MST-KRUSKAL(G, w)
  A = ∅
  for each vertex v ∈ G.V
      MAKE-SET(v)
  sort the edges of G.E into
      nondecreasing order by weight w
  for each edge (u, v) ∈ G.E,
      taken in nondecreasing order by weight
      if FIND-SET(u) ≠ FIND-SET(v)
          A = A ∪ {(u, v)}
          UNION(u, v)
  return A
```

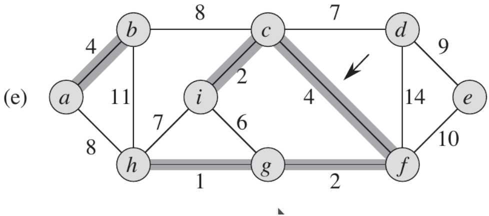
(d)



Árbol de Expansión Mínimo

Algoritmo de Kruskal's

```
MST-KRUSKAL(G, w)
    A = ∅
    for each vertex v ∈ G.V
        MAKE-SET(v)
    sort the edges of G.E into
        nondecreasing order by weight w
    for each edge (u, v) ∈ G.E,
        taken in nondecreasing order by weight
        if FIND-SET(u) ≠ FIND-SET(v)
            A = A ∪ {(u, v)}
            UNION(u, v)
    return A
```



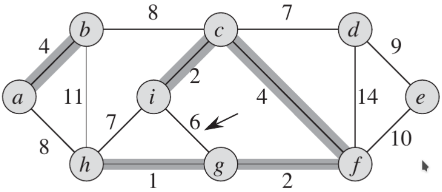
Árbol de Expansión Mínimo

Algoritmo de Kruskal's

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into
    nondecreasing order by weight w
for each edge (u, v) ∈ G.E,
    taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

(f)

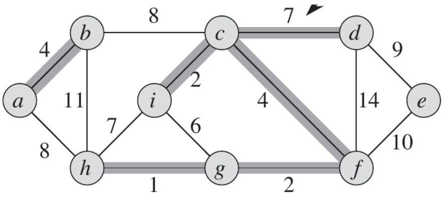


Árbol de Expansión Mínimo

Algoritmo de Kruskal's

```
MST-KRUSKAL(G, w)
  A = ∅
  for each vertex v ∈ G.V
      MAKE-SET(v)
  sort the edges of G.E into
      nondecreasing order by weight w
  for each edge (u, v) ∈ G.E,
      taken in nondecreasing order by weight
      if FIND-SET(u) ≠ FIND-SET(v)
          A = A ∪ {(u, v)}
          UNION(u, v)
  return A
```

(g)

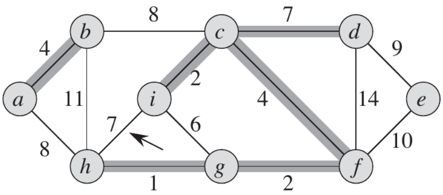


Árbol de Expansión Mínimo

Algoritmo de Kruskal's

```
MST-KRUSKAL(G, w)
A = Ø
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into
    nondecreasing order by weight w
for each edge (u, v) ∈ G.E,
    taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

(h)



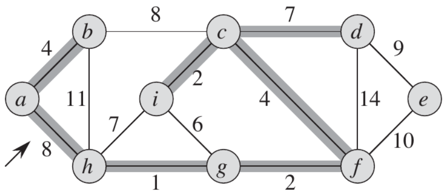
Árbol de Expansión Mínimo

Algoritmo de Kruskal's

MST-KRUSKAL(G, w)

$A = \emptyset$
for each vertex $v \in G.V$
    MAKE-SET(v)
sort the edges of $G.E$ into
    nondecreasing order by weight w
for each edge $(u, v) \in G.E$,
    taken in nondecreasing order by weight
    if FIND-SET$(u) \neq$ FIND-SET(v)
        $A = A \cup \{(u, v)\}$
        UNION(u, v)
return A

(i)

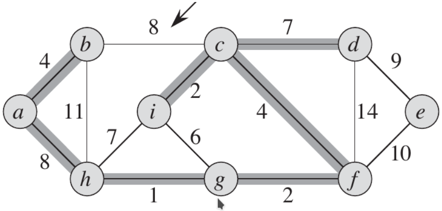


Árbol de Expansión Mínimo

Algoritmo de Kruskal's

```
MST-KRUSKAL(G, w)
  A = Ø
  for each vertex v ∈ G.V
      MAKE-SET(v)
  sort the edges of G.E into
      nondecreasing order by weight w
  for each edge (u, v) ∈ G.E,
      taken in nondecreasing order by weight
      if FIND-SET(u) ≠ FIND-SET(v)
          A = A ∪ {(u, v)}
          UNION(u, v)
  return A
```

(j)



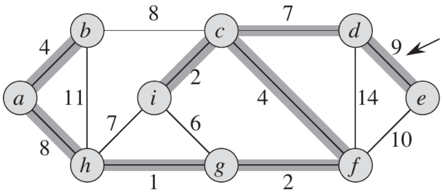
Árbol de Expansión Mínimo

Algoritmo de Kruskal's

MST-KRUSKAL(G, w)

```
A = Ø
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into
    nondecreasing order by weight w
for each edge (u, v) ∈ G.E,
    taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

(k)

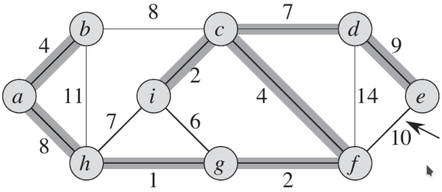


Árbol de Expansión Mínimo

Algoritmo de Kruskal's

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into
    nondecreasing order by weight w
for each edge (u, v) ∈ G.E,
    taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

(l)

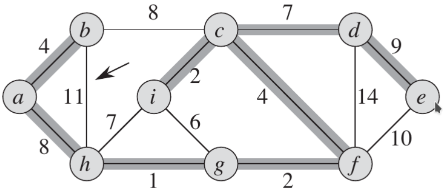


Árbol de Expansión Mínimo

Algoritmo de Kruskal's

```
MST-KRUSKAL(G, w)
  A = ∅
  for each vertex v ∈ G.V
      MAKE-SET(v)
  sort the edges of G.E into
      nondecreasing order by weight w
  for each edge (u, v) ∈ G.E,
      taken in nondecreasing order by weight
      if FIND-SET(u) ≠ FIND-SET(v)
          A = A ∪ {(u, v)}
          UNION(u, v)
  return A
```

(m)

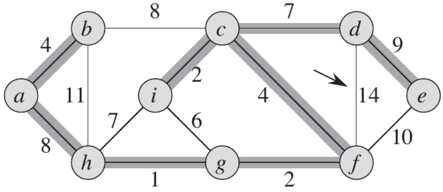


Árbol de Expansión Mínimo

Algoritmo de Kruskal's

```
MST-KRUSKAL(G, w)
A = Ø
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into
    nondecreasing order by weight w
for each edge (u, v) ∈ G.E,
    taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

(n)

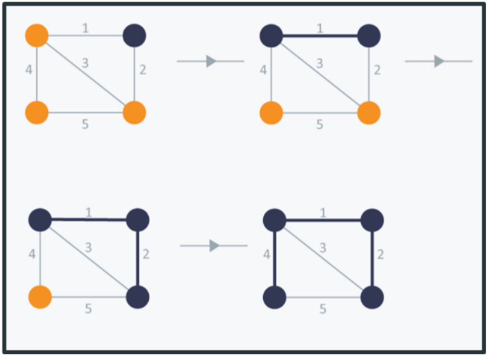


Árbol de Expansión Mínimo

Algoritmo de Prim

1. Escoger un nodo arbitrario y marcarlo
2. Escoger el vértice no marcado (para evitar ciclos) de menor peso desde el nodo escogido y marcar el nodo destino.
3. Repetir el paso 2 hasta cubrir todos los vértices

Algoritmo de aproximación voraz



Árbol de Expansión Mínimo

Algoritmo de Prim

1. Escoger un nodo arbitrario y marcarlo
2. Escoger el vértice no marcado (para evitar ciclos) de menor peso desde el nodo escogido y marcar el nodo destino.
3. Repetir el paso 2 hasta cubrir todos los vértices

Algoritmo de aproximación voraz

$\Theta(E \; log V)$

```
MST-PRIM(G, w, r)
for each vertex u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = ∅
for each vertex u ∈ G.V
    INSERT(Q, u)
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each vertex v in G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
            DECREASE-KEY(Q, v, w(u, v))
```

Árbol de Expansión Mínimo

Algoritmo de Prim

El árbol comienza desde un vértice arbitrario r como raíz y crece hasta que el árbol llega a todos los vértices en V.
En cada paso se añade al árbol A un enlace que conecta A con un vértice aislado, uno en el que ningún enlace de A es incidente.

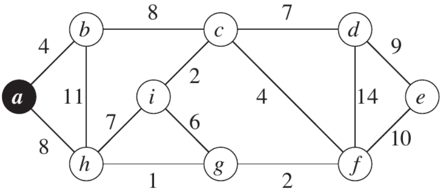
```
MST-PRIM(G, w, r)
for each vertex u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = ∅
for each vertex u ∈ G.V
    INSERT(Q, u)
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each vertex v in G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
            DECREASE-KEY(Q, v, w(u, v))
```

Árbol de Expansión Mínimo

Algoritmo de Prim

```
MST-PRIM(G, w, r)
for each vertex u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = ∅
for each vertex u ∈ G.V
    INSERT(Q, u)
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each vertex v in G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
            DECREASE-KEY(Q, v, w(u, v))
```

(a)

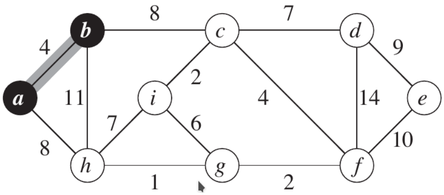


Árbol de Expansión Mínimo

Algoritmo de Prim

```
MST-PRIM(G, w, r)
for each vertex u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = ∅
for each vertex u ∈ G.V
    INSERT(Q, u)
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each vertex v in G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
            DECREASE-KEY(Q, v, w(u, v))
```

(b)

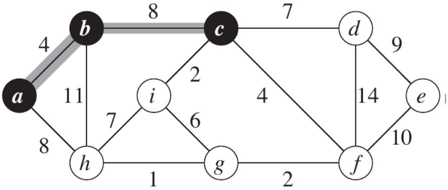


Árbol de Expansión Mínimo

Algoritmo de Prim

```
MST-PRIM(G, w, r)
for each vertex u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = ∅
for each vertex u ∈ G.V
    INSERT(Q, u)
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each vertex v in G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
            DECREASE-KEY(Q, v, w(u, v))
```

(c)

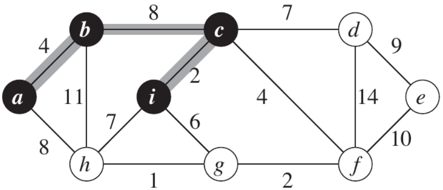


Árbol de Expansión Mínimo

Algoritmo de Prim

```
MST-PRIM(G, w, r)
  for each vertex u ∈ G.V
      u.key = ∞
      u.π = NIL
  r.key = 0
  Q = ∅
  for each vertex u ∈ G.V
      INSERT(Q, u)
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      for each vertex v in G.Adj[u]
          if v ∈ Q and w(u, v) < v.key
              v.π = u
              v.key = w(u, v)
              DECREASE-KEY(Q, v, w(u, v))
```

(d)

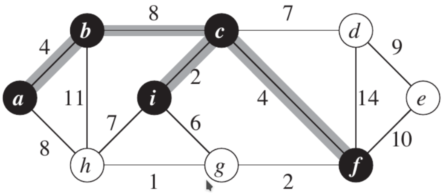


Árbol de Expansión Mínimo

Algoritmo de Prim

```
MST-PRIM(G, w, r)
for each vertex u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = ∅
for each vertex u ∈ G.V
    INSERT(Q, u)
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each vertex v in G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
            DECREASE-KEY(Q, v, w(u, v))
```

(e)

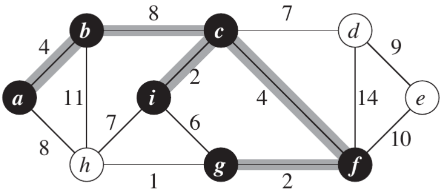


Árbol de Expansión Mínimo

Algoritmo de Prim

```
MST-PRIM(G, w, r)
for each vertex u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = ∅
for each vertex u ∈ G.V
    INSERT(Q, u)
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each vertex v in G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
            DECREASE-KEY(Q, v, w(u, v))
```

(f)

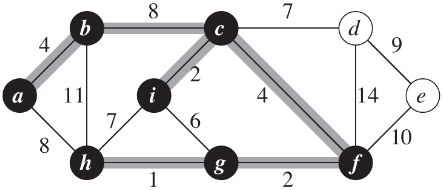


Árbol de Expansión Mínimo

Algoritmo de Prim

```
MST-PRIM(G, w, r)
for each vertex u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = ∅
for each vertex u ∈ G.V
    INSERT(Q, u)
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each vertex v in G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
            DECREASE-KEY(Q, v, w(u, v))
```

(g)

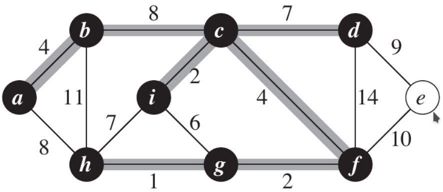


Árbol de Expansión Mínimo

Algoritmo de Prim

```
MST-PRIM(G, w, r)
  for each vertex u ∈ G.V
      u.key = ∞
      u.π = NIL
  r.key = 0
  Q = ∅
  for each vertex u ∈ G.V
      INSERT(Q, u)
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      for each vertex v in G.Adj[u]
          if v ∈ Q and w(u, v) < v.key
              v.π = u
              v.key = w(u, v)
              DECREASE-KEY(Q, v, w(u, v))
```

(h)

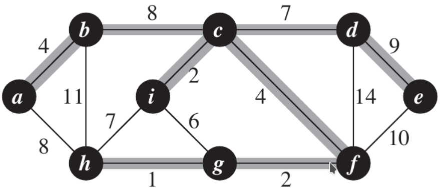


Árbol de Expansión Mínimo

Algoritmo de Prim

```
MST-PRIM(G, w, r)
for each vertex u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = ∅
for each vertex u ∈ G.V
    INSERT(Q, u)
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each vertex v in G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
            DECREASE-KEY(Q, v, w(u, v))
```

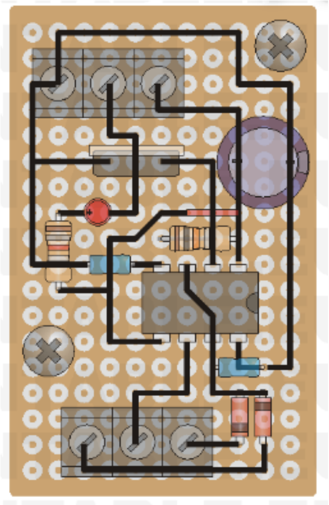
(i)



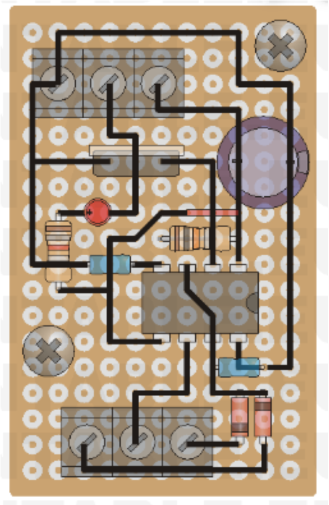
Circuitos

Problema

En el diseño de circuitos electrónicos es necesario interconectar diversos componentes eléctricos con un cable entre ellos.
Para interconectar un conjunto de n pines, podemos utilizar $n - 1$ cables.
La interconexión que use la menor cantidad de cable es el mas deseable.



Circuitos



Problema

Este problema se puede modelar como un grafo no dirigido $G = (V, E)$, dónde
V : es el conjunto de pines,
E : es el conjunto posibles conexiones entre parejas de pines
$w : E \longrightarrow R$; es la longitud del cable de u a v.
Deseamos encontrar un **MST** para G

Solución

¿?