Árbol recubridor mínimo

...

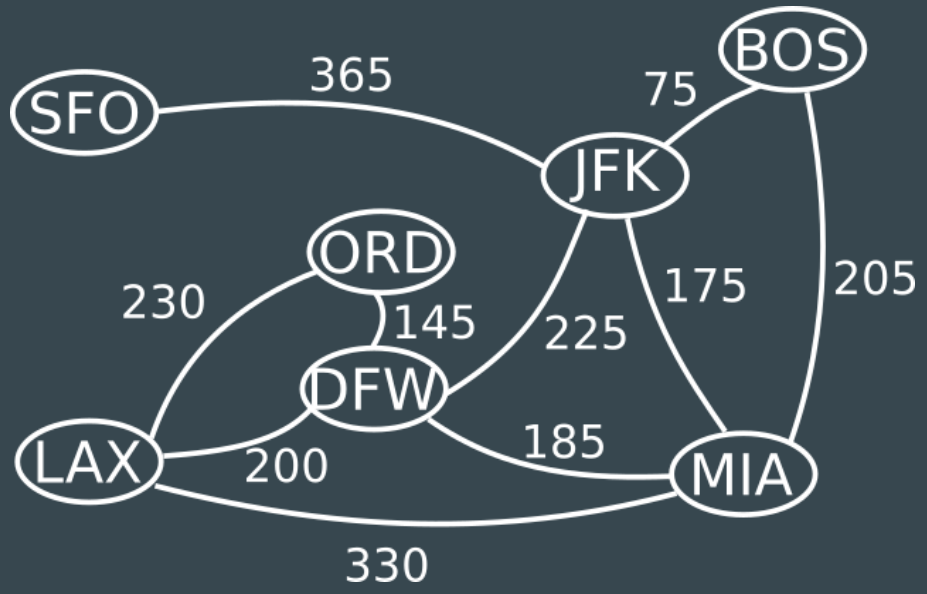
Algoritmos y Estructuras de Datos
2023

UNRaf UNIVERSIDAD NACIONAL DE RAFAELA

Se desean conectar en red todas las computadoras de un aula ¿Cómo podría calcular el orden en que deberían conectarse para utilizar la menor cantidad de cable posible?

Árbol recubridor mínimo

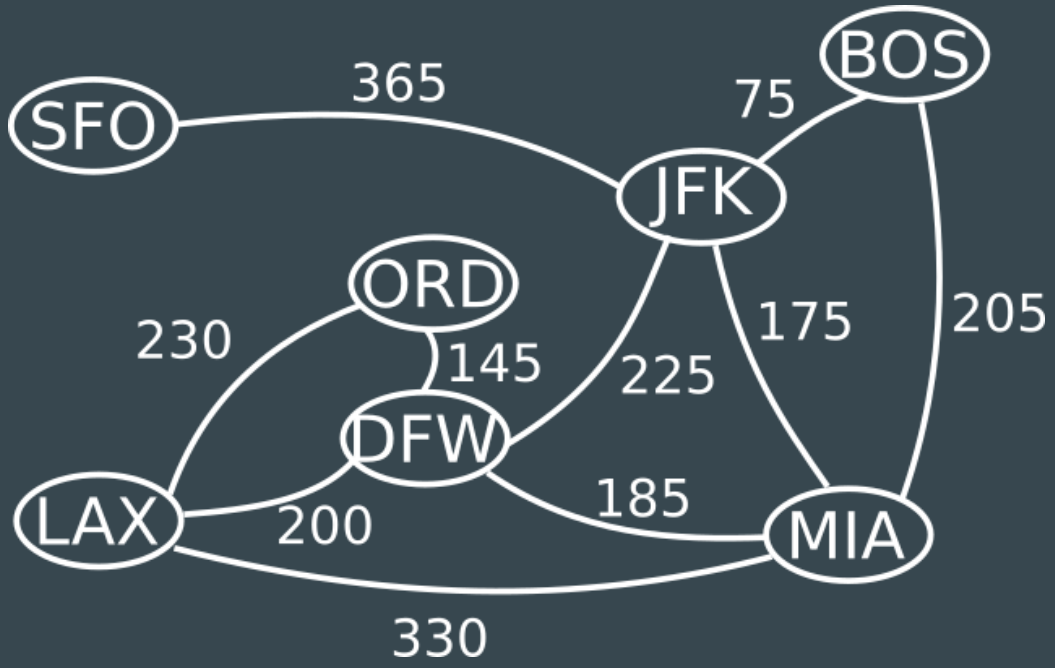
- Un árbol recubridor contiene **todos los nodos** de un grafo **no dirigido** y **conectado**.
- Si el grafo posee ***n*** nodos, el árbol recubridor posee ***n*-1** aristas.
- El **árbol recubridor mínimo** es el árbol con **menor peso total**.



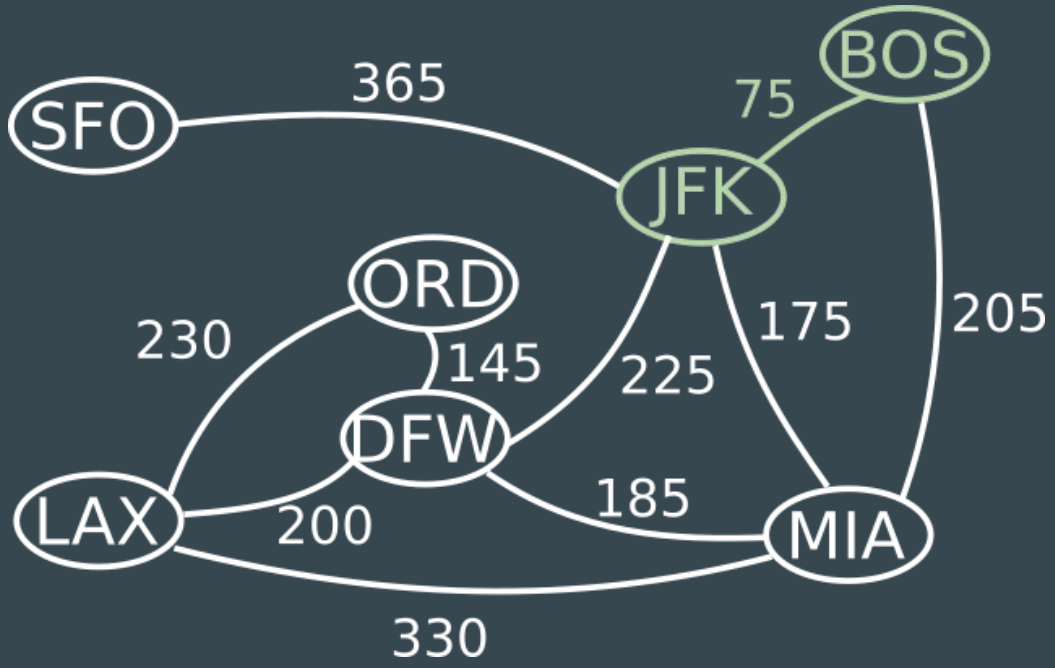
Algoritmo de Kruskal

- Es un algoritmo **ávido**.
- La **heurística** en la que se basa es seleccionar el arco de menor peso,
- A medida que se seleccionan nodos, se van formando distintos subárboles, que serán combinados en un **árbol final**.
- Si la arista de menor peso conecta nodos ya presentes en un mismo árbol, se descarta
- El algoritmo termina cuando se forma un solo árbol de *n*-1 aristas.

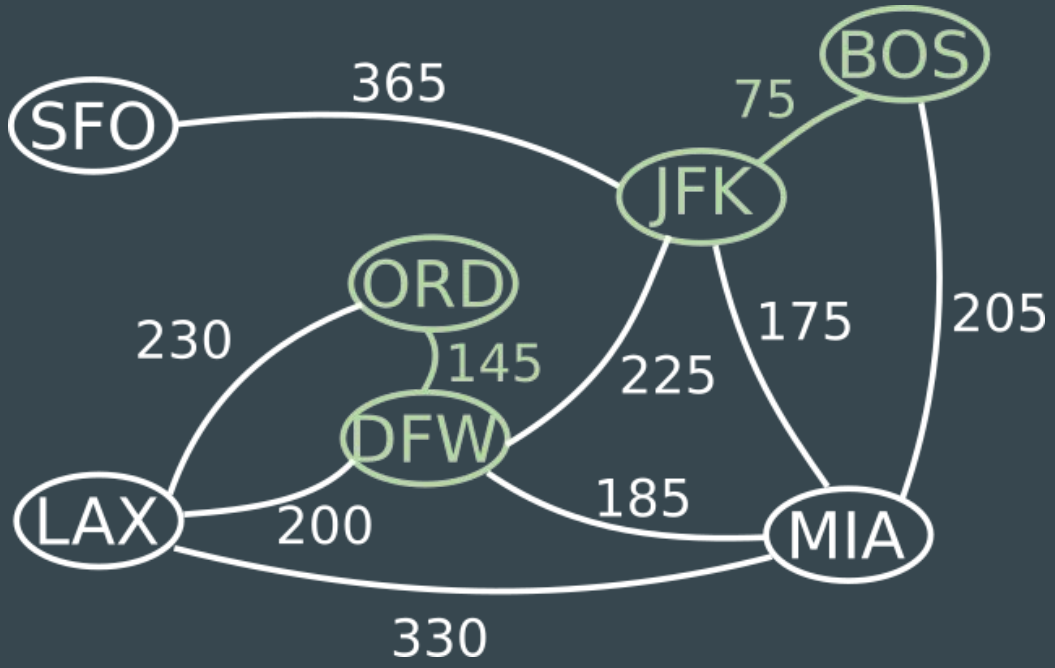
Algoritmo de Kruskal



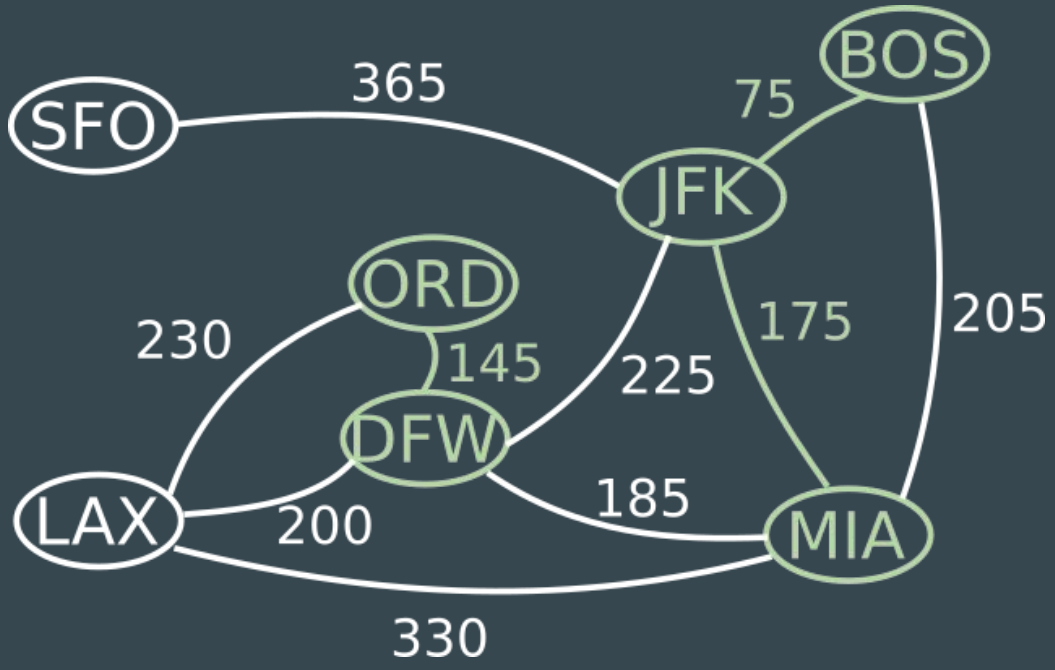
Algoritmo de Kruskal



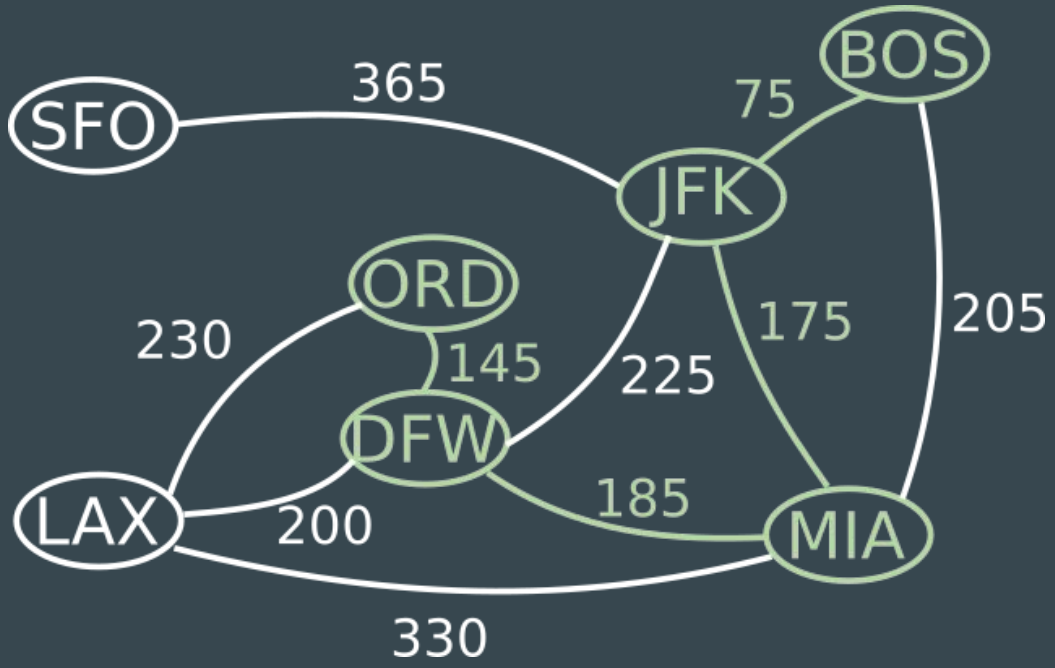
Algoritmo de Kruskal



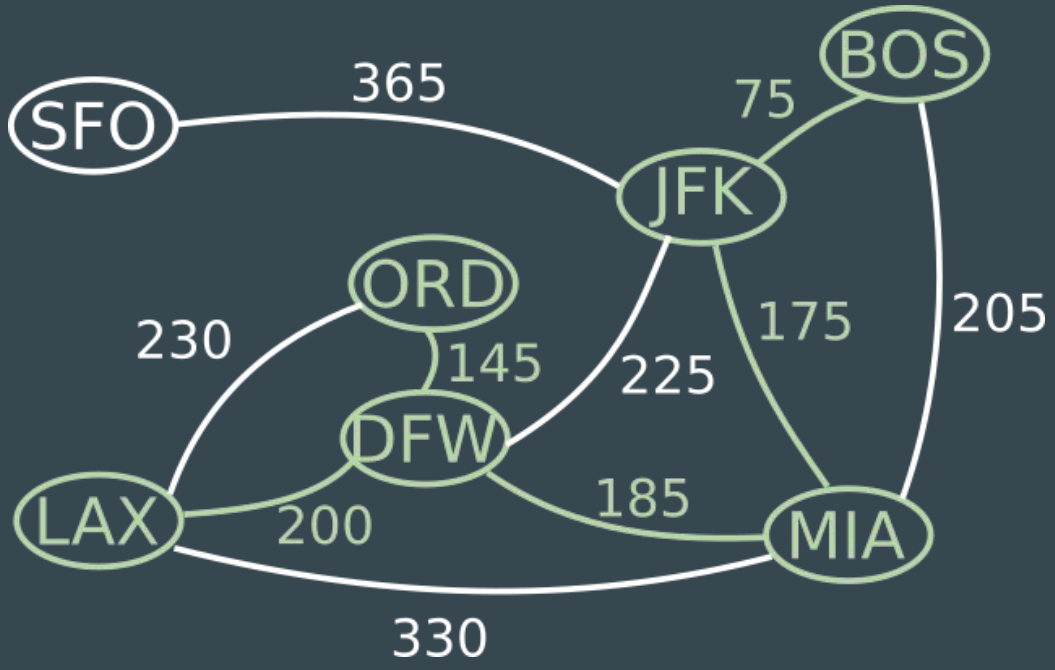
Algoritmo de Kruskal



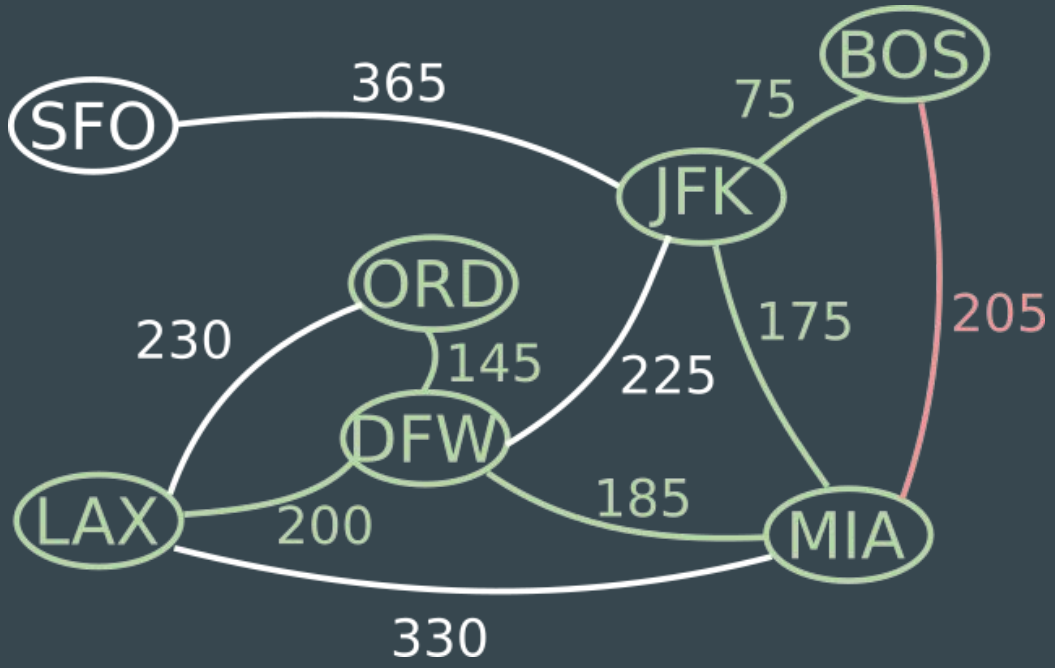
Algoritmo de Kruskal



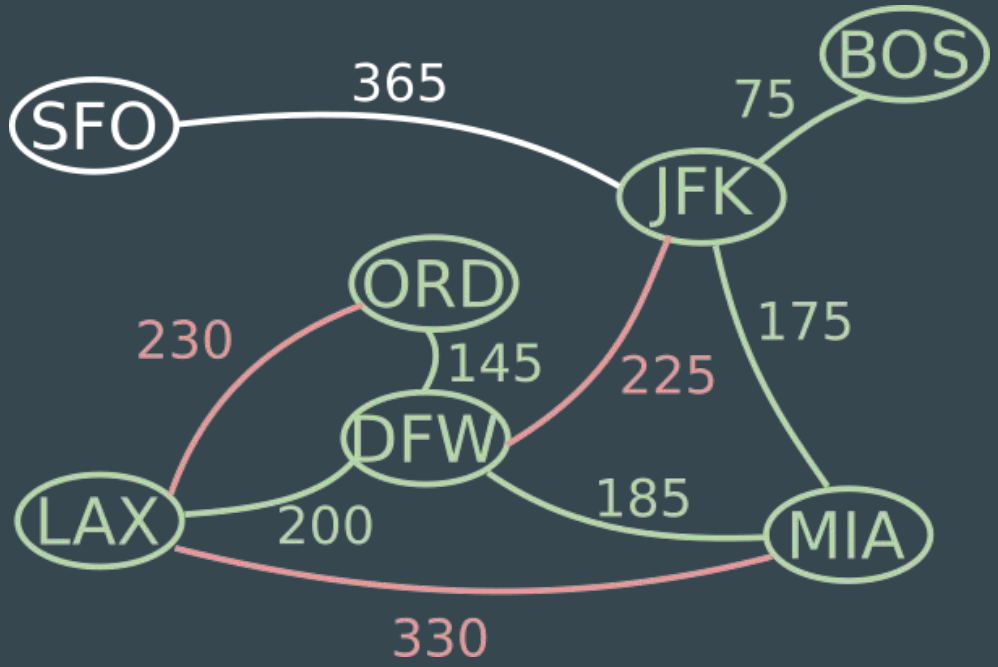
Algoritmo de Kruskal



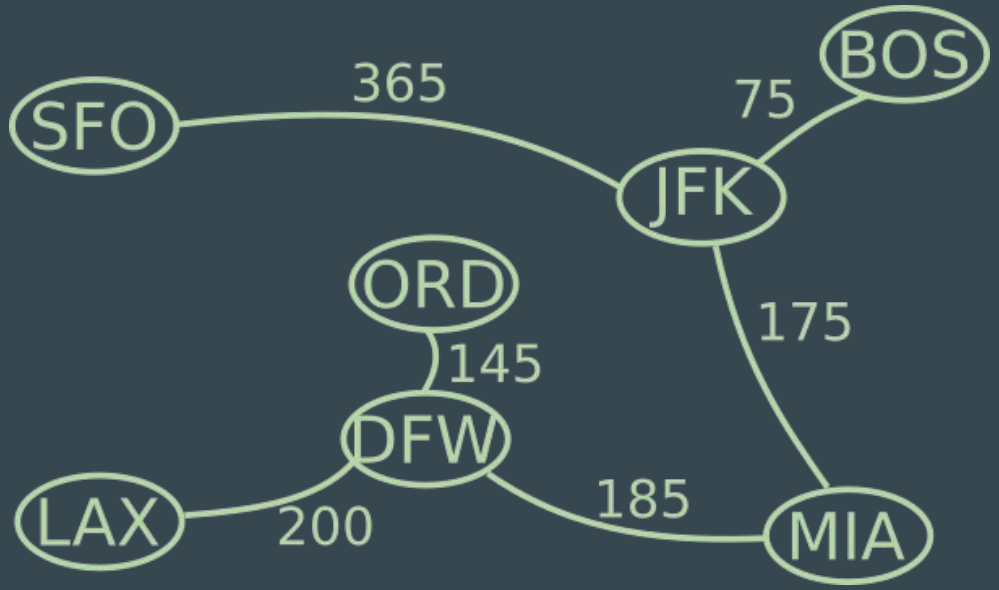
Algoritmo de Kruskal



Algoritmo de Kruskal



Algoritmo de Kruskal



Algoritmo de Kruskal: pseudocódigo

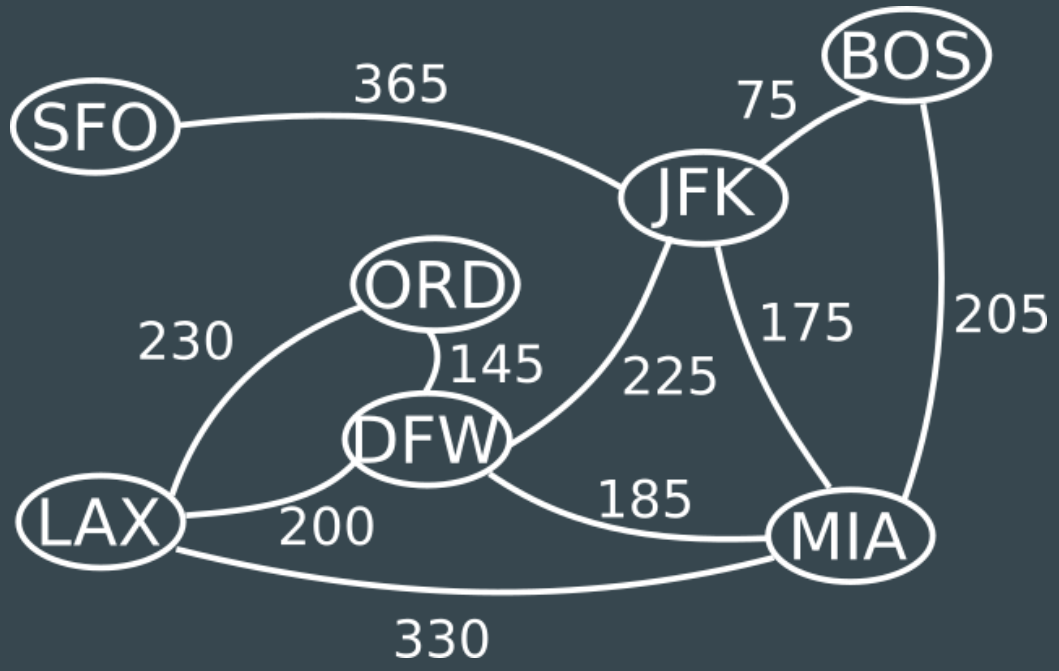
Entrada: grafo, no dirigido, ponderado

armar tabla de pesos, ordenada
mientras que el número de aristas en el árbol != n-1:
    elegir arista de menor peso, quitar de la lista
    si los nodos no están en ningún subárbol:
        crear subárbol
    si están en distintos subárboles:
        combinar subárboles
    si están en el mismo árbol:
        descartar arista

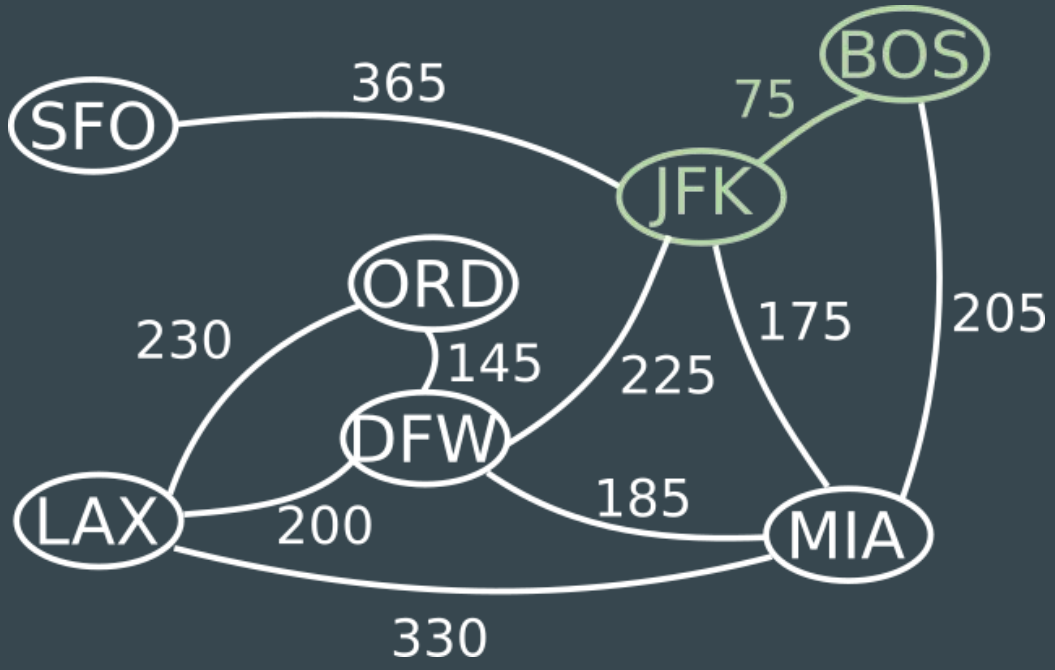
Algoritmo de Prim

- También es un algoritmo ávido.
- Se comienza eligiendo la **arista de menor peso**, y se continúa con la arista de menor peso conectada a los nodos **ya visitados**.
- Si la arista de menor peso lleva a un nodo ya visitado, se descarta.
- El algoritmo termina cuando todos los nodos fueron visitados

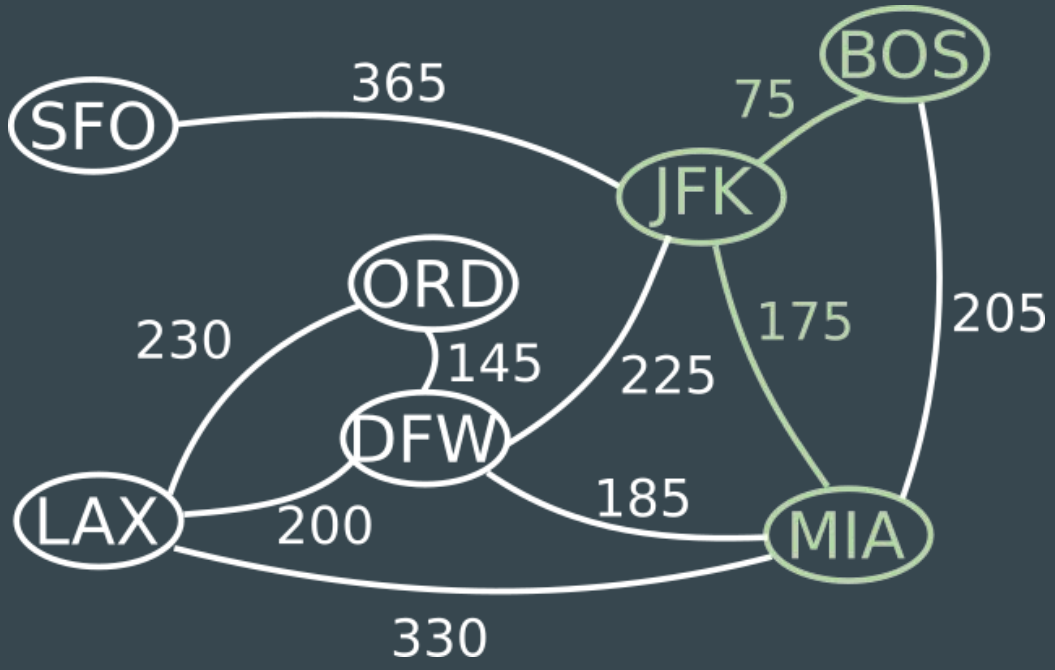
Algoritmo de Prim



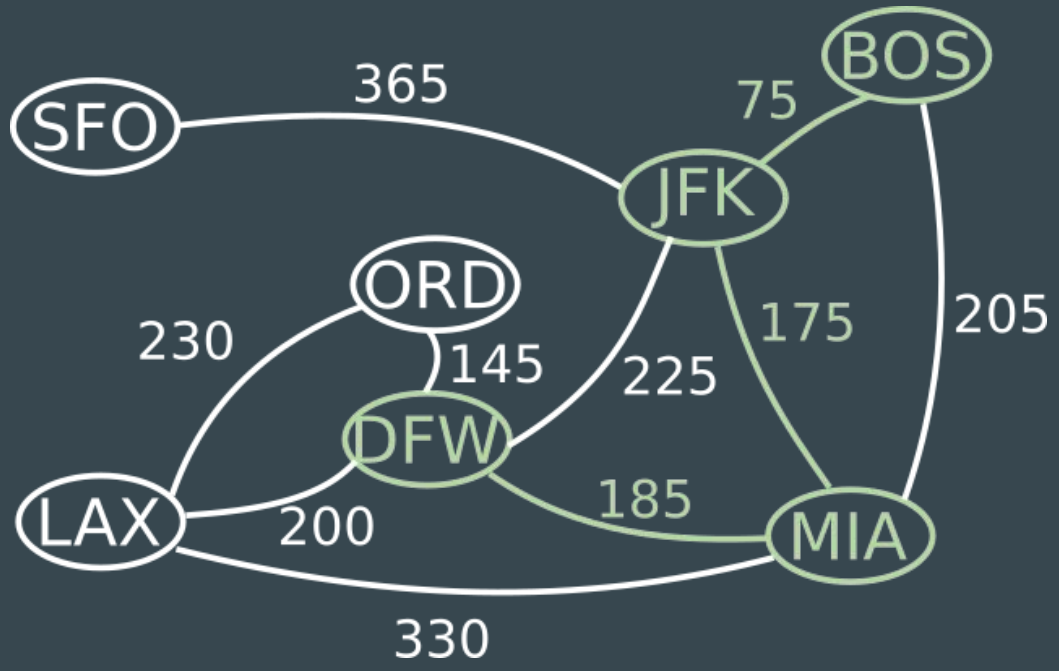
Algoritmo de Prim



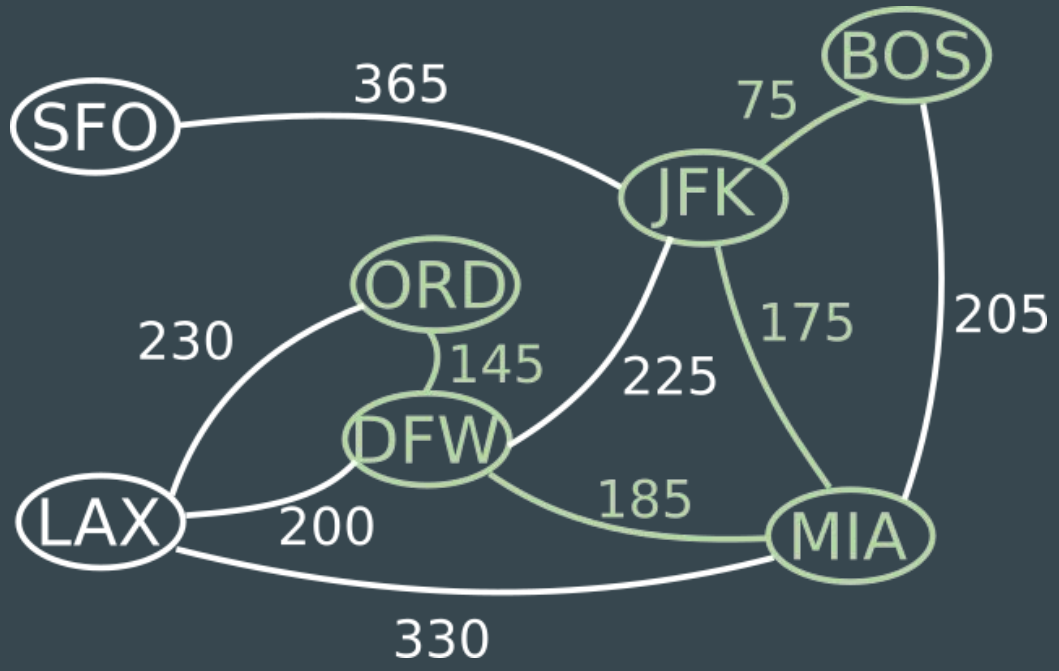
Algoritmo de Prim



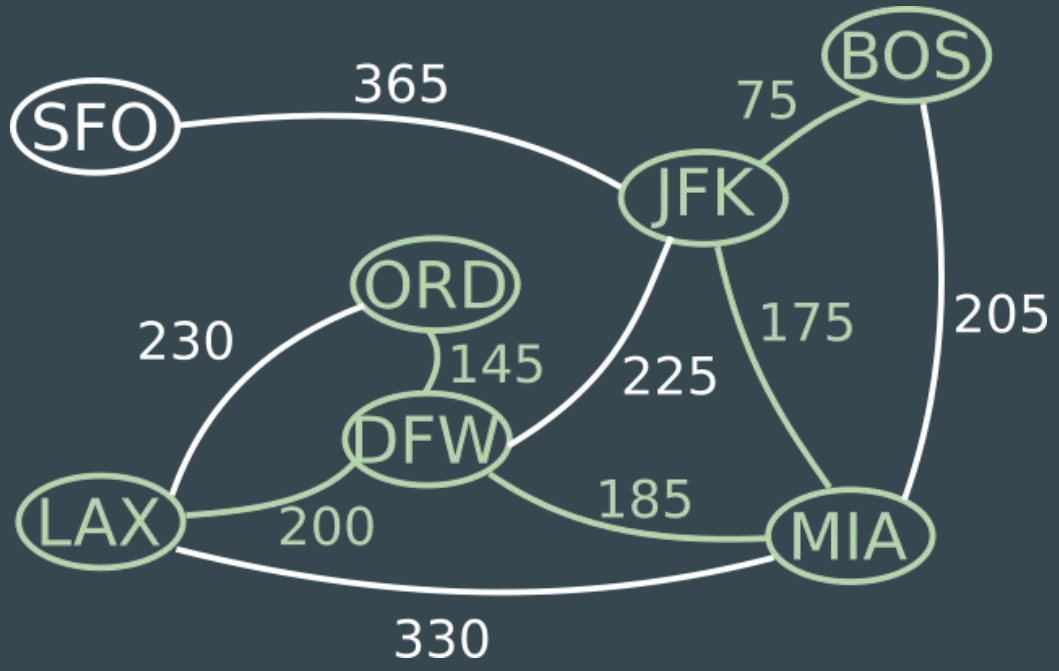
Algoritmo de Prim



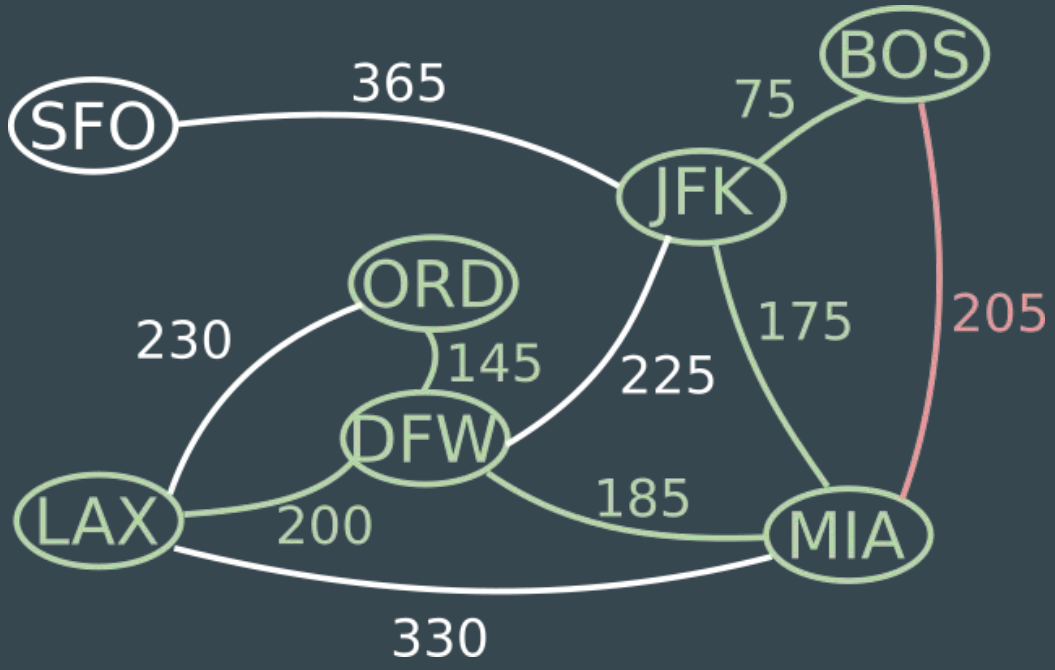
Algoritmo de Prim



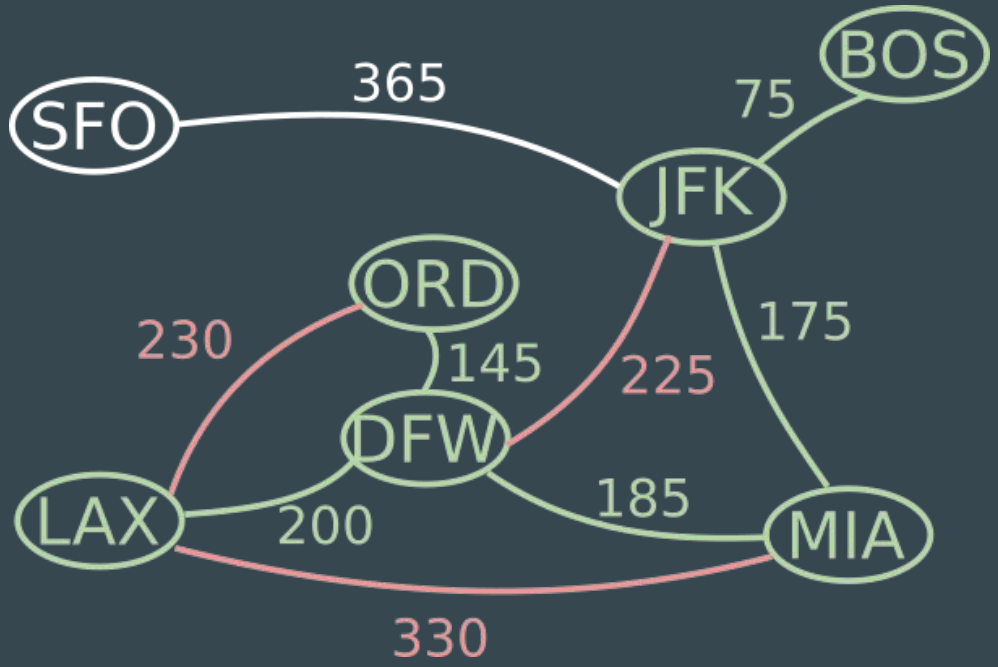
Algoritmo de Prim



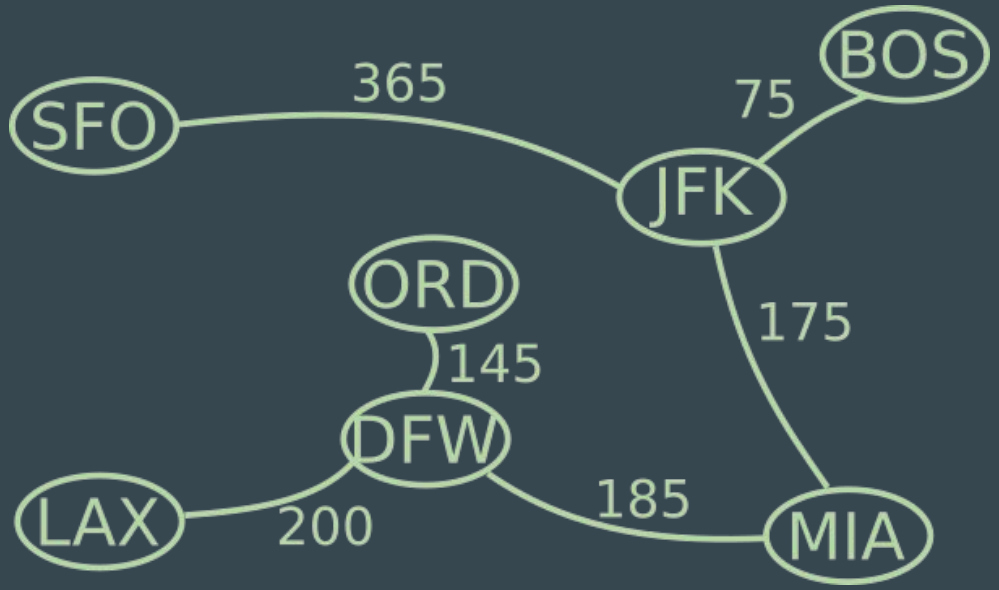
Algoritmo de Prim



Algoritmo de Prim



Algoritmo de Prim



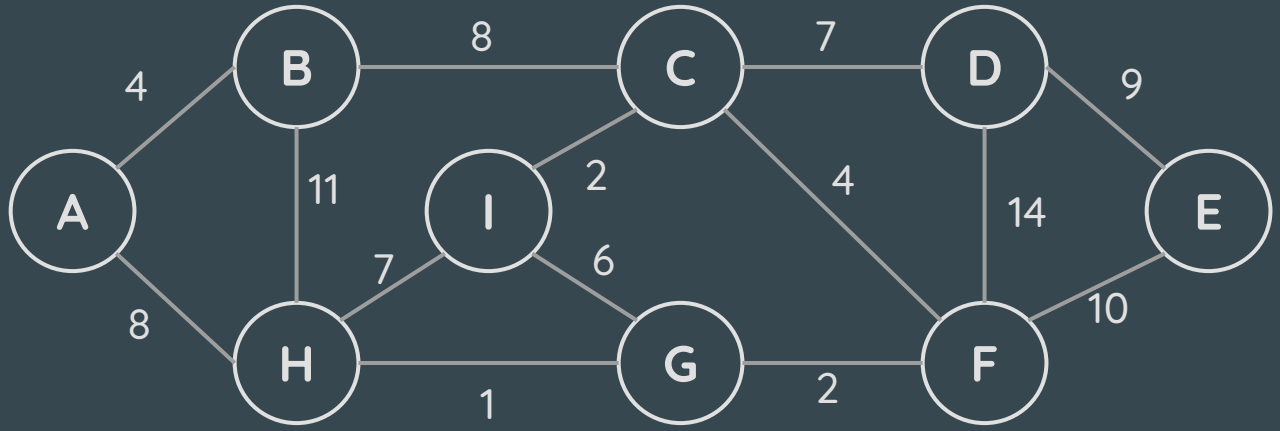
Algoritmo de Prim: pseudocódigo

Entrada: grafo, no dirigido, ponderado

seleccionar **arista de menor peso**
añadir arista la árbol, nodos a la lista de visitados
mientras queden nodos por visitar:
    seleccionar arista de menor peso conectada a nodos visitados
    si el nodo de destino no fue visitado:
        añadir arista al árbol, nodo de destino a lista de visitados

Ejemplo

A partir del siguiente grafo, calcule los árboles recubridores mínimos usando los algoritmos de Kruskal y de Prim



Algoritmo de Kruskal: código

```
from operator import itemgetter

def kruskal (grafo):
  arm = {}
  arboles = []
  aristas = 0
  lista_ordenada = ordenar_lista(grafo)
  while aristas != len(grafo)-1:
    inicio, fin, peso = lista_ordenada.pop(0)
    agregar = anadir_nodos(arboles, inicio, fin)
    if agregar == True:
      print(inicio, fin, peso) # No es necesario, sirve para ver el orden
      aristas += 1
      if inicio not in arm:
        arm[inicio] = {}
      arm[inicio][fin] = peso
  return arm
```

Algoritmo de Kruskal: código

```
def ordenar_lista (grafo):
  resultado = []
  for i in grafo:
    for f in grafo[i]:
      resultado.append((i, f, grafo[i][f]))
      resultado.sort(key=itemgetter(2))
  return resultado
```

Algoritmo de Kruskal: código

```
def anadir_nodos(arboles, inicio, fin):
  indice_inicio = indice_fin = None
  for i in range(0, len(arboles)):
    if inicio in arboles[i]:
      indice_inicio = i
    if fin in arboles[i]:
      indice_fin = i

  if indice_inicio == None and indice_fin != None:
    arboles[indice_fin].append(inicio)
    return True
  elif indice_inicio != None and indice_fin == None:
    arboles[indice_inicio].append(fin)
    return True
  elif indice_inicio == None and indice_fin == None:
    arboles.append([inicio, fin])
    return True
  elif indice_inicio != None and indice_fin != None and indice_inicio != indice_fin:
    arboles.append(arboles[indice_inicio]+arboles[indice_fin])
    quitar_inicio = arboles[indice_inicio]
    quitar_fin = arboles[indice_fin]
    arboles.remove(quitar_inicio)
    arboles.remove(quitar_fin)
    return True
```

Algoritmo de Prim: código

```
def prim (grafo):
  arm = {}
  visitados = []
  while len(visitados) < len(grafo):
    inicio, fin, peso = encontrar_arista_menor(grafo, visitados)
    print(inicio, fin, peso) # No es necesario, sirve para ver el orden

    if inicio not in visitados:
      visitados.append(inicio)
    if fin not in visitados:
      visitados.append(fin)

    if inicio not in arm:
      arm[inicio] = {}
    arm[inicio][fin] = peso
  return arm
```

Algoritmo de Prim: código

```
def encontrar_arista_menor (grafo, visitados):
  nodo_inicio = ""
  nodo_fin = ""
  menor_peso = float("inf")
  if len(visitados) == 0:
    for inicio in grafo:
      for fin in grafo[inicio]:
        if grafo[inicio][fin] < menor_peso:
          menor_peso = grafo[inicio][fin]
          nodo_inicio = inicio
          nodo_fin = fin
  else:
    for inicio in visitados:
      for fin in grafo[inicio]:
        if grafo[inicio][fin] < menor_peso and fin not in visitados:
          menor_peso = grafo[inicio][fin]
          nodo_inicio = inicio
          nodo_fin = fin

  return nodo_inicio, nodo_fin, menor_peso
```

Bibliografía

- Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2013). Data structures and algorithms in Python. Capítulo 14.7
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to algorithms. MIT press. Capítulo 23