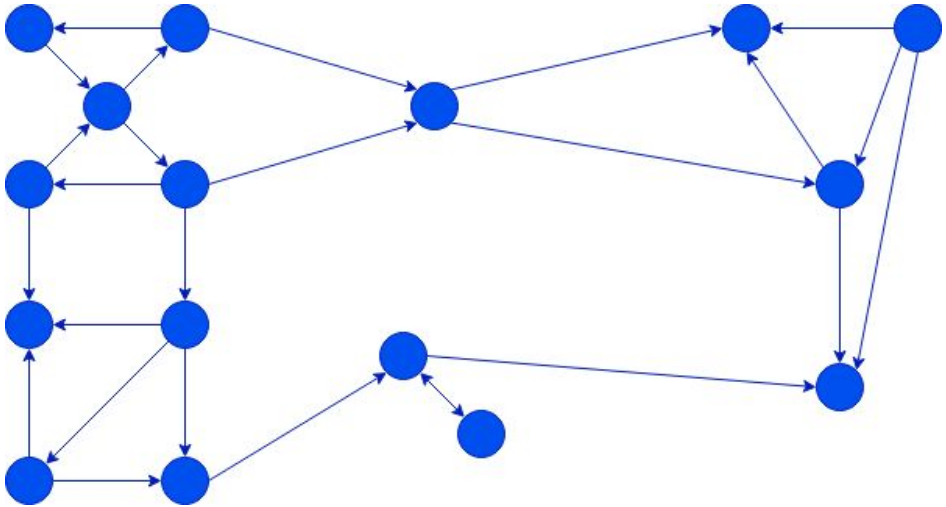
Strongly Connected Component

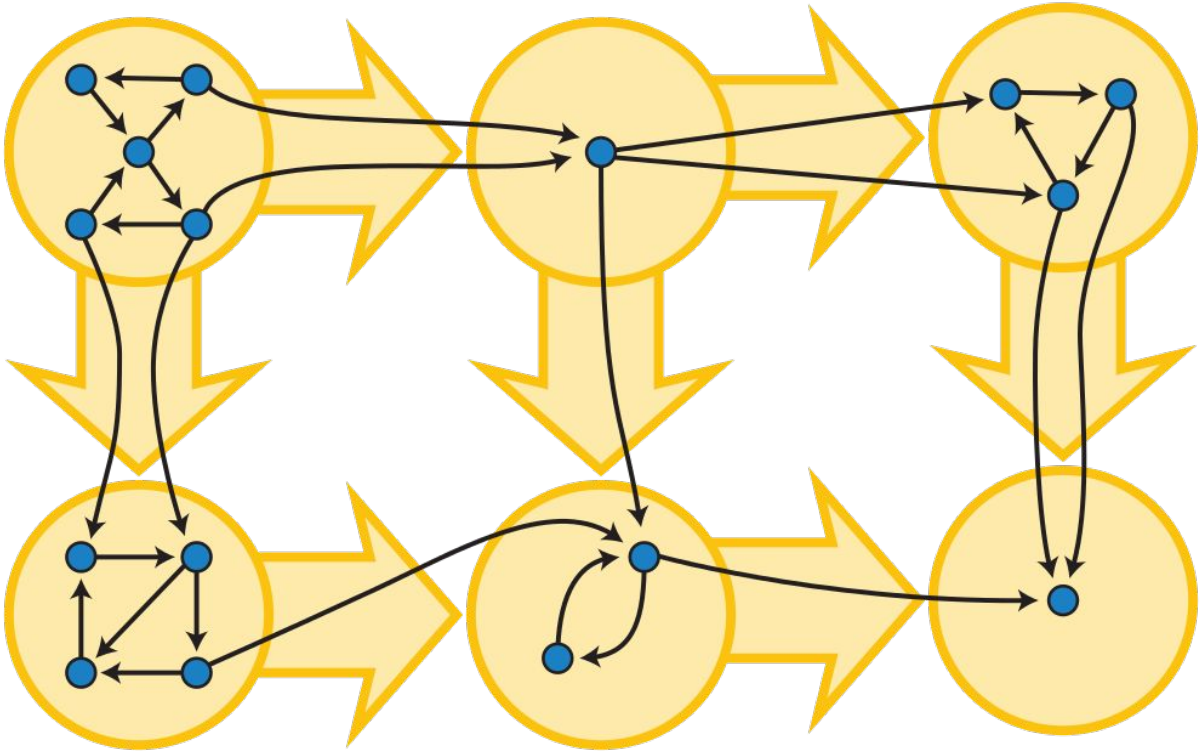
Bruno Aguilar
Ayrton Mera
Jaime Olguin

Kosaraju's algorithm



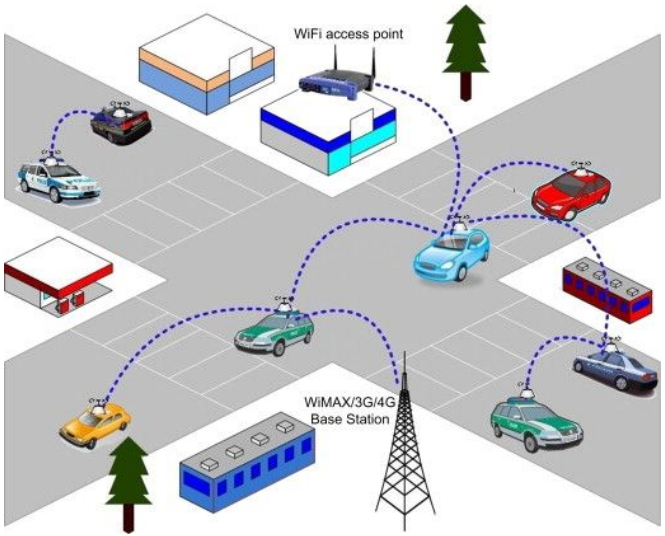
1. Crear una pila vacía S
2. DFS transversal en el grafo.
3. Invertir la dirección de los arcos para obtener el grafo transpuesto

Kosaraju's algorithm



¿Cuál es la utilidad del método?

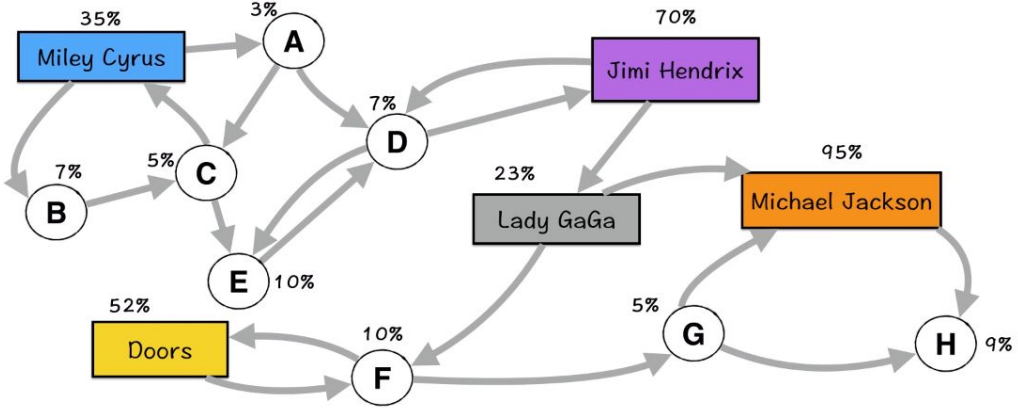
Uso general para encontrar grupos de entidades altamente relacionados:

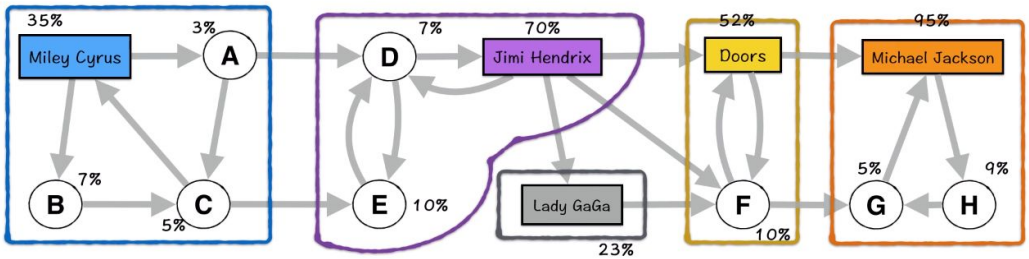


Ejemplos:

- Reconocimiento de grandes corporaciones trasnacionales.
- Configuración de redes inalámbricas de saltos múltiples **(Wireless multihop networks).**
- Uso en redes sociales, buscando ciertas preferencias de grupos desconocidos.
- Plus de entrevistas.

Comparado con otras estructura o métodos ¿Cuáles son sus ventajas y desventajas?





Se desempeña mejor que:

- Resolución y Clausura Transitiva
- Backtracking Limitado

Desventajas:

- Restringido
- Paralelismo ineficiente

¿Existen desafíos relacionados?¿Se puede mejorar el método? ¿Dónde habría que enfocarse?

Desafíos:

Flexibilización

Mejoras:

Algoritmo de Tarja

¡Código en ejecución!

Clase Graph:

```
class Graph:
    def __init__(self,vertices):
        self.V= vertices
        self.graph = defaultdict(list)

    def addEdge(self,u,v):
        self.graph[u].append(v)
```

```
g = Graph(16)
g.addEdge(0,2)
g.addEdge(2,1)
g.addEdge(1,0)
g.addEdge(3,2)
g.addEdge(2,4)
g.addEdge(4,3)
g.addEdge(1,5)
g.addEdge(4,5)
g.addEdge(5,6)
g.addEdge(5,8)
g.addEdge(6,7)
g.addEdge(7,8)
g.addEdge(8,6)
g.addEdge(8,9)
g.addEdge(7,9)
g.addEdge(10,9)
g.addEdge(10,11)
g.addEdge(11,10)
g.addEdge(5,10)
g.addEdge(12,13)
print ("Following are strongly connected components " +
                                "in given graph")
g.printSCCs()
```

Consultas?