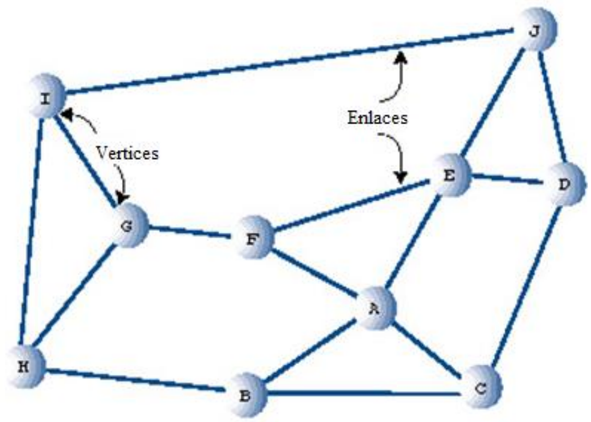
Grafos

Estructuras de Datos

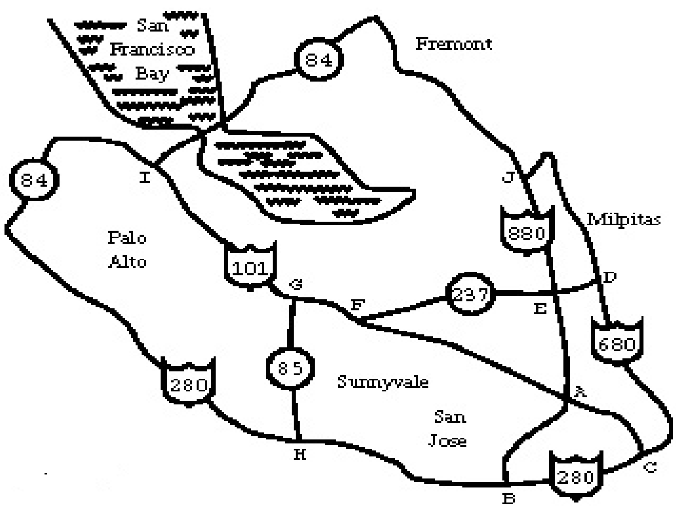
Grafos

- **Los grafos son una estructura de datos que representa relaciones entre entidades**
  - **Los *Vértices* representan entidades**
  - **Los *Enlaces* representan alguna clase de relación**



Ejemplo

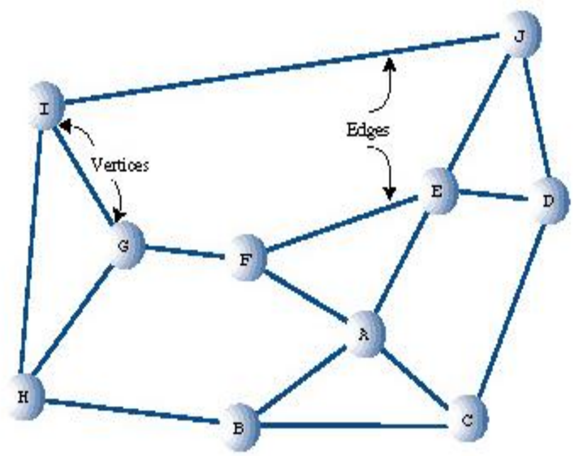
- **El grafo de la diapositiva anterior podría ser usando para modelar las carreteras entre ciudades:**



Adyacencia

- **Dos vértices son *adyacentes* uno al otro si están conectados por un enlace simple**

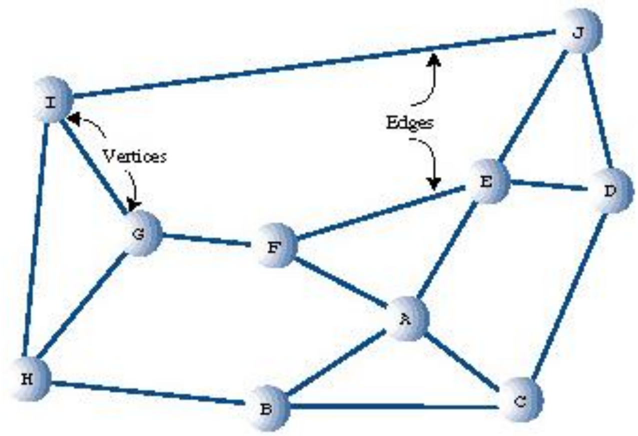
- **Por ejemplo:**
  - **I y G son adyacentes**
  - **A y C son adyacentes**
  - **I y F no son adyacentes**



- **Dos nodos adyacentes son considerados *vecinos***

Camino

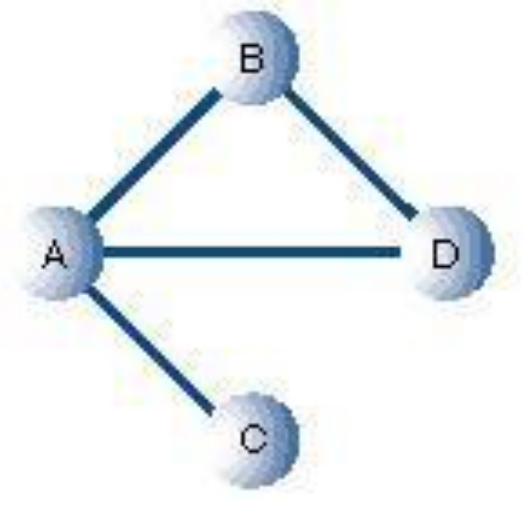
- **Un *camino* es una secuencia de enlaces**



- **Los caminos en este grafo incluyen a:**
  - **BAEJ, CAFG, HGFEJDCAB, HIJDCBAFE, etc.**

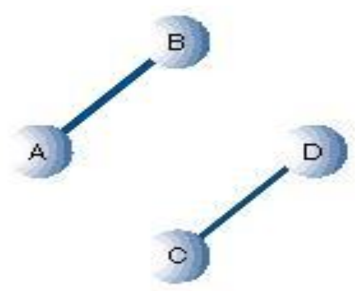
Grafos Conectados

- **Un grafo está conectado si hay al menos un camino de cada vértice a cada otro vértice**



Grafo No conectado

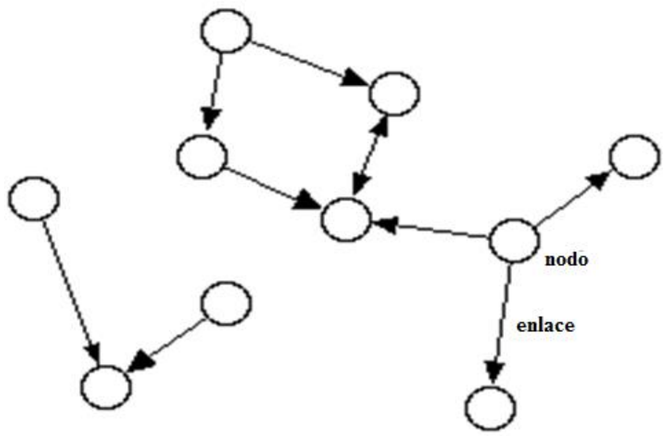
- **Un *grafo no conectado* consiste de varias *componentes conectadas*:**



- **Componentes conectados de este grafo son:**
  - **AB, y CD**
- **Trabajaremos con grafos conectados**

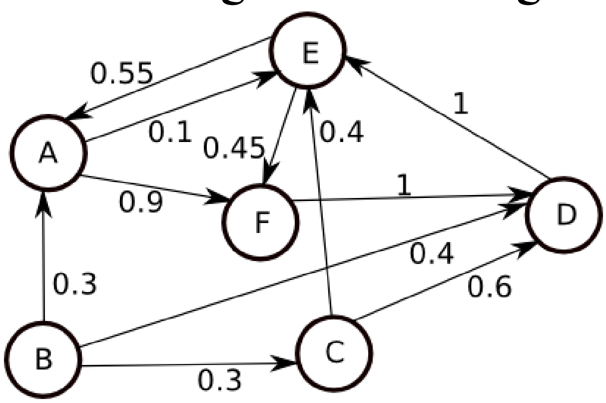
Grafo Dirigido

- **Un grafo donde los enlaces tienen direcciones**
  - **Por lo general, están designados por una flecha**



Grafos Ponderados

- **Un grafo donde cada enlace tiene un peso, el cual cuantifica la relación**
  - **Por ejemplo, puede asignar distancias entre las ciudades**
  - **O los costos de las aerolíneas**
- **Estos grafos pueden ser dirigidos o no dirigidos**



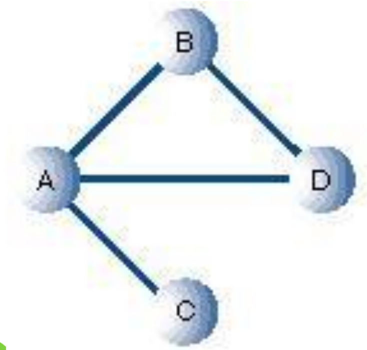
Vértices: Implementación Java

- **Representar un vértice como una clase Java con:**
  - **Un String**
  - **Un atributo booleano para comprobar si se ha visitado**

- **Para especificar los enlaces**
  - **hacer esto con una *matriz* de adyacencia o una *lista* de adyacencia.**

Matriz de Adyacencia

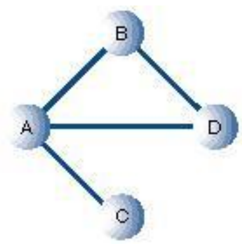
- **Una matriz de adyacencia de un grafo con n nodos, es de tamaño n x n**
  - **La posición (i, j) contiene un 1 si hay un enlace de conexión del nodo i al nodo j**
  - **cero en caso contrario**
- **Por ejemplo, aquí está un grafo y su matriz de**



	A	B	C	D
A	0	1	1	1
B	1	0	0	1
C	1	0	0	0
D	1	1	0	0

¿Redundancia?

- **Esto puede parecer un poco redundante:**

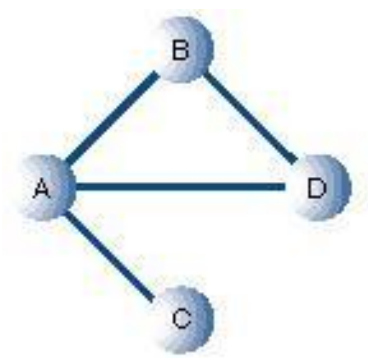


	A	B	C	D
A	0	1	1	1
B	1	0	0	1
C	1	0	0	0
D	1	1	0	0

- **¿Por qué almacenar dos piezas de información para el mismo enlace?**
  - **es decir, (A, B) y (B, A)**
- **Desafortunadamente, no hay una manera fácil de evitar esto**
  - **Debido a que los enlaces no tienen una dirección**
  - **No hay concepto de "padres" e "hijos"**

Lista de Adyacencia

- **Una serie de listas enlazadas**
  - **Indexado por vértice, y proporciona una lista enlazada de los vecinos**
- **Este es el mismo gráfico, con su lista de adyacencia:**



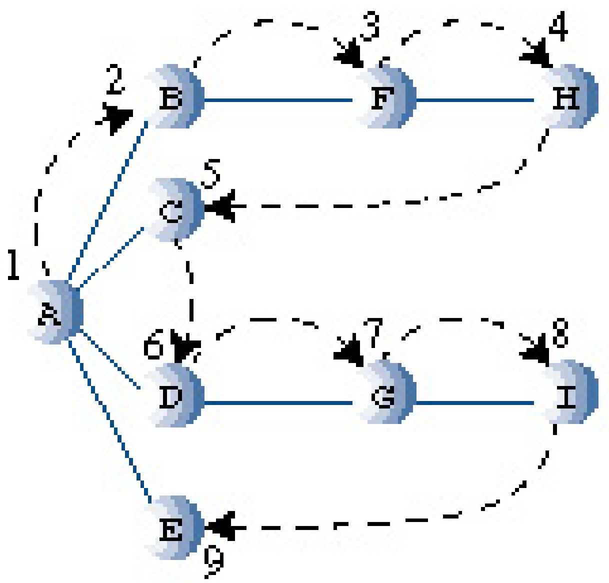
Vertice	Lista que contiene Vertices Adyacentes
A	B—>C—>D
B	A—>D
C	A
D	A—>B

Aplicación: Búsquedas

- Es una operación fundamental para un grafo:
  - A partir de un vértice en particular
  - Buscar todos los otros vértices que puede ser alcanzados siguiendo los caminos
- Ejemplo de aplicación
  - ¿Cuántas ciudades en la región del Bío-Bío se pueden alcanzar en tren desde Concepción?
- Dos enfoques
  - Búsqueda en Profundidad (DFS)
  - Búsqueda en Amplitud (BFS)

Búsqueda en Profundidad (DFS)

- **Idea**
  - **Elija un punto de partida**
  - **Siga un camino hacia los vértices no visitados, siempre que sea posible hasta llegar a un camino sin salida**
  - **Al llegar a un camino sin salida, vuelva al punto anterior y siga con los vértices no visitados**
  - **Deténgase cuando cada camino sea sin salida**

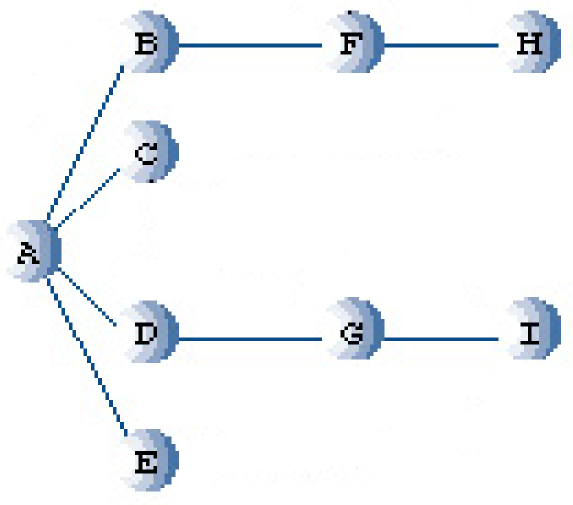


Búsqueda en Profundidad(DFS)

- **Algoritmo**
  - **Escoja un vértice (llámelo A) como su punto de partida**
  - **Visite este vértice, y:**
    - **Empújelo (Push) en una pila de vértices visitados**
    - **Márquelo como visitado (para evitar que lo visitemos nuevamente después)**
  - **Visite cualquier vecino de A que no haya sido visitado**
    - **Repita el proceso**
  - **Cuando A no tenga más vecinos no visitados**
    - **Sáquelo (Pop) de la pila**
  - **Termine cuando la pila este vacía**
- **Nota: Llegamos tan lejos desde el punto de partida hasta llegar a un camino sin salida, y luego pop (se puede aplicar a laberintos)**

Ejemplo

- **Parta en A, y ejecute una búsqueda en profundidad en este grafo y muestre en cada paso el contenido de la pila**
  - **En cada paso, tenemos que hacer una visita o un pop**



Búsqueda en Profundidad: Complejidad

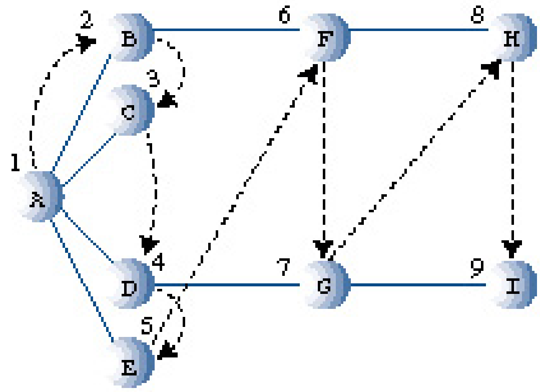
- **Sea |V| el número de vértices en un grafo**
- **Y sea |E| el número de enlaces**

- **En el peor caso, visitamos cada vértice y cada enlace:**
  - **Tiempo O(|V| + |E|)**

- **En una primera mirada, esto no luce mal**
  - **Pero recuerde que los grafos tienen ¡muchos enlaces!**
  - **Peor caso, cada vértice está conectado el resto:**
    - **$(n-1) + (n-2) + \ldots + 1 = O(n^2)$**
  - **Se torna muy caro si el grafo tiene muchos enlaces**

Búsqueda en Amplitud (BFS)

- **La misma aplicación que DFS; queremos encontrar todos los vértices que podamos conseguir a partir de un punto de partida, llamado A**
- **Sin embargo esta vez, en vez de ir tan lejos como sea posible hasta que nos encontramos con un camino sin salida, como es DFS**
  - **Visitaremos primero todos los vértices más cercanos**
  - **Una vez que hemos visitados todos los vértices más cercanos, nos alejaremos por un enlace a un vecino**

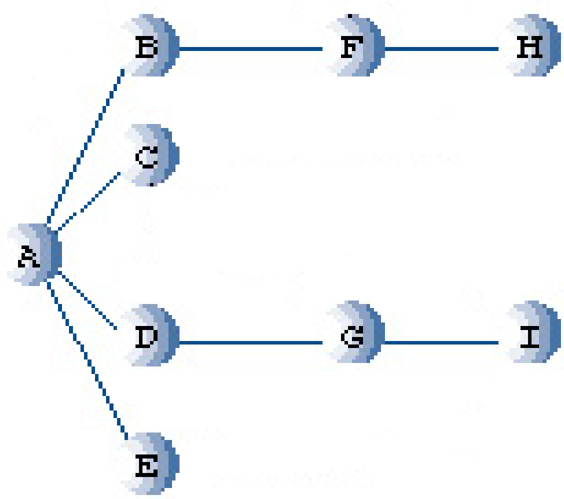


Búsqueda en Amplitud (BFS)

- **¡Vamos a usar una cola en vez de un a pila!**
- **Algoritmo**
  - **Parta en un vértice, llámelo current**
  - **Si hay un vértice no visitado, márquelo, y insértelo en la cola**
  - **Si no hay:**
    - **Si la cola esta vacía, hemos terminado, sino:**
    - **Remueva un vértice de la cola y asigne a current ese vértice y repita el proceso**

Ejemplo

- Parta en A, y ejecute búsqueda por amplitud en este grafo, y muestre el contenido de la cola en cada paso
  - En cada paso, haremos una visita o una eliminación



Búsqueda en Amplitud: Complejidad

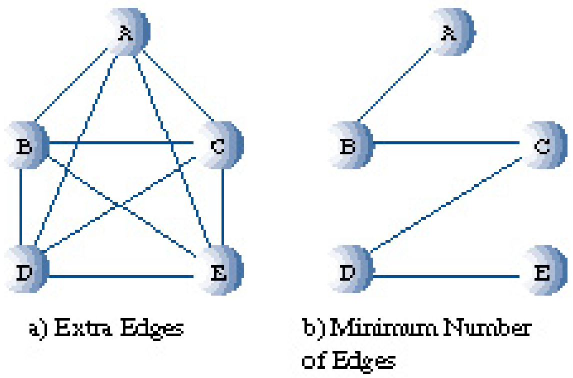
- Sea |V| el número de vértices del grafo
- Y sea |E| el número de enlaces

- En el peor caso, visitaremos cada vértice y cada enlace:
  - Tiempo O(|V| + |E|)
  - Igual que DFS

- Nuevamente, si el grafo tiene muchos enlaces, nos acercaremos a un tiempo de ejecución cuadrático, que es el peor caso

Arboles de Expansión Minima[1] (MSTs)

- **El comentario sobre si un grafo posee un gran número de vértices hace que nuestros valiosos algoritmos de búsqueda se retrasen:**
- **Sería bueno tener un grafo y reducir el número de enlaces al número mínimo requerido para abarcar todos los vértices:**

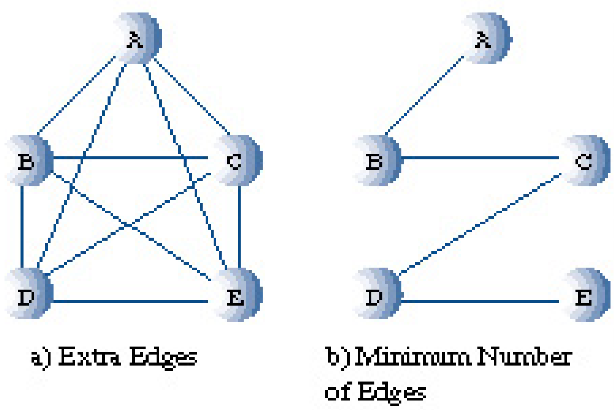


¿Cual es el número de enlaces ahora?

[1]Minimum Spanning Trees

Ya lo hemos hecho...

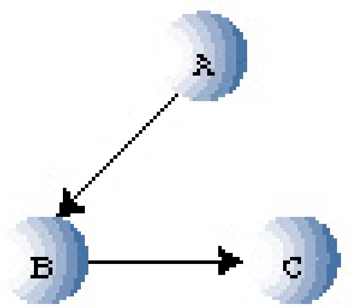
- Realmente, si ejecutamos DFS ya hemos computado el MST!



- Piense sobre esto: siga un mismo camino el mayor tiempo posible, y luego regresas (backtrack) (visite cada nodo a lo más una vez)
  - Tendrás que guardar los enlaces a medida que avances

Grafos Dirigidos

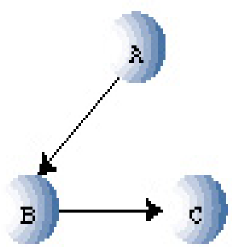
- **Un grafo dirigido es un grafo donde los enlaces tienen direcciones, identificadas por flechas:**



- **Esto simplifica un poco a la matriz de adyacencia…**

Matriz de Adyacencia

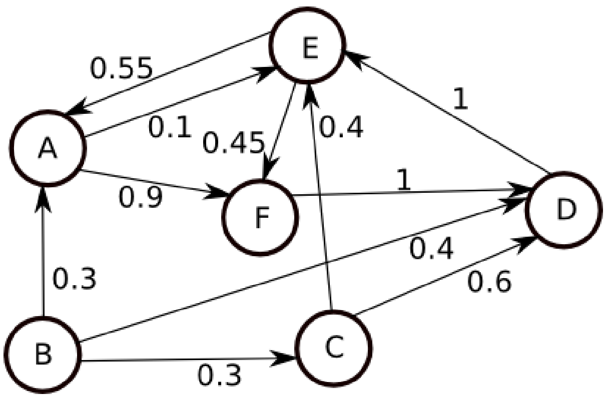
- La matriz de adyacencia para este grafo no contiene entradas redundantes
  - Porque cada enlace tiene un origen y un destino
  - Entonces la entrada (i, j) es 1 si hay un enlace que va desde i a j
    - 0 en caso contrario



	A	B	C
A	0	1	0
B	0	0	1
C	0	0	0

Grafos Ponderados

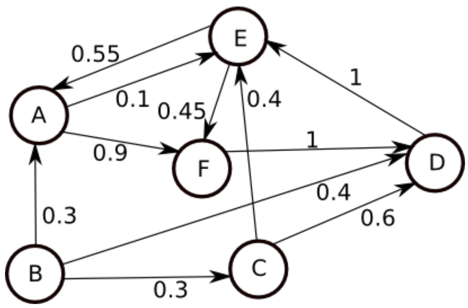
- **Una vez mas, un grafo donde los enlaces tienen pesos, los cuales cuantifican la relación**
- **Estos grafos pueden ser dirigidos y no dirigidos**



Grafos Ponderados

- La lista de adyacencia de un grafo ponderando contiene los pesos de los enlaces
  - En vez de 0 y 1
- Si no hay enlaces que conecten los vértices i y j, se usa un peso INFINITO (¡no 0!)
  - Porque '0' también puede ser un peso
  - También la mayoría de las aplicaciones de grafos ponderados son para encontrar árboles de mínima amplitud o el camino más corto
  - Recuerde también que si un grafo es no dirigido, la información redundante debe ser almacenada

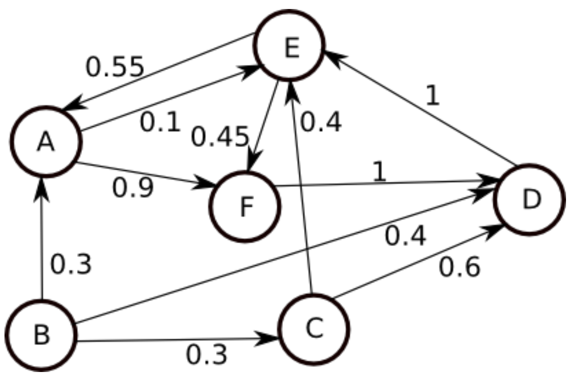
Grafos Ponderados



	A	B	C	D	E	F
A	INF	INF	INF	INF	0.1	0.9
B	0.3	INF	0.3	0.4	INF	INF
C	INF	INF	INF	0.6	0.4	INF
D	INF	INF	INF	INF	1	INF
E	0.55	INF	INF	INF	INF	0.45
F	INF	INF	INF	1	INF	INF

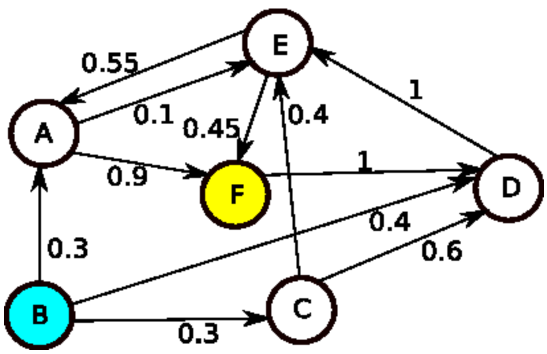
Algoritmo de Dijkstra

- **Dado un grafo ponderado, encuentre el camino más corto (en termino de los pesos de sus enlaces) entre dos vértices del grafo**
- **Numerosas aplicaciones**
  - **El ticket de avión más barato entre dos ciudades**
  - **La distancia de manejo más corta en términos de consumo de litros de gasolina**



Algoritmo de Dijkstra

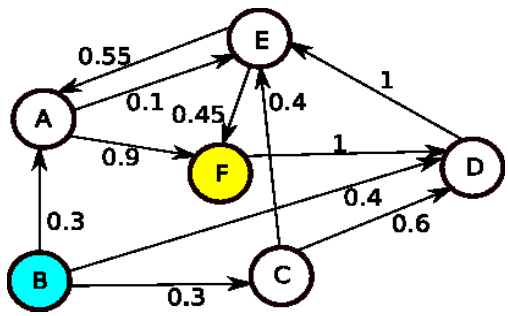
- **Deseamos encontrar el camino más corto entre B y F en el siguiente grafo:**



- **Idea: Mantenga una tabla de los caminos más cortos actuales desde B a todos los otros vértices (y las rutas que toman)**
  - **Cuando la termine, la tabla tendrá los caminos más cortos desde B a todos los otros vértices**

Algoritmo de Dijkstra

- **Este es nuestro grafo inicial:**

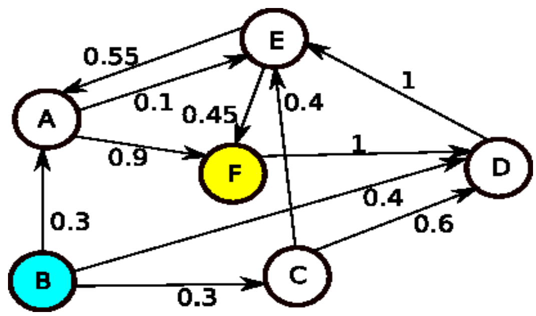


- **Aquí esta nuestra tabla:**

A	C	D	E	F
INF	INF	INF	INF	INF

Paso 1

- **Tome todos los enlaces que salen de B, y ponga sus pesos en la tabla con el vértice fuente**

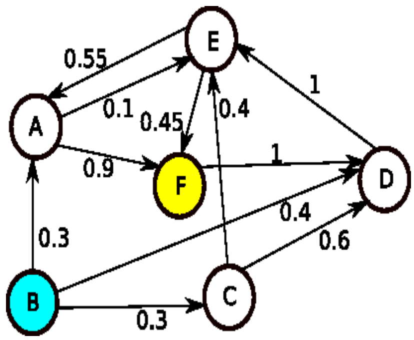


- **Aquí esta nuestra tabla:**

A	C	D	E	F
0.3 (B)	0.3 (B)	0.4 (B)	INF	INF

Paso 2

- **Escoja el enlace con el peso más pequeño y márquelo como el camino más corto de B (¿Como sabemos eso?)**

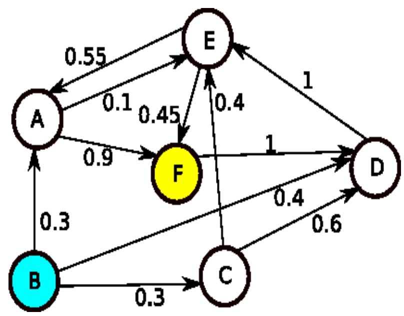


- **Aquí esta nuestra tabla:**

A	C	D	E	F
0.3* (B)	0.3 (B)	0.4 (B)	INF	INF

Paso 3

- **Ahora escoja uno de los enlaces con mínimo costo y repita el proceso (explore vértices ady. y marque su peso total)**

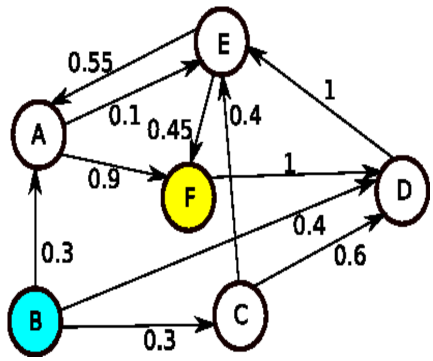


- **Aquí esta nuestra tabla:**

A	C	D	E	F
0.3* (B)	0.3 (B)	0.4 (B)	INF	INF

Paso 4

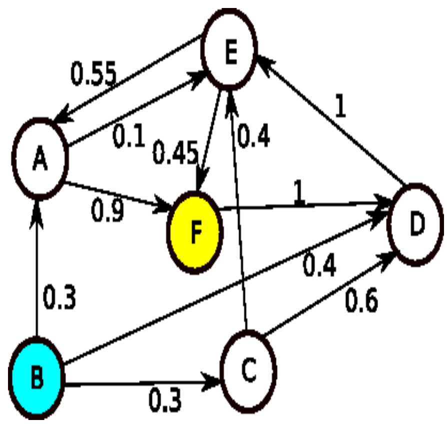
- **En este caso, veamos A**
  - **Explore vértices adyacentes**
  - **Ingrese el peso total desde B a estos vértices**
    - **Si ese peso es menor que**
    - **la entrada actual en la tabla**
    - **Ignore los marcados (*)**
- **Aquí esta nuestra tabla:**



A	C	D	E	F
0.3* (B)	0.3 (B)	0.4 (B)	0.4 (A)	1.2 (A)

Paso 5

- **Ahora, A esta marcado y**
- **Vistamos su vecinos**
  - **Escoja la menor entrada en la tabla (en este caso C) y repita el proceso**

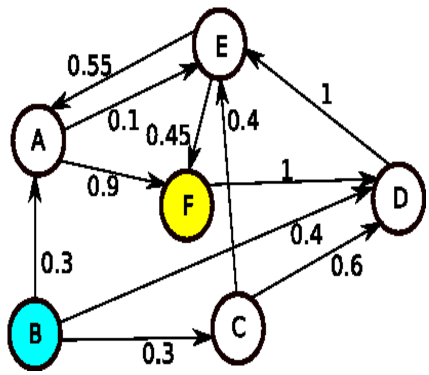


- **Aquí esta nuestra tabla:**

A	C	D	E	F
0.3* (B)	0.3* (B)	0.4 (B)	0.4 (A)	1.2 (A)

Paso 6

- **Visite los vecinos de C que no están marcados**
  - **Inserte su peso total**
  - **En la tabla, SI es**
  - **Menor que la entrada**
  - **actual**

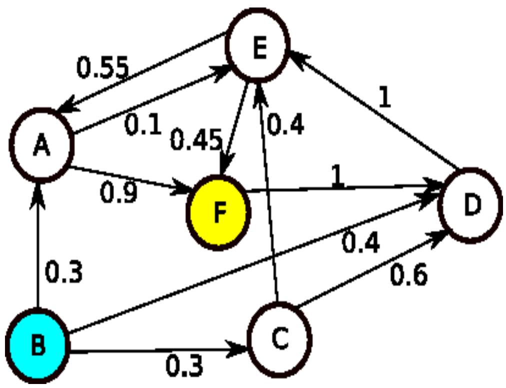


- **En realidad, nada cambia en la tabla:**

A	C	D	E	F
0.3* (B)	0.3* (B)	0.4 (B)	0.4 (A)	1.2 (A)

Paso 7

- **Visitemos D**
  - **Que solo contiene un enlace a E**
  - **0.4+1 = 1.4, el cual**
  - **es más grande que 0.4**

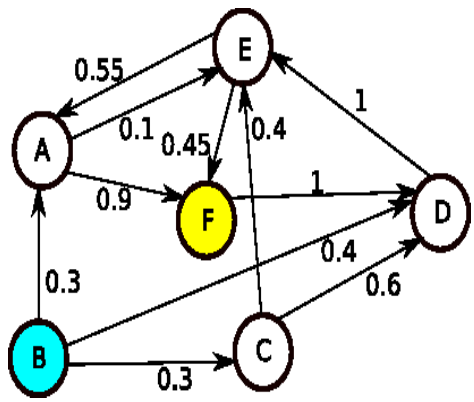


- **Nuevamente, nada cambia en la tabla:**

A	C	D	E	F
0.3* (B)	0.3* (B)	0.4* (B)	0.4 (A)	1.2 (A)

Paso 8

- **Visitemos E**
  - **Tiene dos enlaces salientes**
    - **Uno a A (marcado, ignore)**
    - **Uno a F, el cual cambia**
    - **La tabla a**
      - **0.4 + 0.45 = 0.85**
      - **Que es menor que la**
      - **Entrada actual, 1.2**

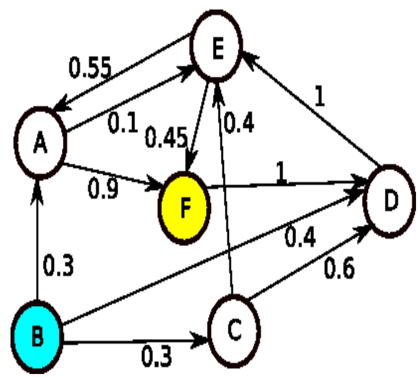


- **La tabla cambia, encontramos un camino más corto a F:**

A	C	D	E	F
0.3* (B)	0.3* (B)	0.4* (B)	0.4* (A)	0.85 (E)

Paso 9

- **Solo un vértice queda**
- **Pero ya hemos terminado**
  - **El camino más corto puede ser**
  - **obtenido partiendo desde**
  - **la entrada destino y**
  - **trabaje hacia atrás**
    - **F <- E <- A <- B**



- **El camino más corto desde B a F es: B -> A -> E -> F**
  - **Peso total: 0.85**

A	C	D	E	F
0.3* (B)	0.3* (B)	0.4* (B)	0.4* (A)	0.85* (E)

Ejemplo Propuesto

- **Veamos este ejemplo:**