Estructuras para Gráficas

UNIDAD 5

ALGORITMOS Y ESTRUCTURAS DE DATOS

Definiciones

Un grafo *G = (V, E)* está formado por un conjunto de vértices V y un conjunto de aristas E

En ocasiones las aristas se denominan arcos y los vértices nodos

Los grafos dirigidos se denominan digrafos

Un grafo con costos en sus aristas se denomina grafo pesado

Definiciones

Algunas situaciones que pueden representarse:

- Mapas de caminos
- Diagramas de circuitos
- Caminos más cortos
- Horarios de clase
- Planeación
- Rutas críticas
- Orden causal

Representación

La forma más común es por medio de un diagrama

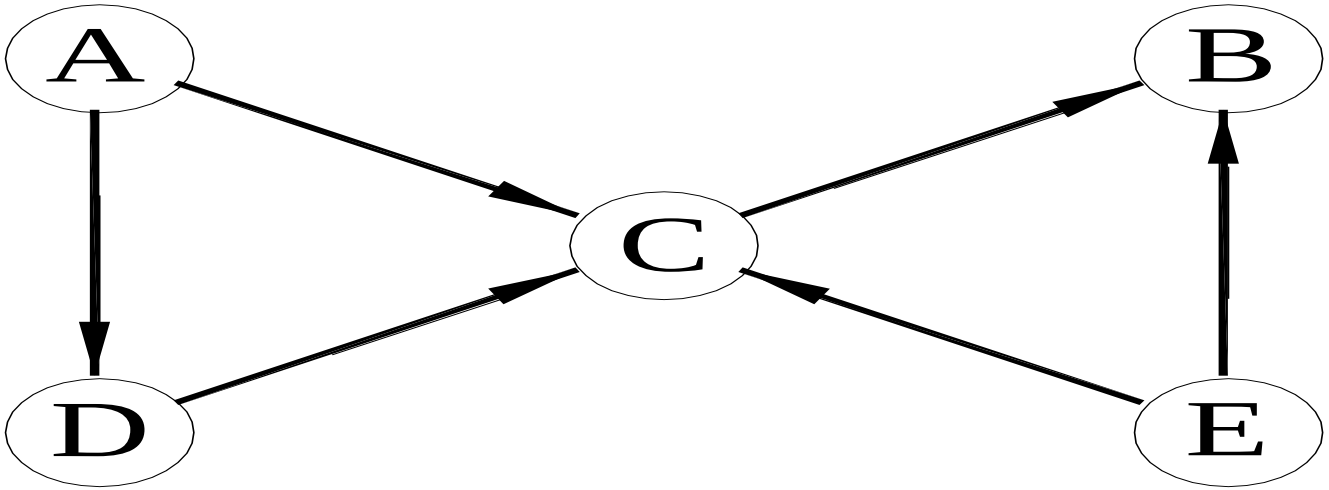
Los vértices se representan con círculos y las aristas con líneas dirigidas o no

Ejemplo

V = {A, B, C, D, E}

E = {e_1, e_2, e_3, e_4, e_5, e_6}

G = {(A,D), (A,C), (D,C), (E,C), (E,B), (C,B)}



Propiedades

Vértices adyacentes. Dos vértices son adyacentes si se encuentran unidos por una arista

Grado de un vértice. El grado de un vértice es la cantidad de aristas que llegan o salen de él

Semiciclo. Es una arista que sale y llega al mismo vértice

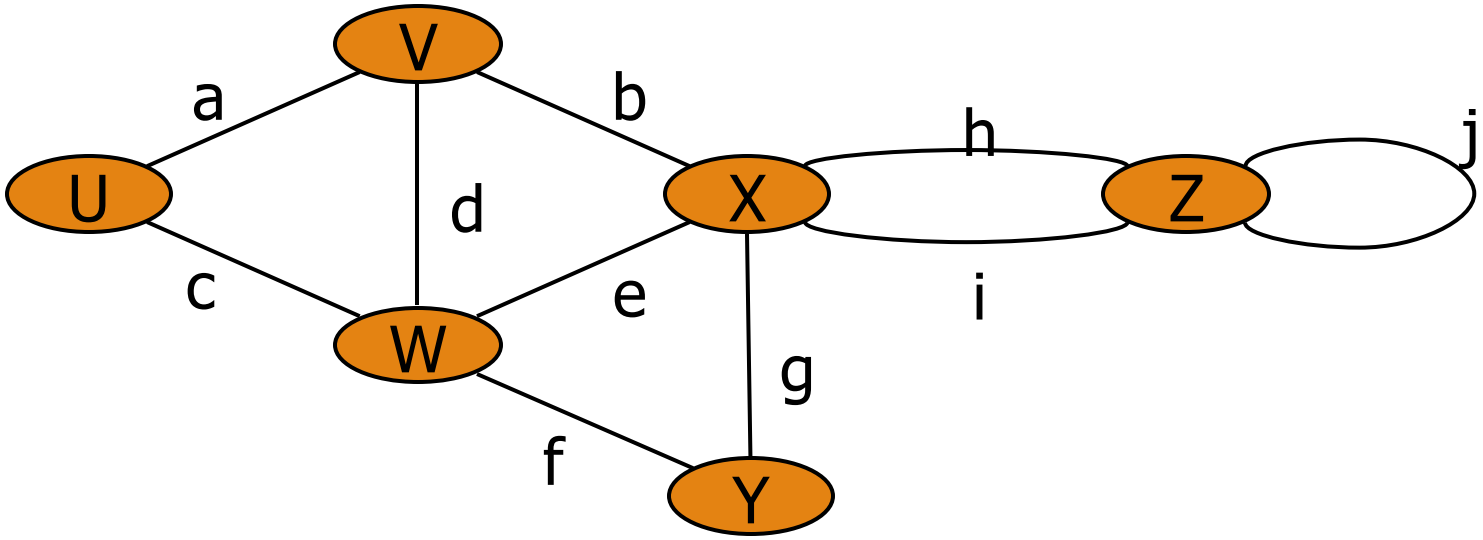
Arcos paralelos. Son dos o más aristas que unen a diferentes vértices

Ejemplo

Grado de los vértices: {u,2} {v,3} {w,3} {x,5} {y,2} {z,3}

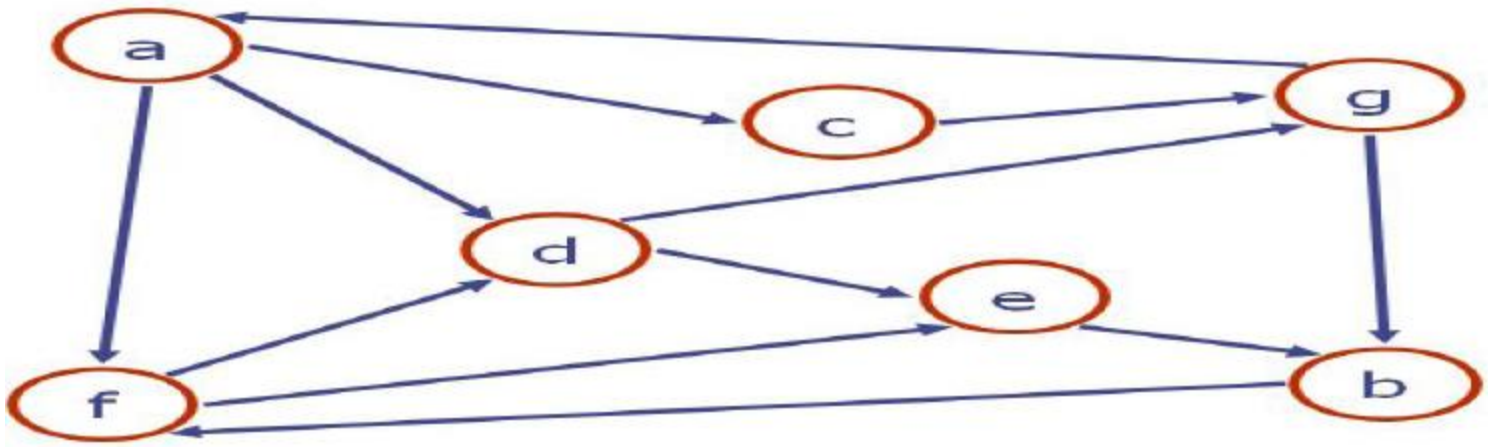
Aristas que son semi ciclos : j

Aristas paralelas : h, i



Grafo fuertemente conectado

Un grafo esta fuertemente conectado si para cualesquiera dos vértices *u* y *v*, *u* alcanza a *v* y *v* alcanza a *u*



Representación con estructuras

Existen diferentes estructuras de datos que pueden utilizarse para representar un grafo no importando si es pesado o dirigido o no son:

- Matriz de adyacencia
- Lista de adyacencia

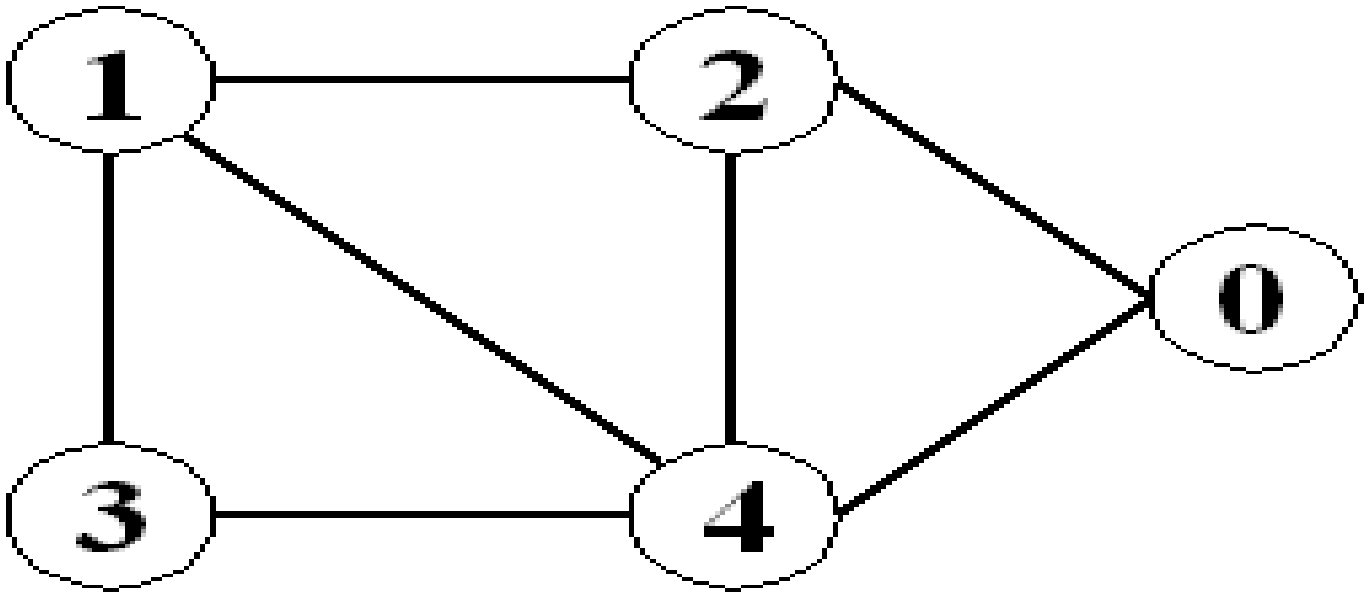
Matriz de adyacencia

Sea un grafo *G = (V, E),* donde:

$$V = \{0, 1, 2, 3, .., n\}$$

Se representa con una matriz de adyacencia M, tal que M es una matriz booleana de *nxn*, en dode *M[i][j]* es verdadero si existe una arista entre i y j, en caso contrario *M[i][j]* es falso

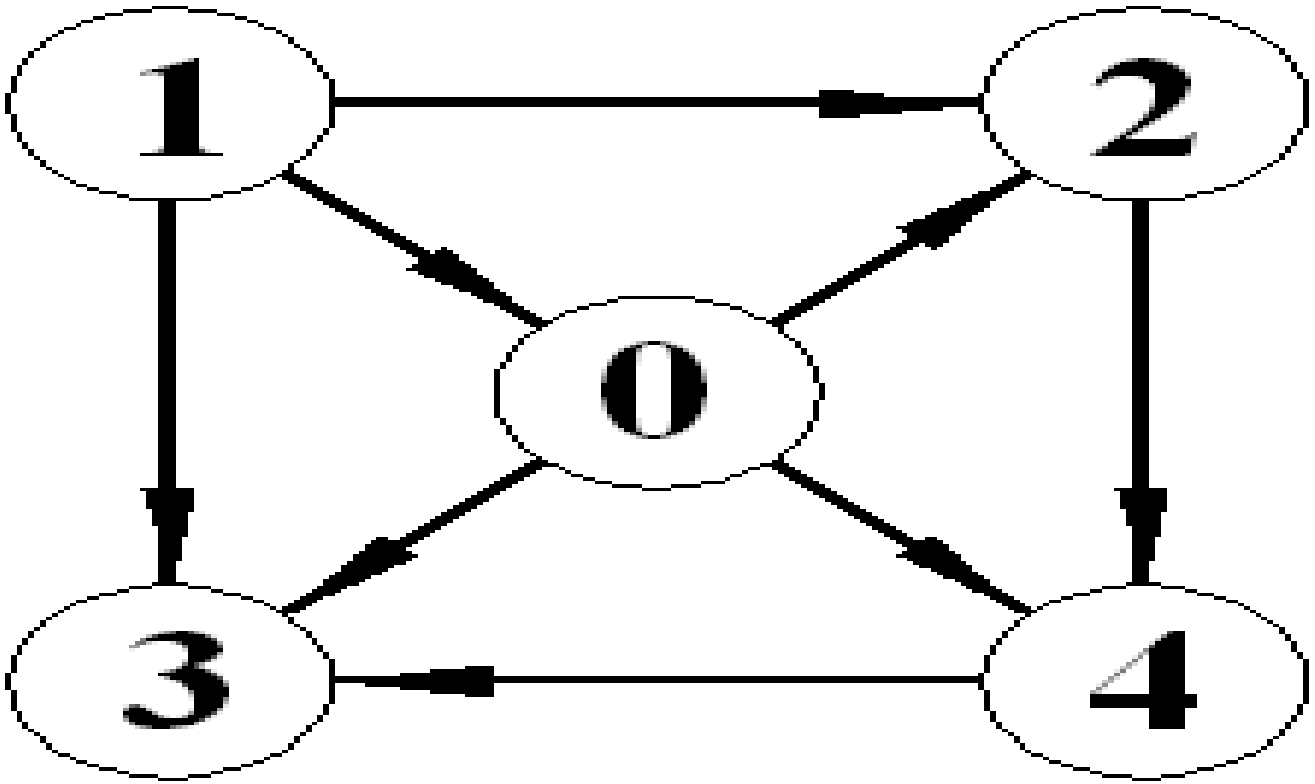
Ejemplo



Ejemplo

	0	1	2	3	4
0	Verdadero	Falso	Verdadero	Falso	Verdadero
1	Falso	Verdadero	Verdadero	Verdadero	Verdadero
2	Verdadero	Verdadero	Verdadero	Falso	Verdadero
3	Falso	Verdadero	Falso	Verdadero	Verdadero
4	Verdadero	Verdadero	Verdadero	Verdadero	Verdadero

Matriz de adyacencia



Matriz de adyacencia

	0	1	2	3	4
0	Verdadero	Falso	Verdadero	Verdadero	Verdadero
1	Verdadero	Verdadero	Verdadero	Verdadero	Falso
2	Falso	Falso	Verdadero	Falso	Verdadero
3	Falso	Falso	Falso	Verdadero	Falso
4	Falso	Falso	Falso	Verdadero	Verdadero

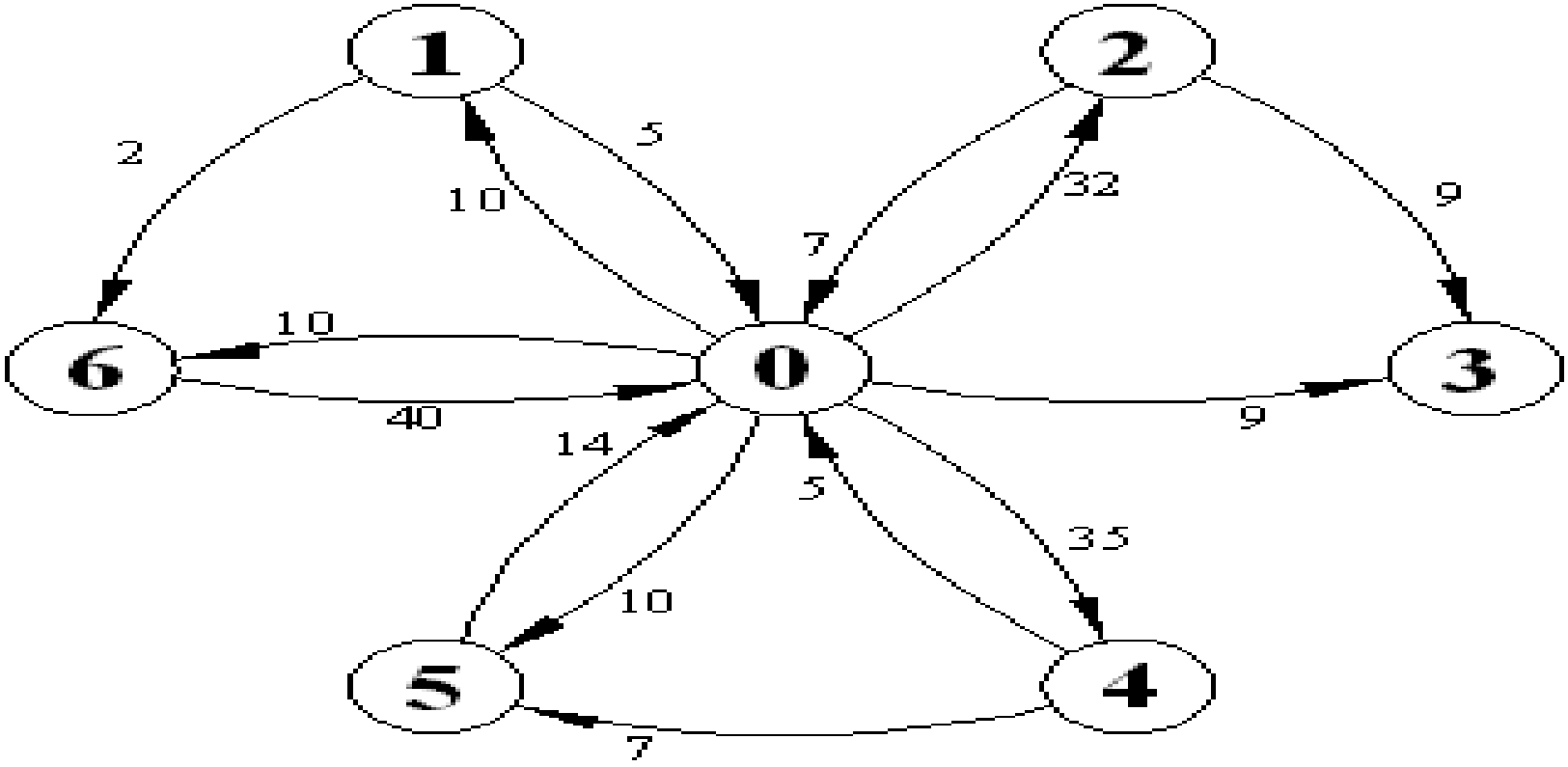
Matrices etiquetadas de adyacencia

Se utiliza para representar grafos pesados

La representación es similar pero el elemento *[i][j]* tiene el costo de la arista que va de *i* a *j*

Si no existe un arco entre *i* y *j* se le asigna un valor especial a la entrada (generalmente infinito)

Ejemplo



Ejemplo

	0	1	2	3	4	5	6
0	0	10	32	9	35	10	10
1	5	0	∞	∞	∞	∞	2
2	7	∞	0	9	∞	∞	∞
3	∞	∞	∞	0	∞	∞	∞
4	5	∞	∞	∞	0	7	∞
5	14	∞	∞	∞	∞	0	∞
6	40	∞	∞	∞	∞	∞	0

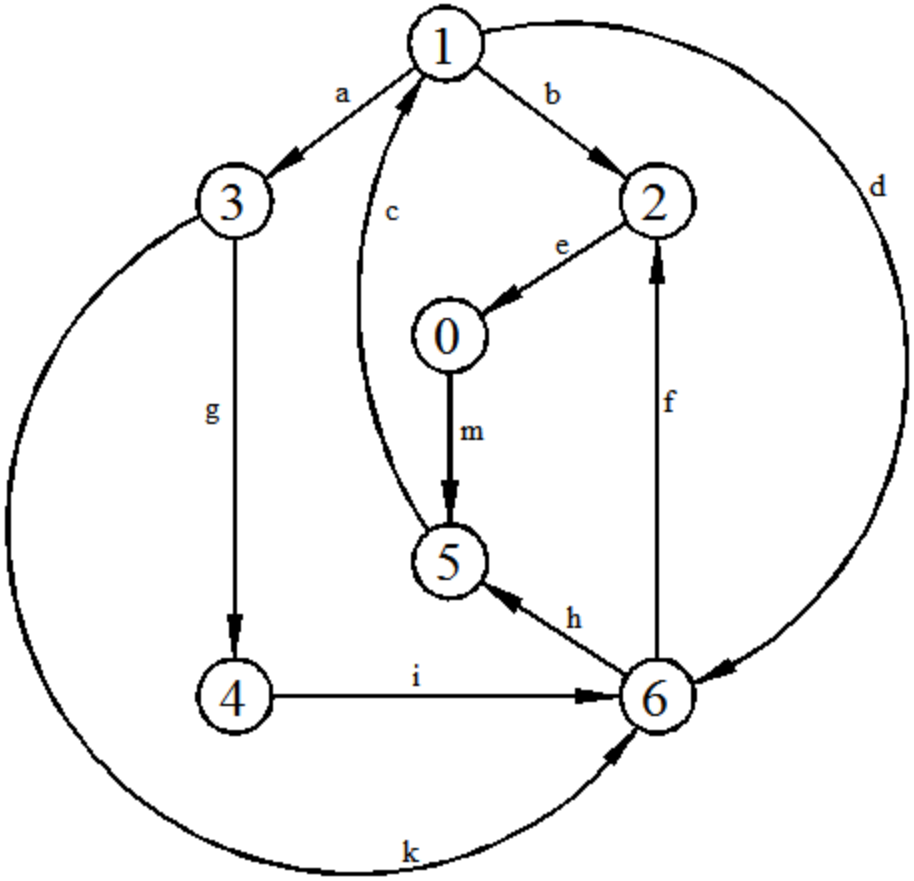
Listas de adyacencia

Cada vértice es una lista que contiene en algún orden dado todos los vértices adyacentes a él

Para representar un grafo se puede utilizar una lista o un arreglo donde cada elemento de *i* es una lista de adyacencia para el vértice *i*

Para representar un grafo con pesos se añade a cada nodo un nuevo dato con el peso de su arco

Ejemplo



Lista de adyacencia

$$
\begin{array}{lllllll}
0 & \rightarrow & 5_m & & & & \\
1 & \rightarrow & 2_b & \rightarrow & 3_a & \rightarrow & 6_d \\
2 & \rightarrow & 0_e & & & & \\
3 & \rightarrow & 4_g & \rightarrow & 6_k & & \\
4 & \rightarrow & 6_i & & & & \\
5 & \rightarrow & 1_c & & & & \\
6 & \rightarrow & 2_f & \rightarrow & 5_h & &
\end{array}
$$

RECORRIDOS

Recorridos

Los algoritmos más conocidos para recorrer los nodos de un grafo son:

- Búsqueda primero en profundidad
- Búsqueda primero en amplitud

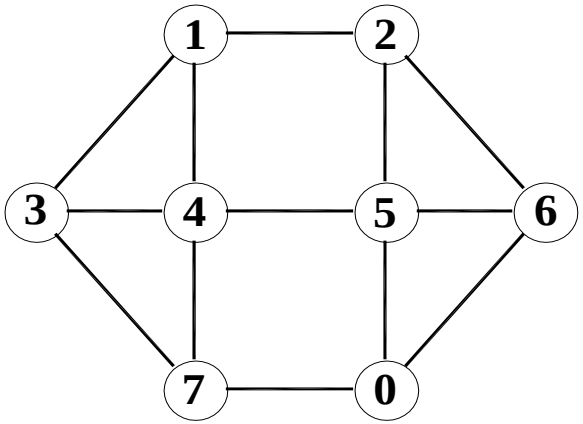
Búsqueda primero en profundidad

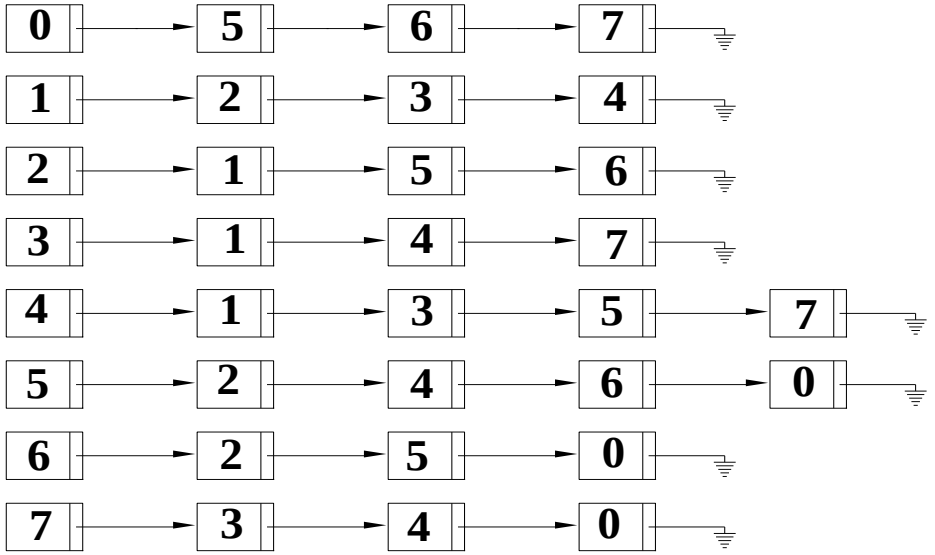
La idea es visitar nuevos vértices

El recorrido inicia en cualquier vértice llamado la raíz de la búsqueda que se marca como visitado

Un vértice *w* adyacente a *v*, que no se encuentre marcado como visitado se empieza a recorrer recursivamente

Ejemplo





Si el algoritmo inicia con el vértice 1, el orden en que se visitan los nodos es:

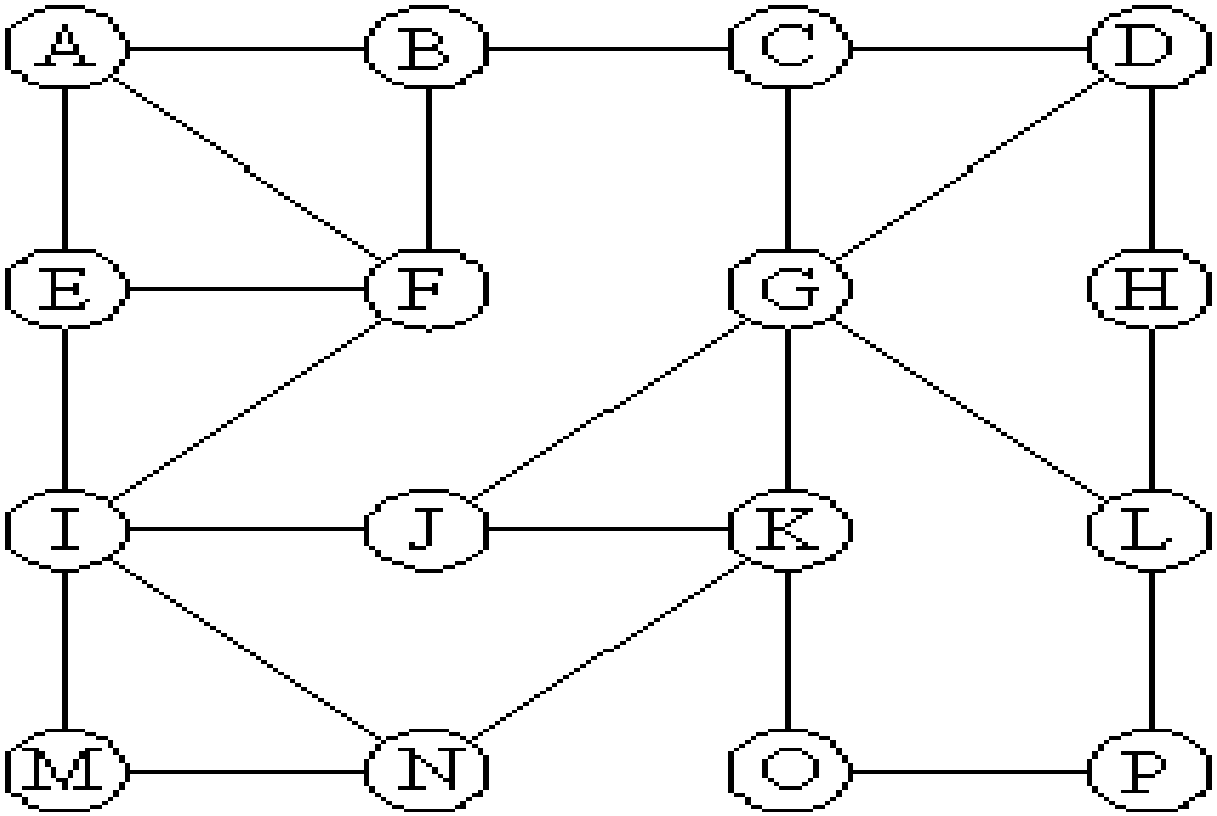
1, 2, 5, 4, 3, 7, 0, 6

Búsqueda en amplitud

Dado un nodo, se visita a todos sus vértices adyacentes y posteriormente a los adyacentes de estos adyacentes

Se puede ver como un recorrido en niveles, en el nivel 0 encontramos al vértice inicial, en el 1 a sus adyacentes y así sucesivamente

Ejemplo



Ejemplo, cont...

Nivel 0: A

Nivel 1: B, F, E

Nivel 2: C, I

Nivel 3: D, G, J, M, N

Nivel 4: H, L, K

Nivel 5: O, P

El recorrido podría quedar: A, B, F, E, C, I, D, G, J, M, N, H, L, K, O, P

Árboles Abarcadores Mínimos (AAM)

Definiciones

Un Árbol Abarcador de un grafo *G* es un sub-grafo de G que es un árbol y contiene a todos los vértices de *G*

Un Árbol Abarcador Mínimo es un árbol abarcador con peso mínimo

Definiciones

Un árbol abarcador mínimo debe cumplir con las siguientes características:

- Si el árbol no es conexo (que todos los vértices estén conectados), no existe un árbol abarcador
- Un árbol abarcador mínimo en un grafo *G* es un sub-grafo *T* de *G* que cumple con las siguientes propiedades:
  - T es conexo
  - T es acíclico

Árbol Abarcador Mínimo

Para encontrar un árbol abarcador mínimo dado un grafo *G* se tienen los siguientes algoritmos:

- Algoritmo de Prim/Dijkstra
- Algoritmo de Kruskal

Si es posible obtener un AAM de un grafo, el resultado debe ser el mismo sin importar el algoritmo utilizado

Algoritmo de Prim

Selecciona un vértice arbitrario y se ramifica hacia fuera desde la parte construida seleccionando un nuevo vértice

Los vértices pueden dividirse en tres categorías:

- Vértices en el árbol construido
- Vértices en el límite o bordo, no están en el árbol pero son adyacentes a algún vértice en el árbol
- Vértices no visitados

Algoritmo de Prim

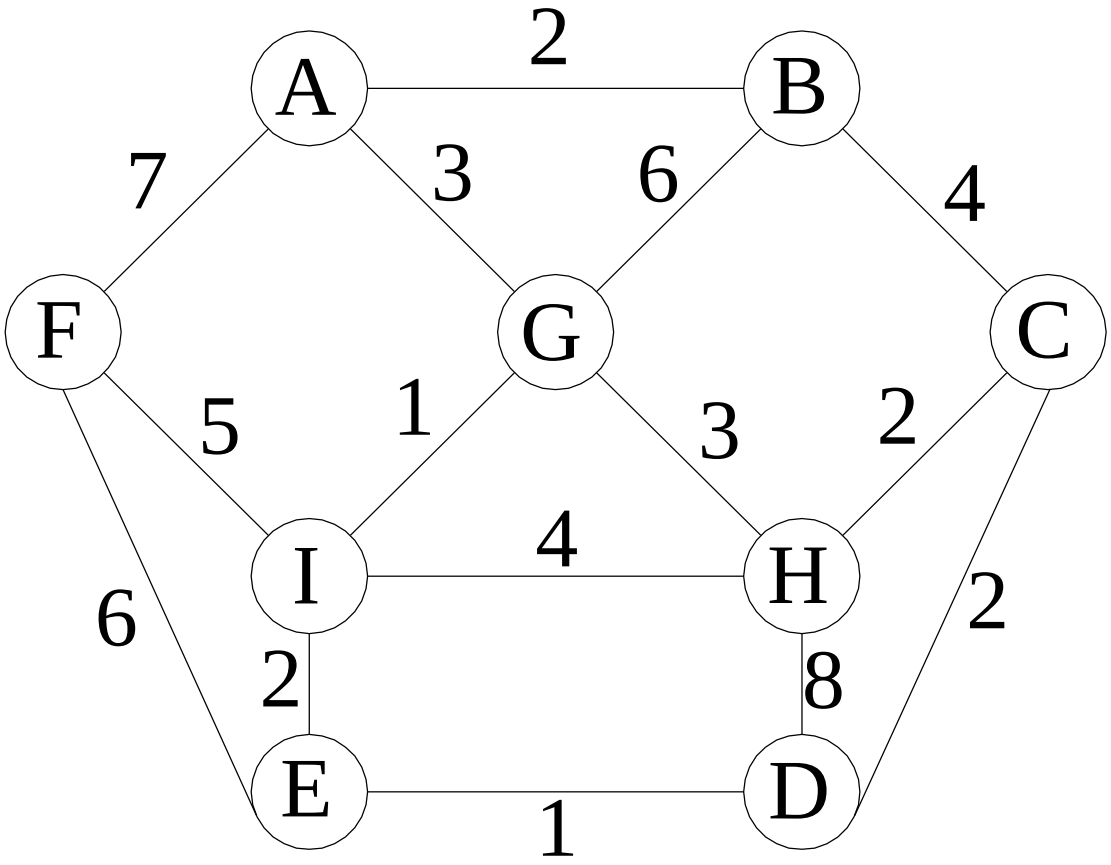
La idea es seleccionar un vértice límite y un arco incidente

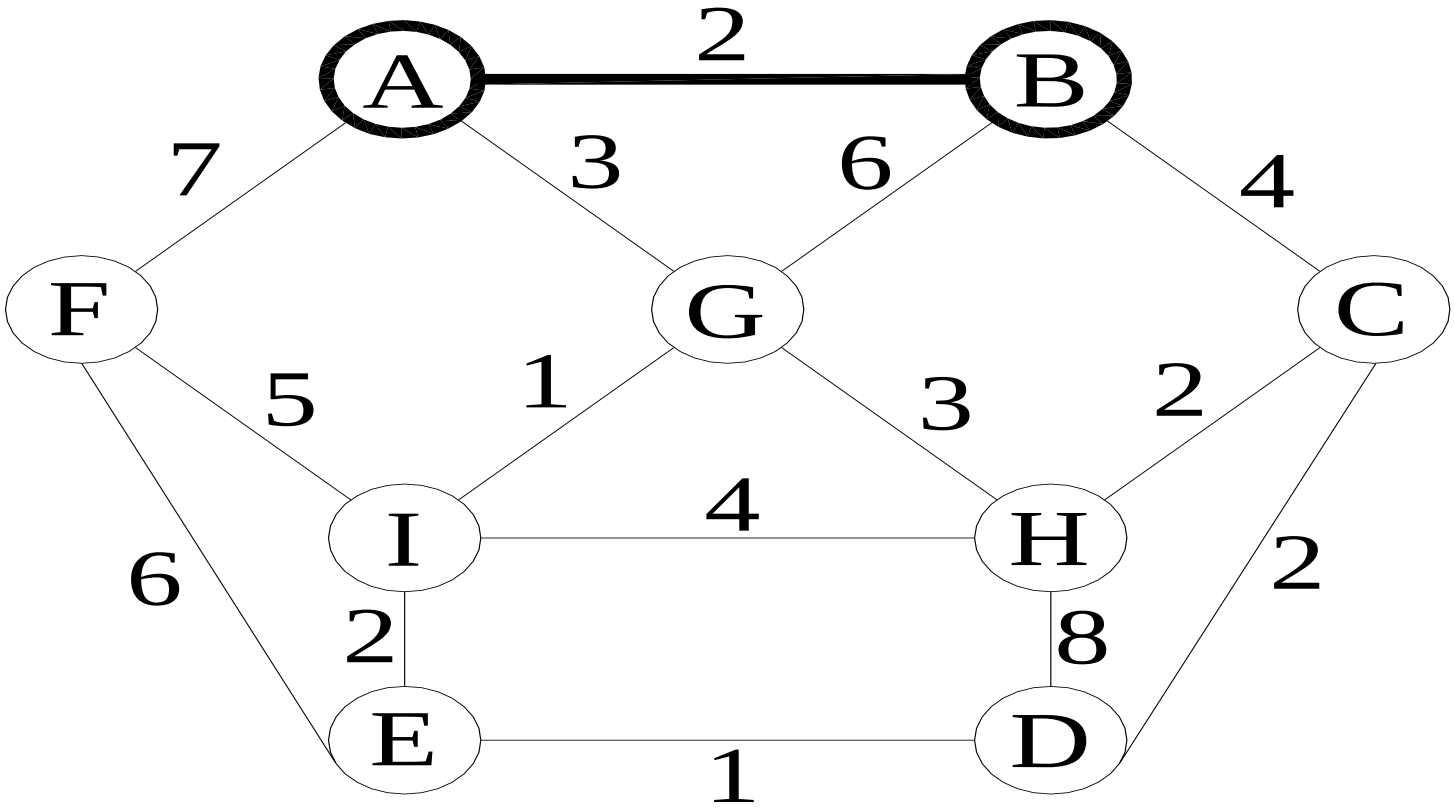
El algoritmo selecciona un arco de un vértice del árbol o un vértice límite con peso mínimo

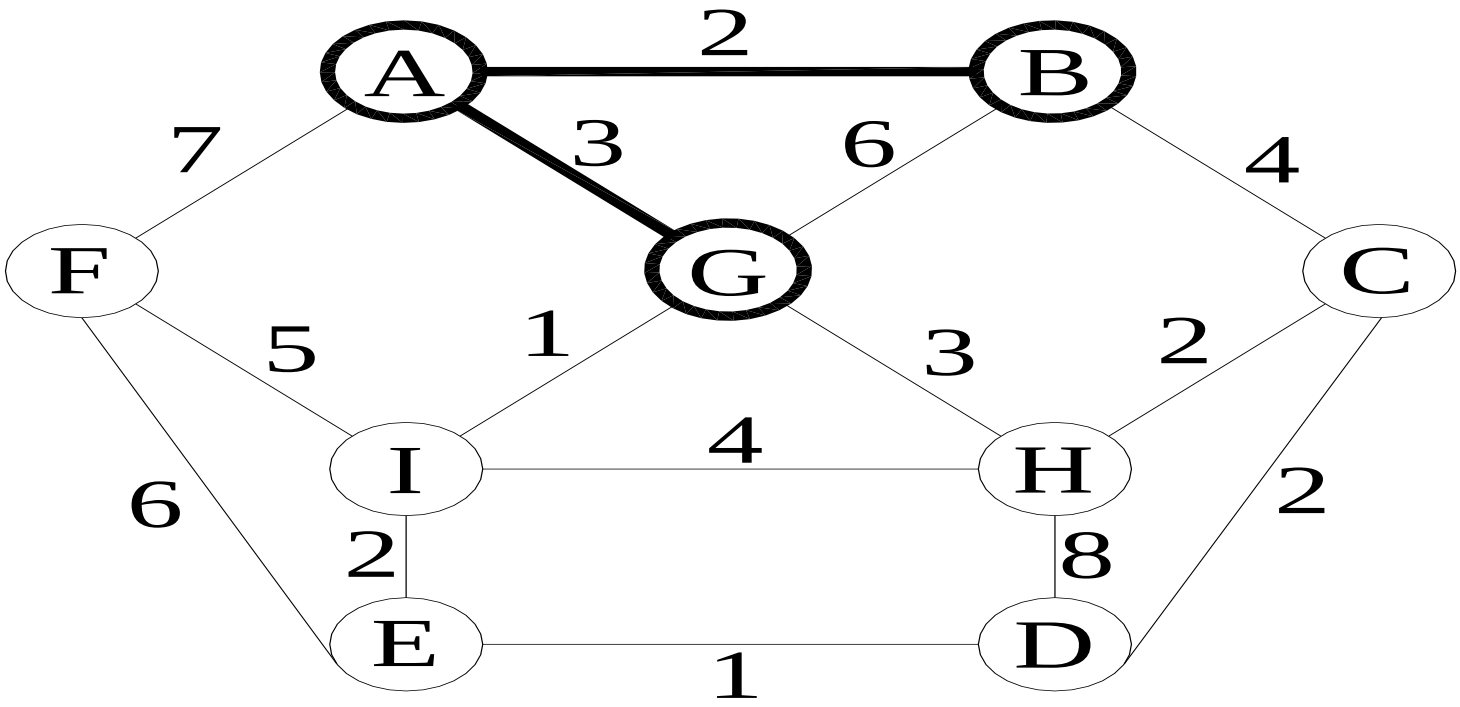
Después de cada iteración, puede haber nuevos vértices límites

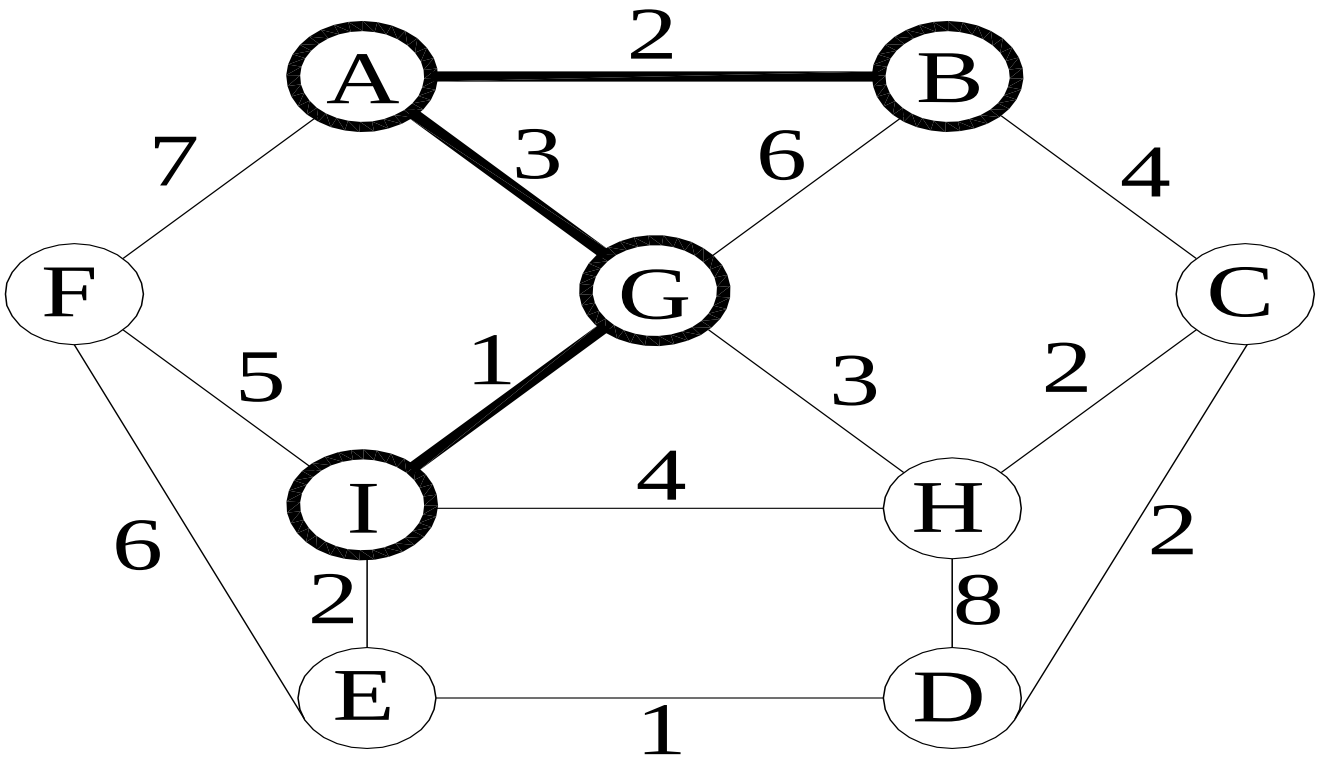
No importa el vértice de inicio, el resultado debe ser el mismo

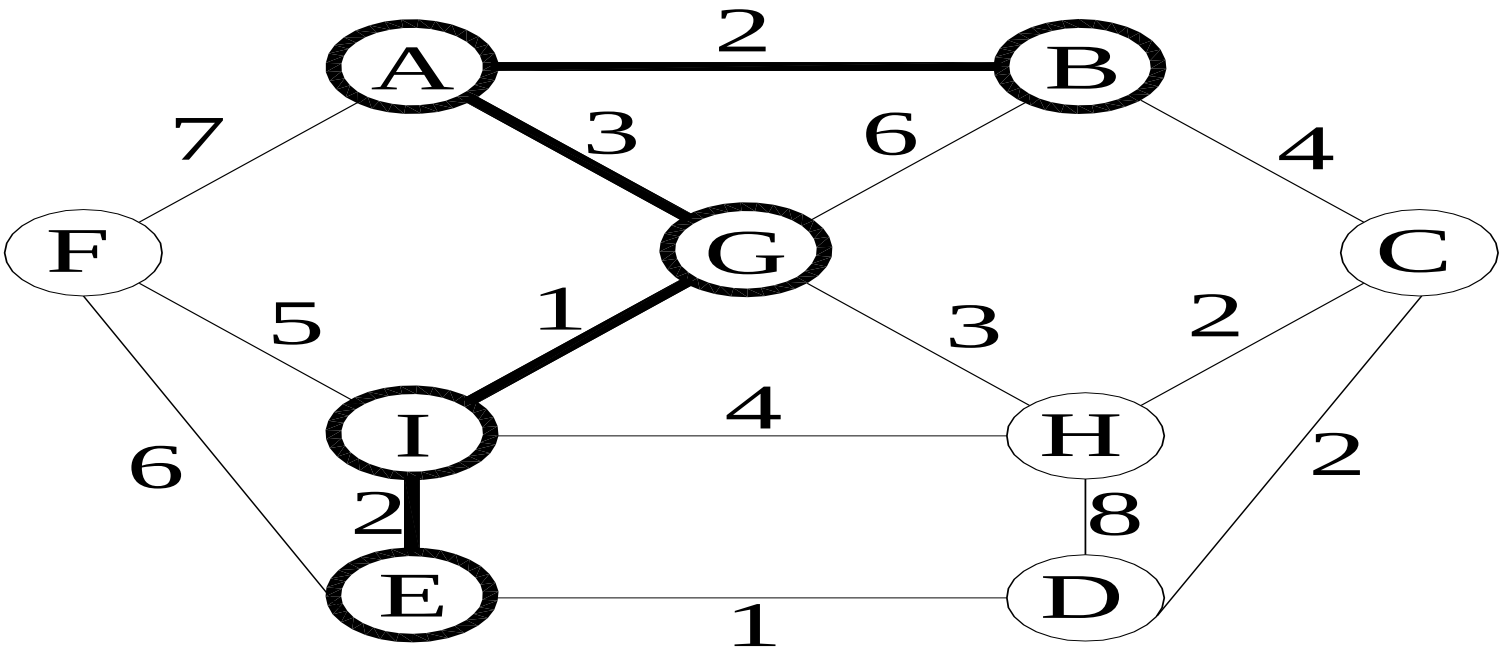
Ejemplo

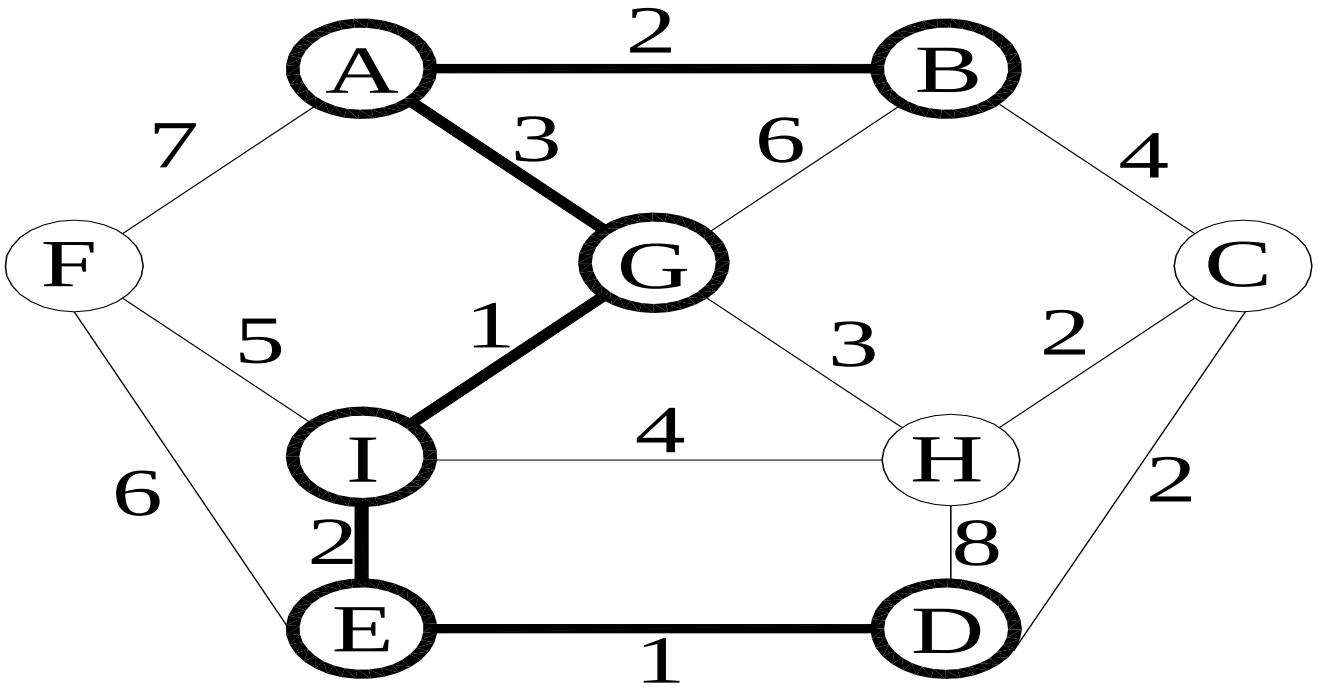


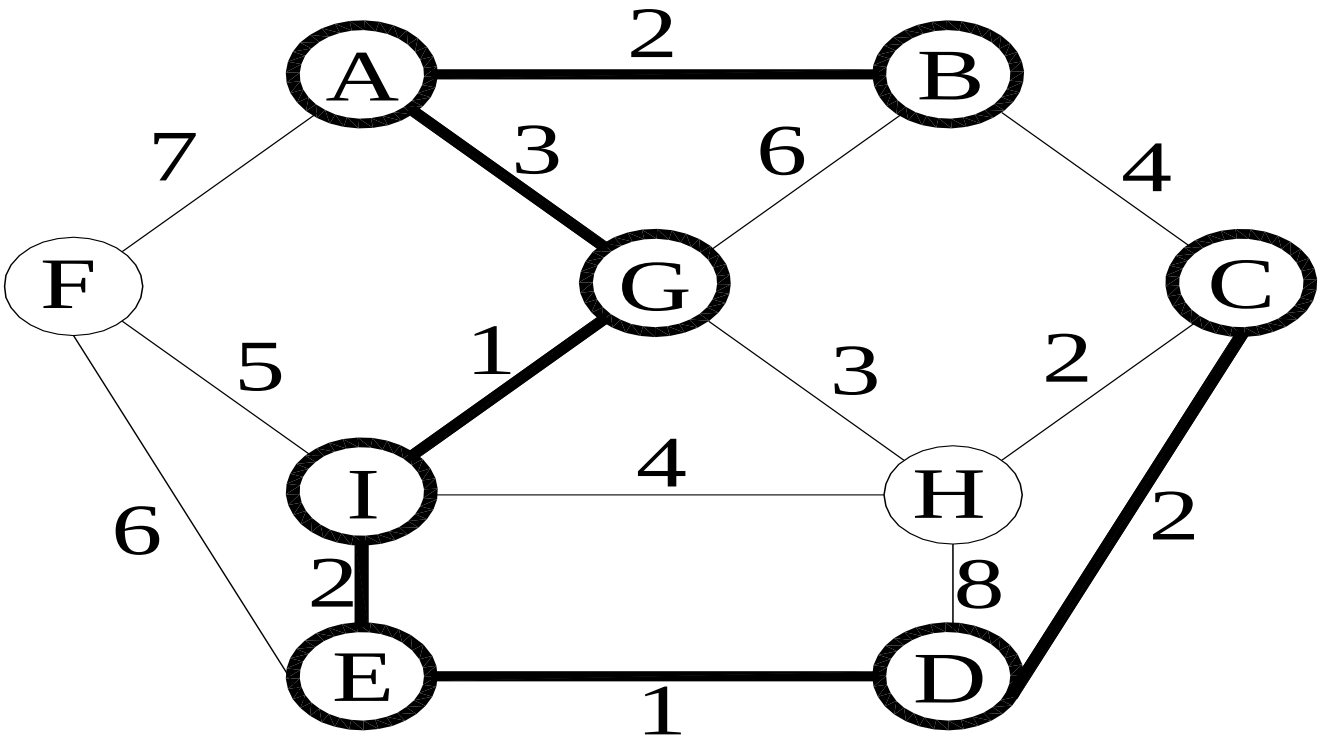

A
B
C
D
E
F
G
H
I
2
7
3
6
4
5
1
3
2
4
6
2
8
1

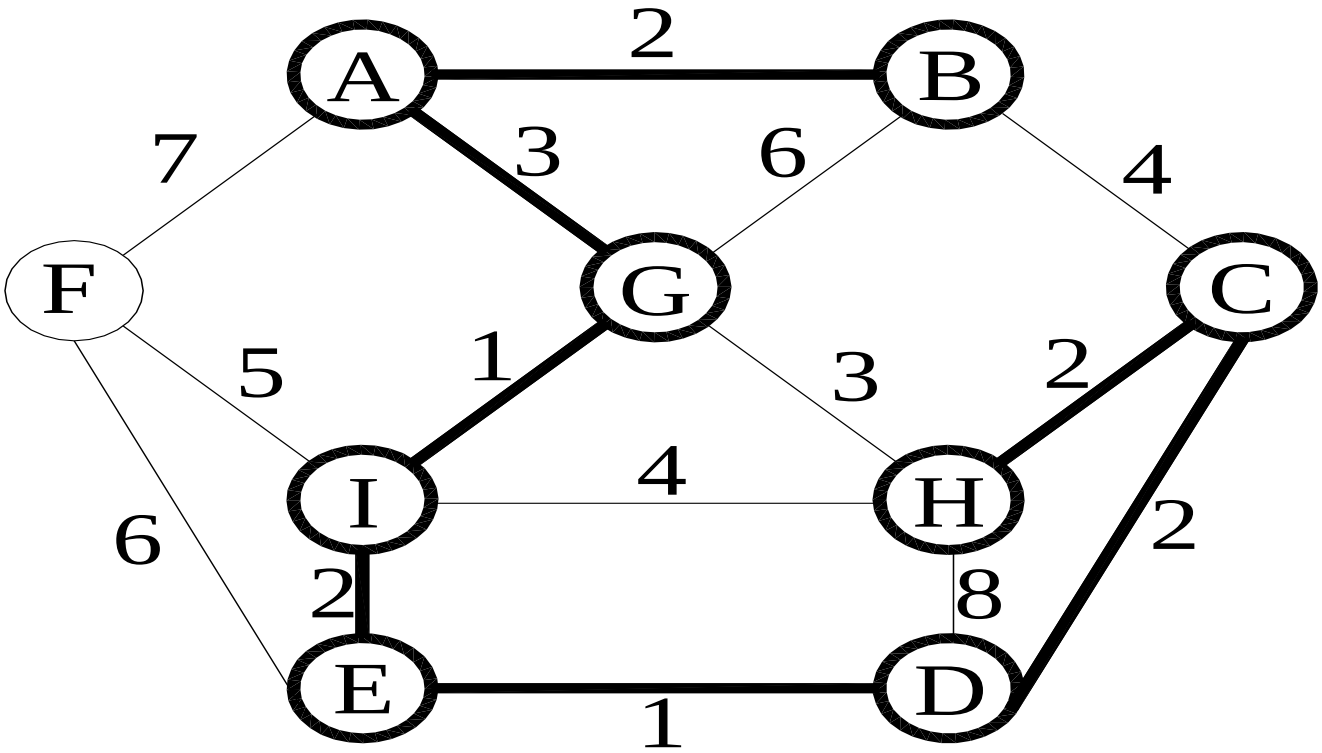

A
B
C
D
E
F
G
H
I
2
7
3
6
4
5
1
3
2
4
6
2
8
2
1

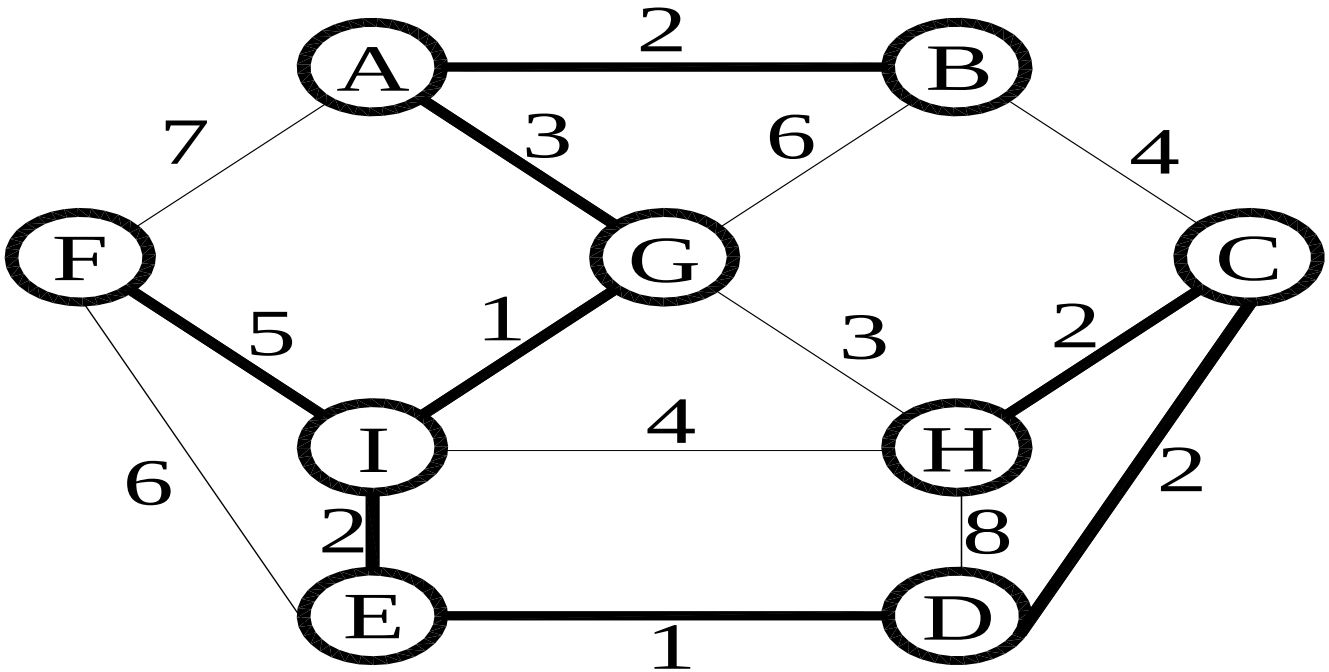

A
B
C
D
E
F
G
H
I
2
7
3
6
4
5
1
3
2
4
6
2
8
2
1

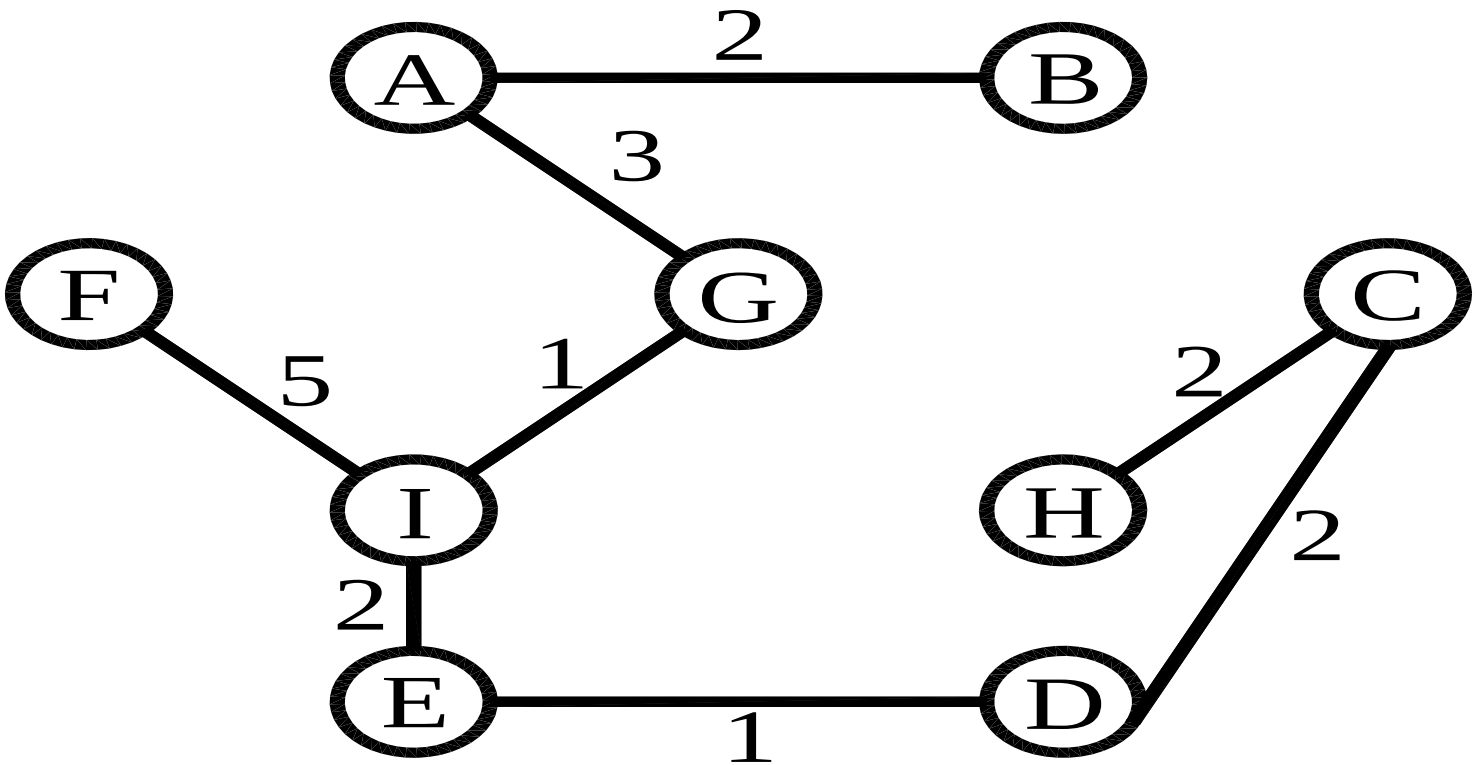

A
B
C
D
E
F
G
H
I
2
7
3
6
4
5
1
3
2
4
6
2
8
1


A
B
C
D
E
F
G
H
I
2
7
3
6
4
5
1
3
2
4
6
2
8
1


A
B
C
D
E
F
G
H
I
2
7
3
6
4
5
1
3
2
4
6
2
8
2
1


A
B
C
D
E
F
G
H
I
2
7
3
6
4
5
1
3
2
4
6
2
8
2
1


A
B
C
D
E
F
G
H
I
2
7
3
6
4
5
1
3
2
4
6
2
8
2
1


A
B
C
D
E
F
G
H
I
2
3
5
1
2
2
2
1

Finalmente se debe obtener el costo del árbol abarcado mínimo

Costo = (A, B) + (A, G)+ (G, I) +(I, F) + (I, E) + (E, D) + (C, D) + (H, C)

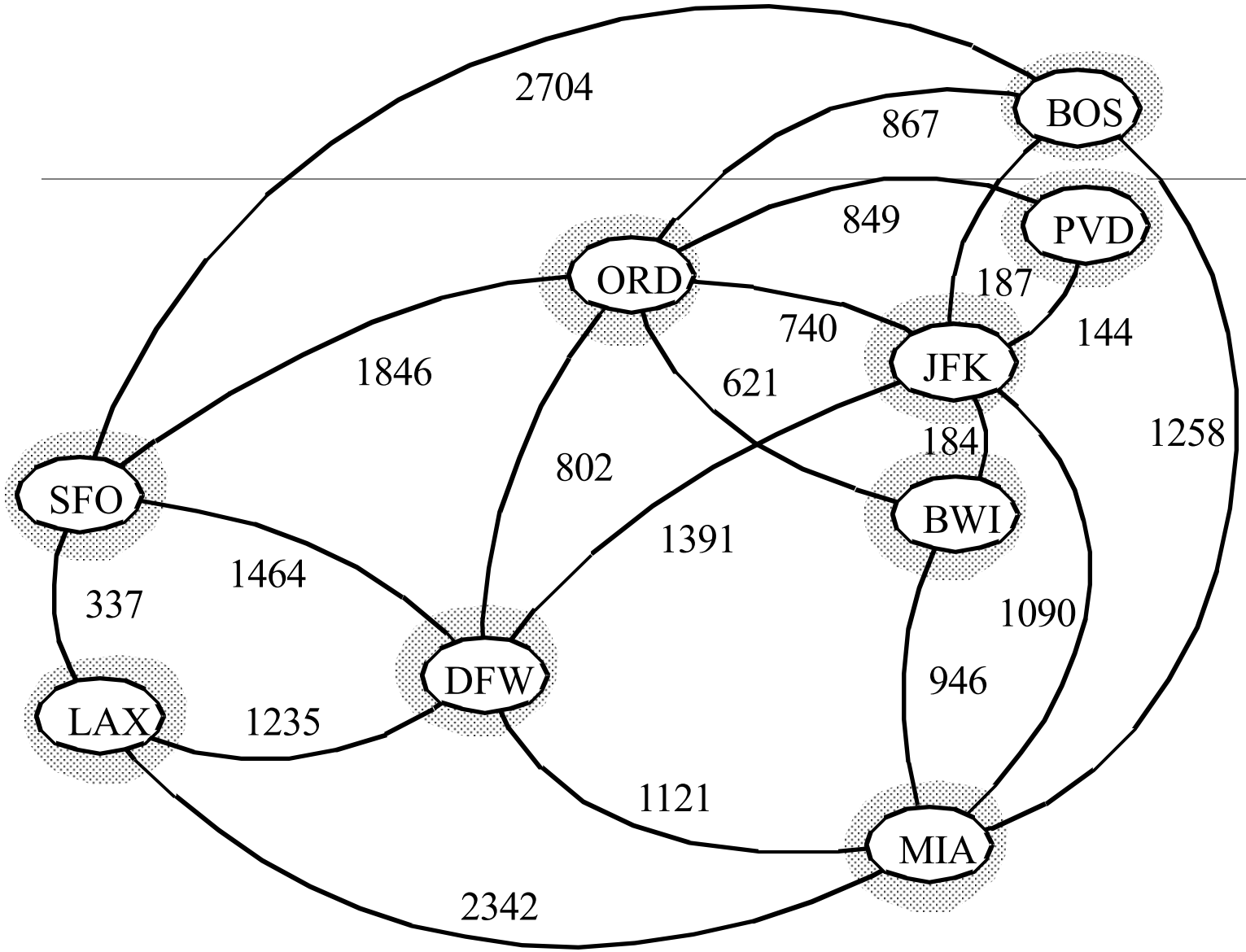
Costo = 2 +3 +1 +5 +2 +1 +2 +2 = 18

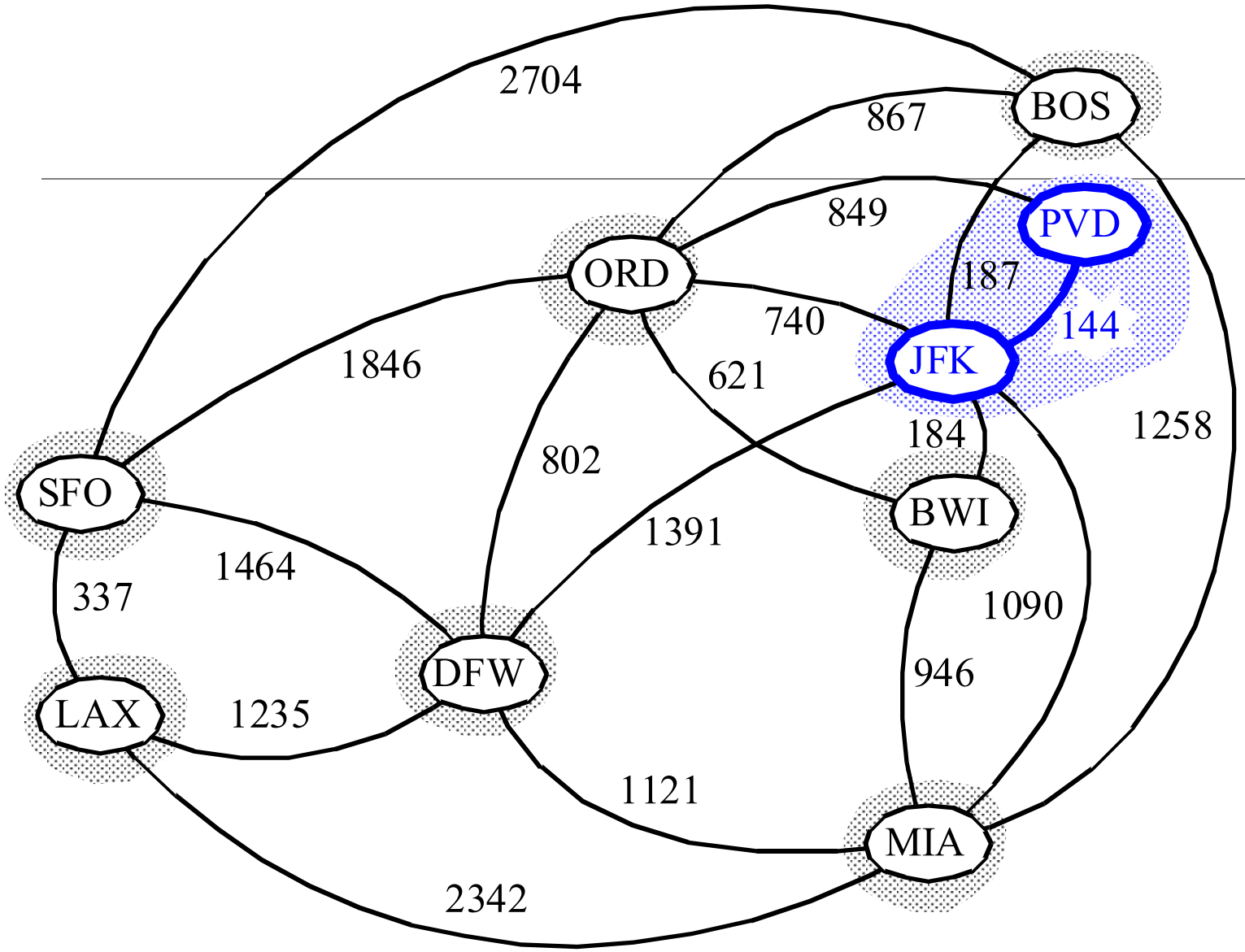
Algoritmo de Kruskal

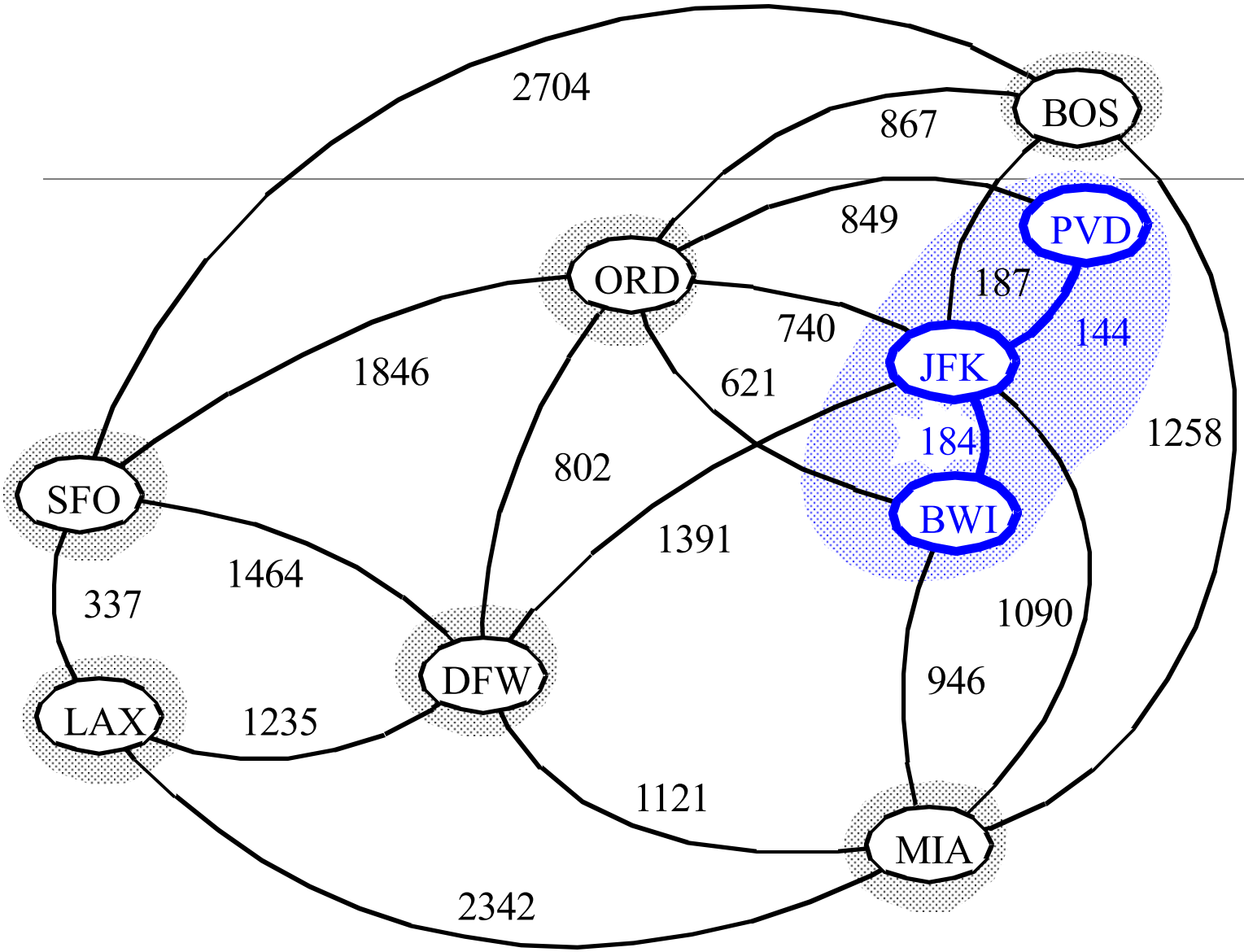
Se basa en añadir un arco a la vez

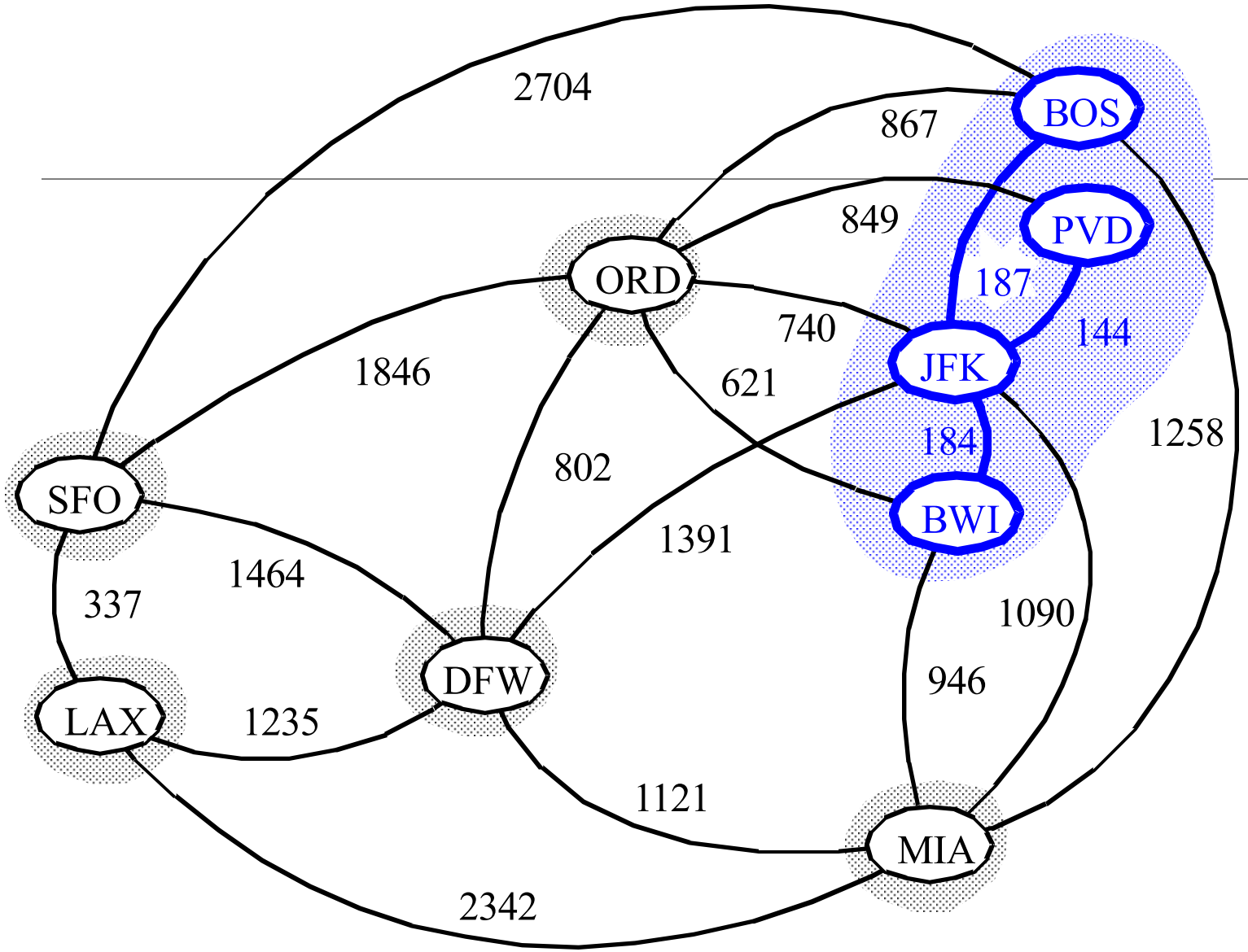
Se añade el arco que tenga asociado el menor costo

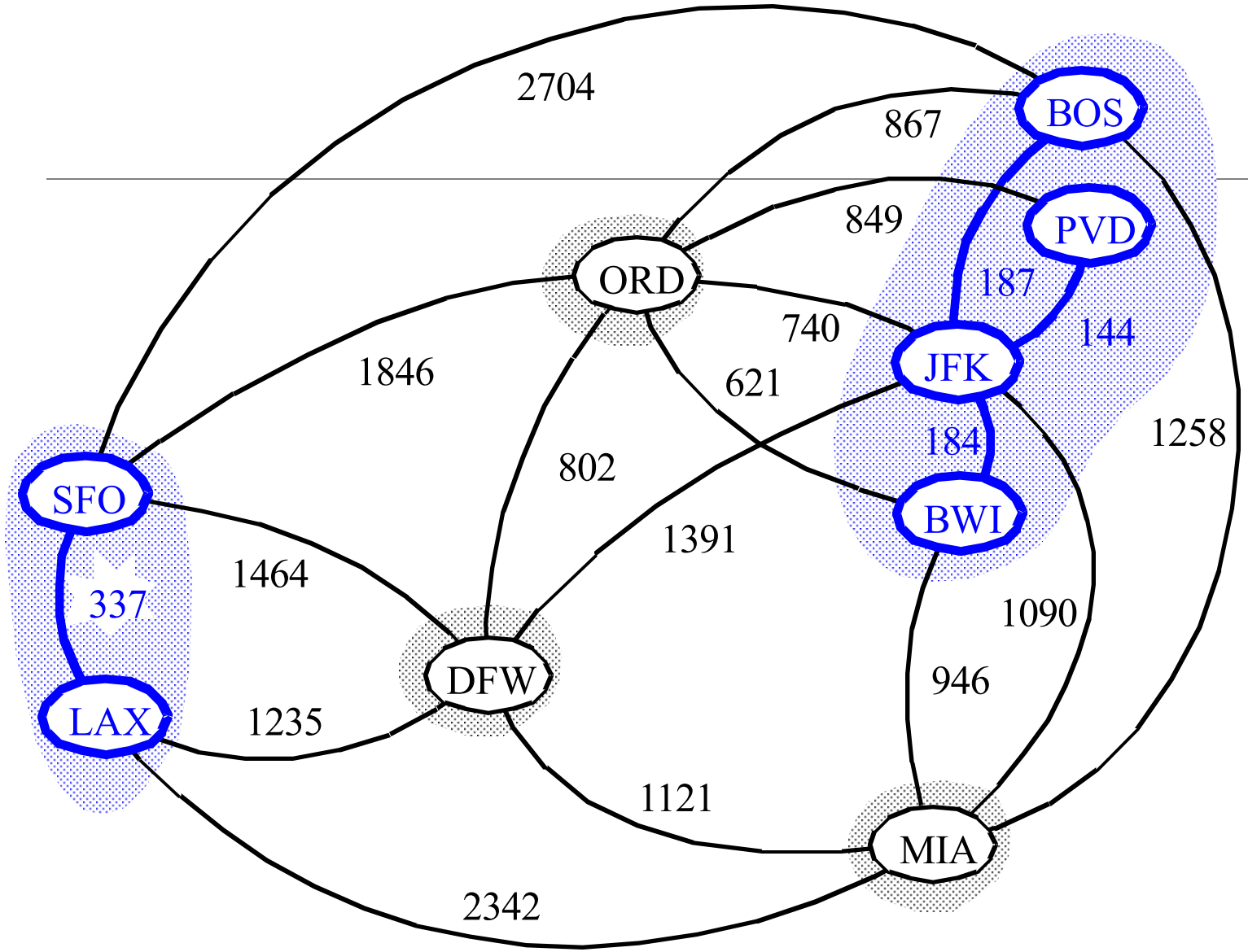
Los arcos seleccionados no deben formar ciclos

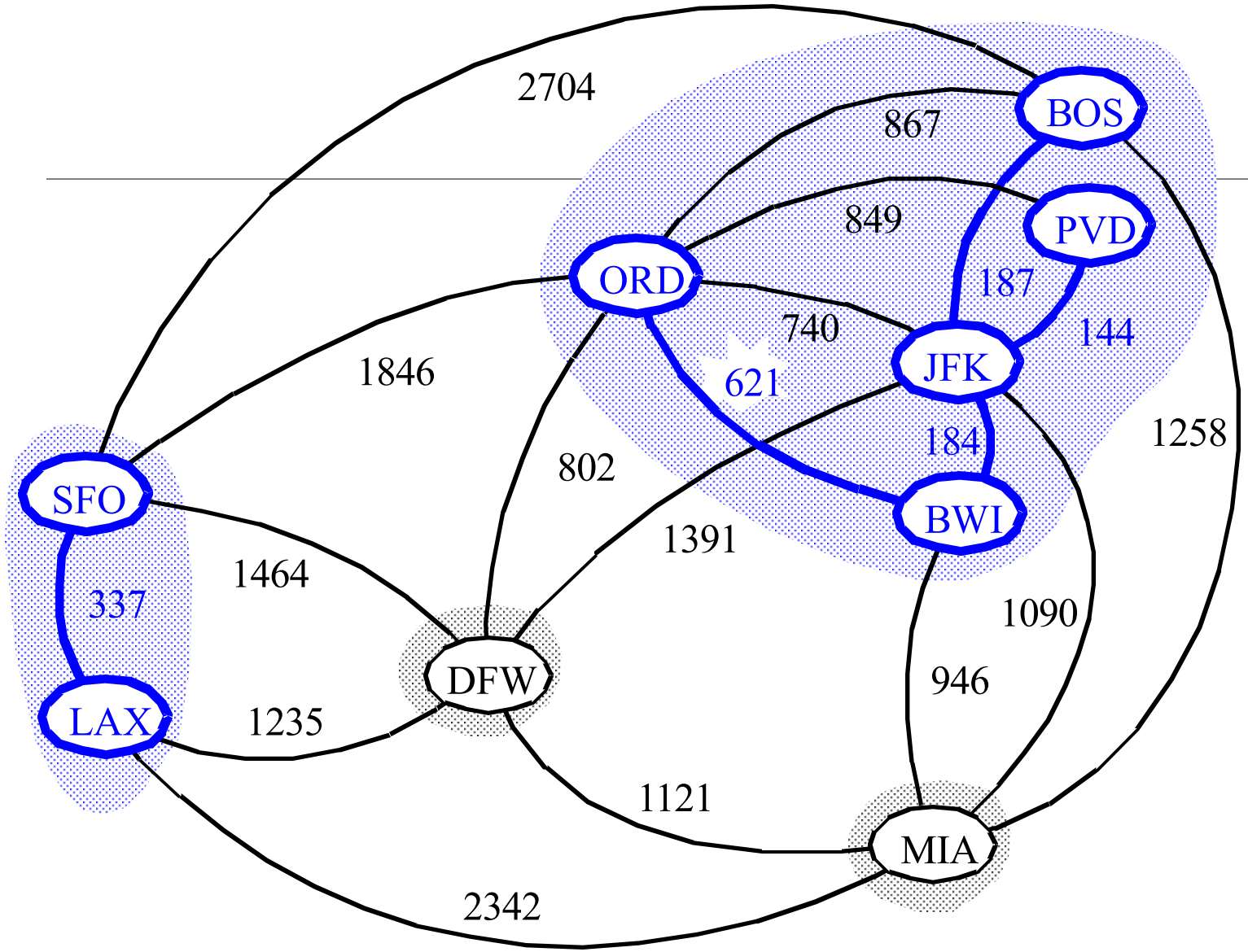

2704
867
BOS
849
PVD
ORD
187
740
144
JFK
1846
621
1258
184
802
SFO
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

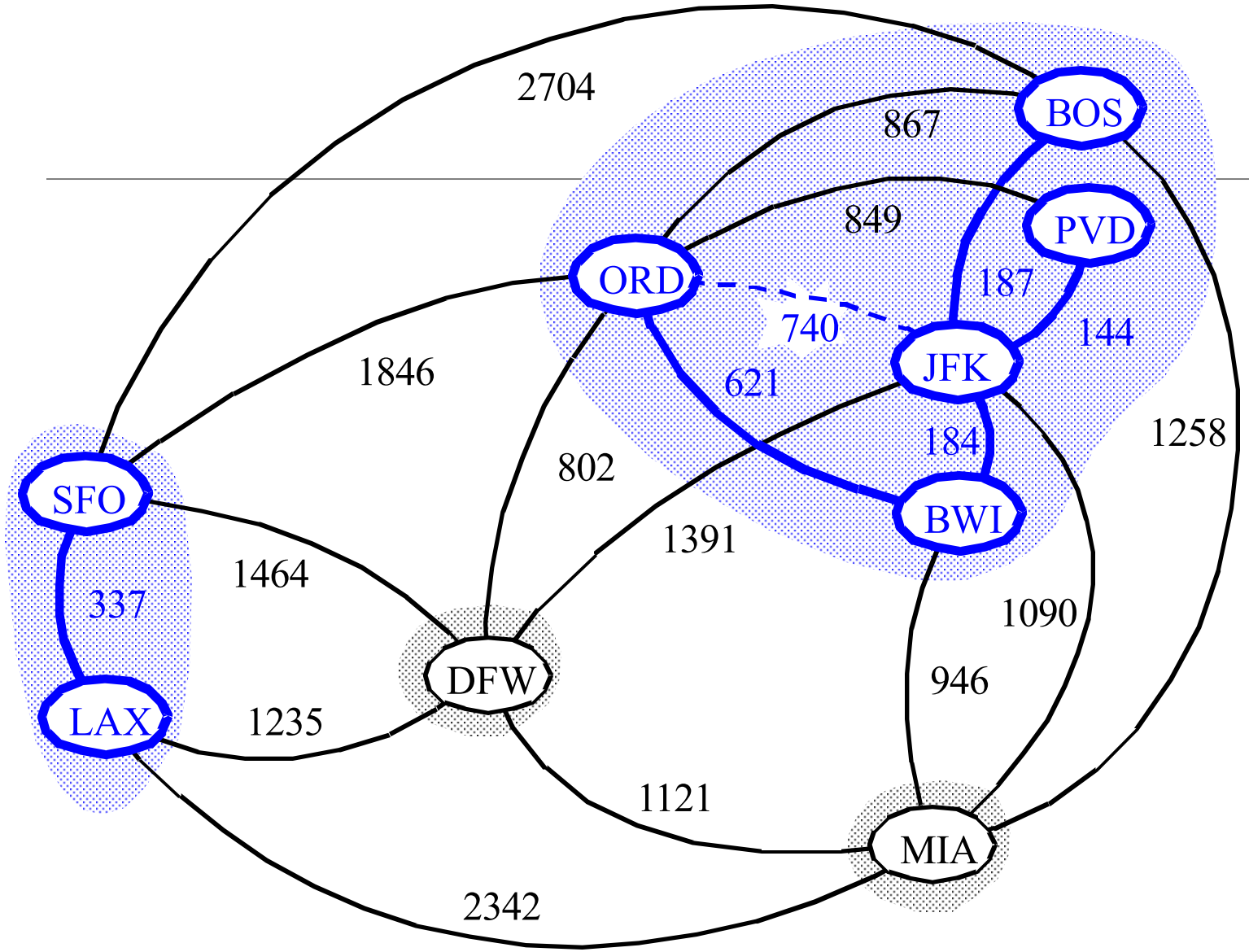

2704
867
BOS
849
PVD
ORD
187
740
144
JFK
1846
621
184
1258
802
SFO
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

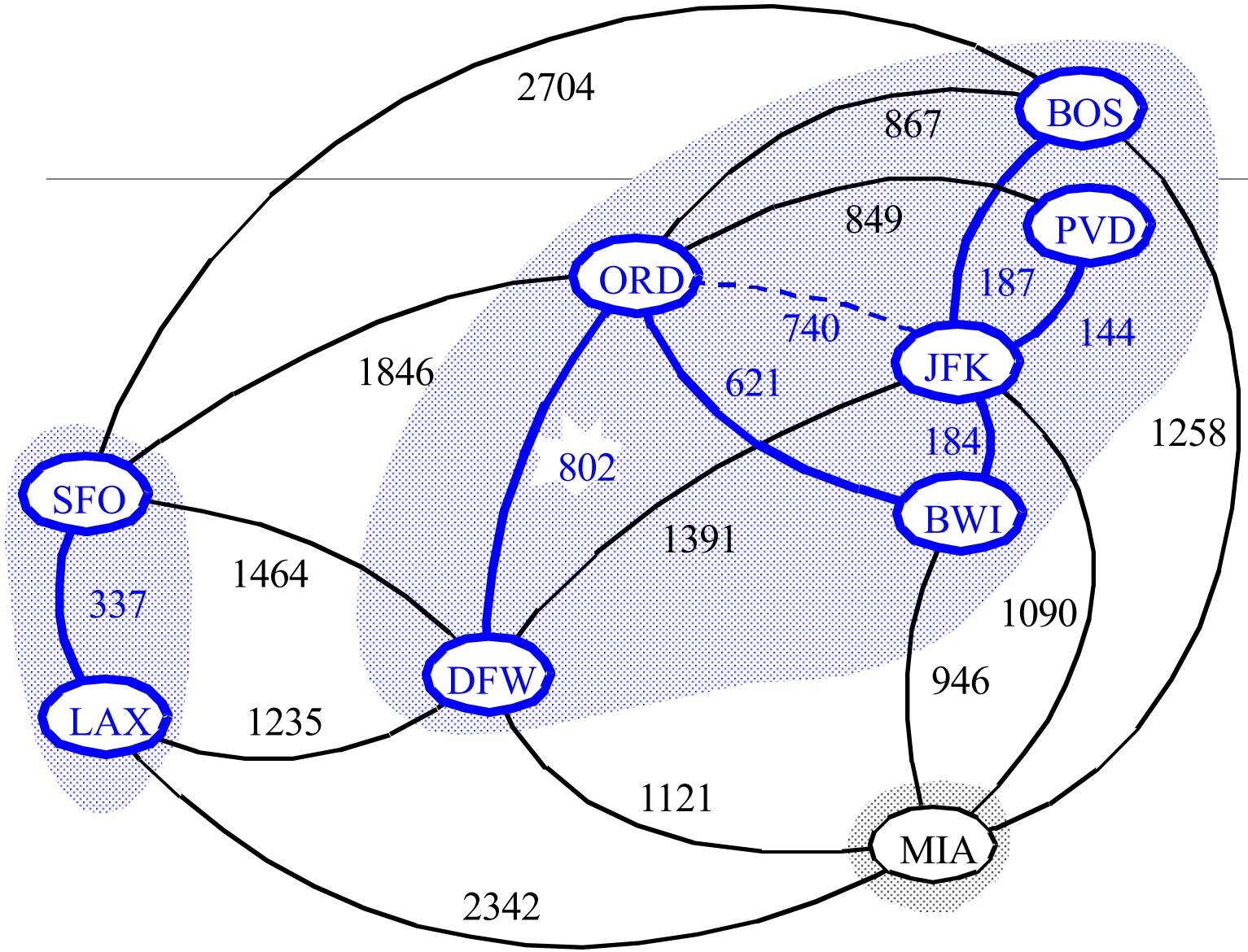

2704
867
BOS
849
PVD
ORD
187
740
144
JFK
1846
621
1258
184
802
SFO
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

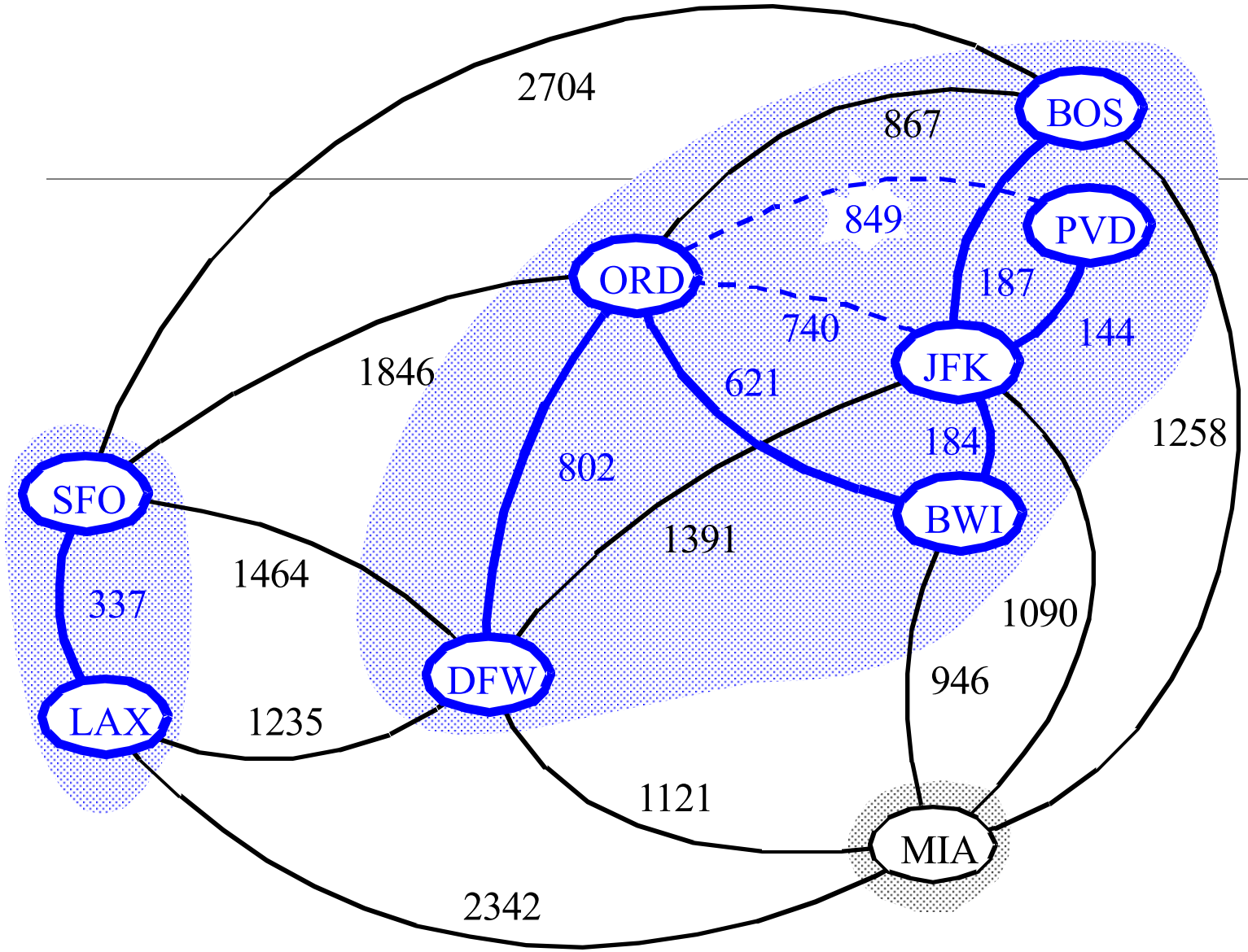

2704
867
BOS
849
PVD
ORD
187
740
144
JFK
1846
621
1258
184
802
SFO
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

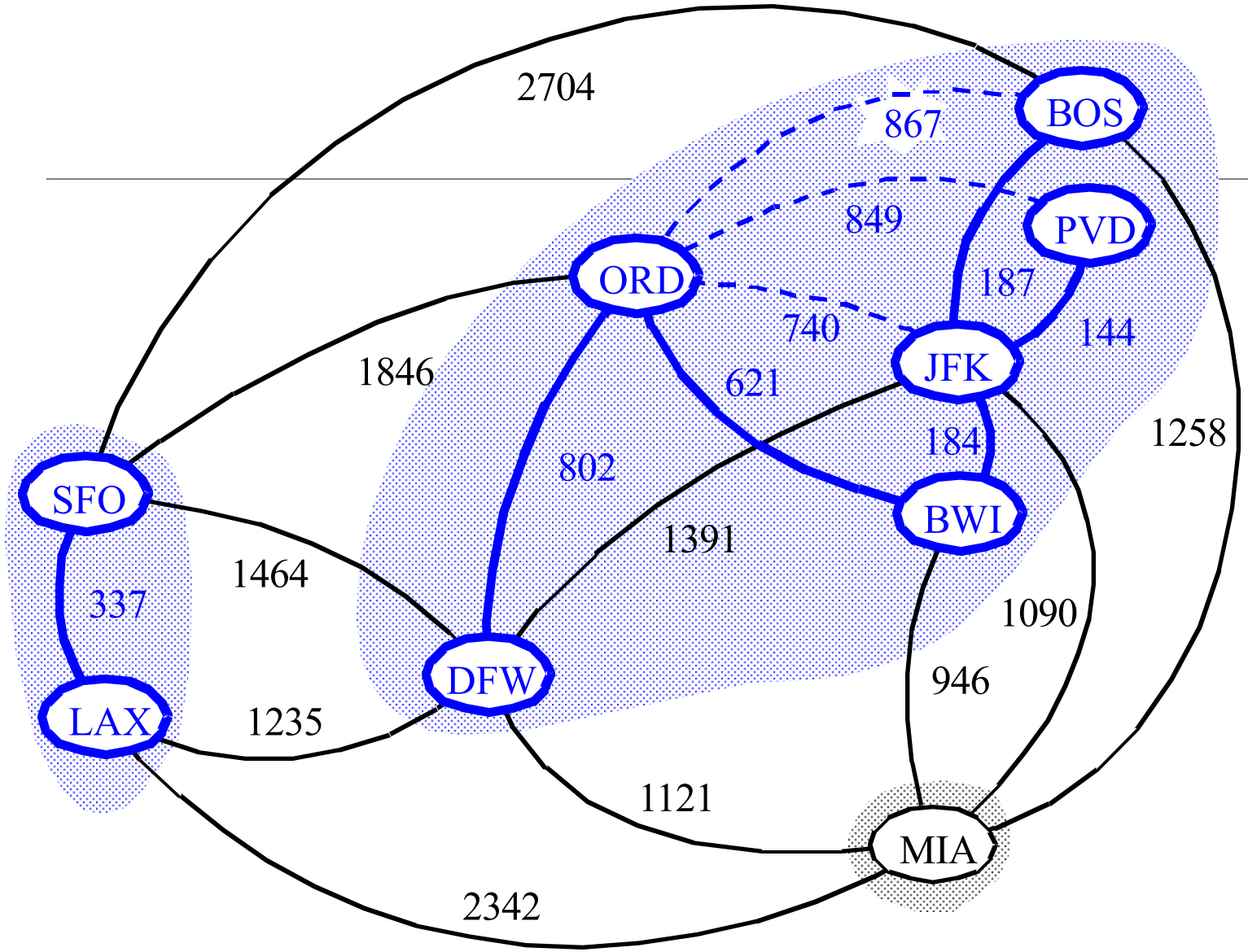

2704
867
BOS
849
PVD
ORD
187
144
740
JFK
1846
621
1258
184
802
SFO
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

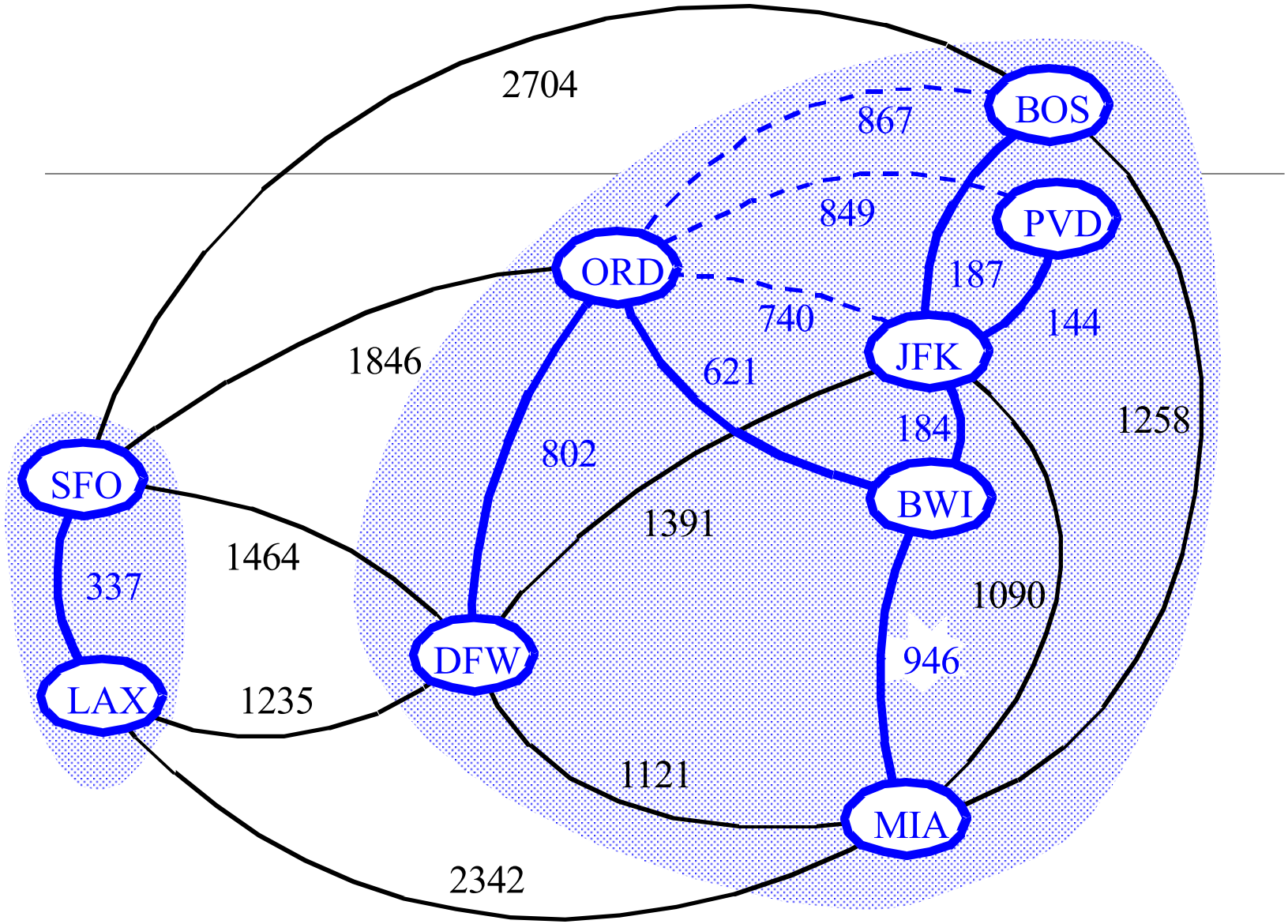

2704
867
BOS
849
PVD
ORD
187
740
144
1846
JFK
621
184
1258
802
SFO
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

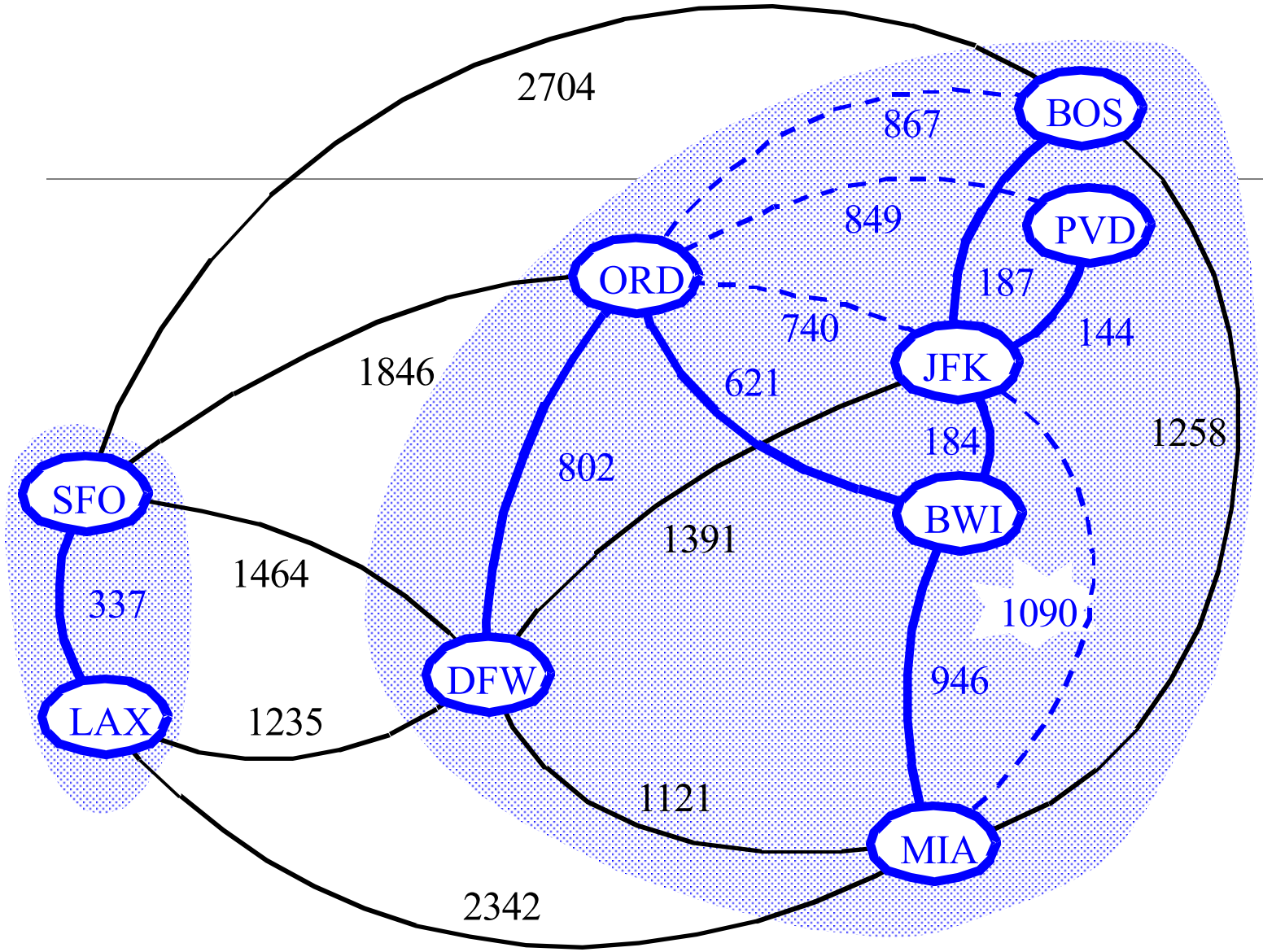

2704
867
BOS
849
PVD
ORD
187
740
144
JFK
1846
621
184
1258
802
SFO
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

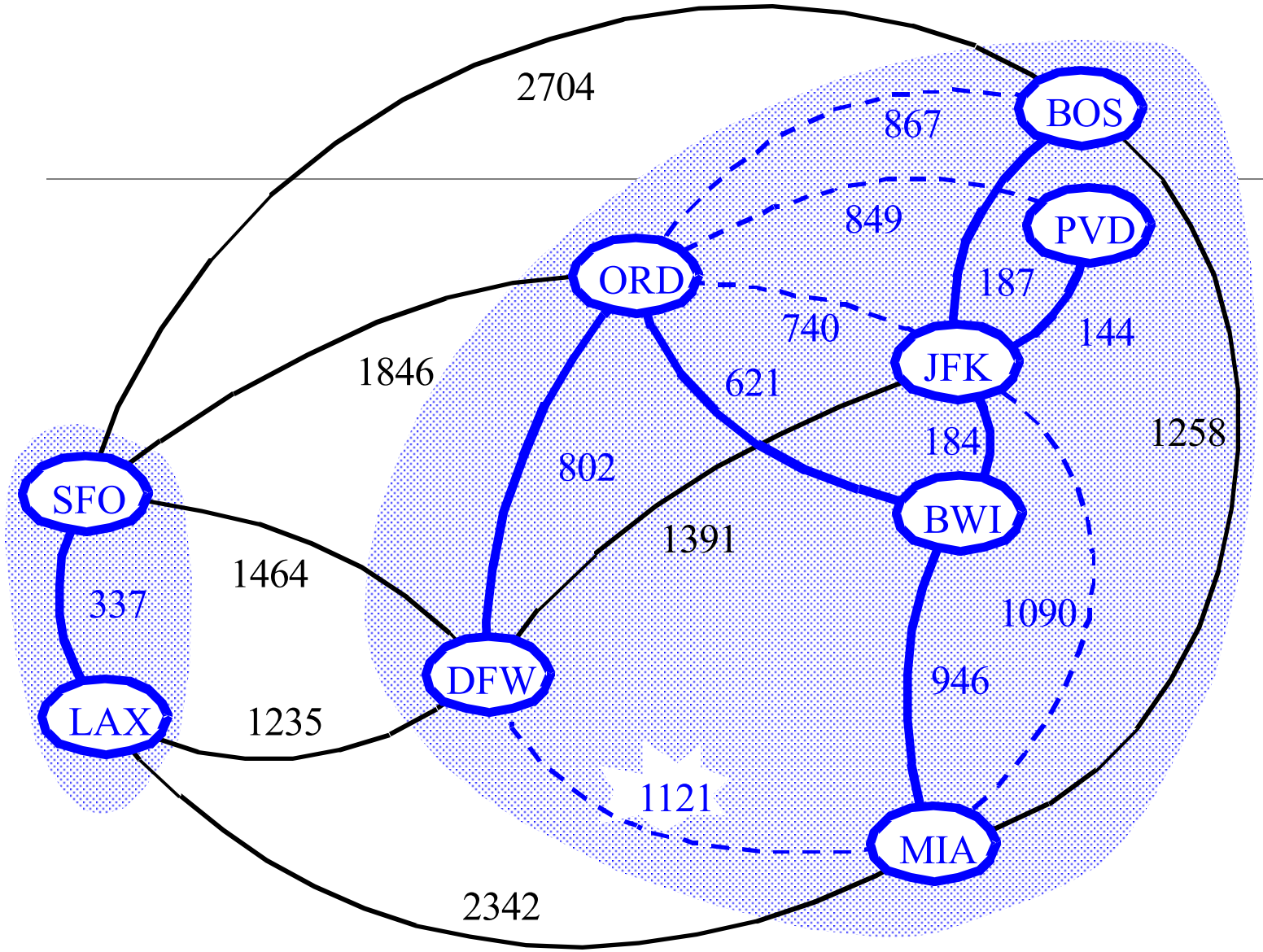

2704
867
BOS
849
PVD
ORD
187
740
144
JFK
1846
621
1258
184
802
SFO
BWI
1391
1464
337
1090
946
DFW
LAX
1235
1121
MIA
2342

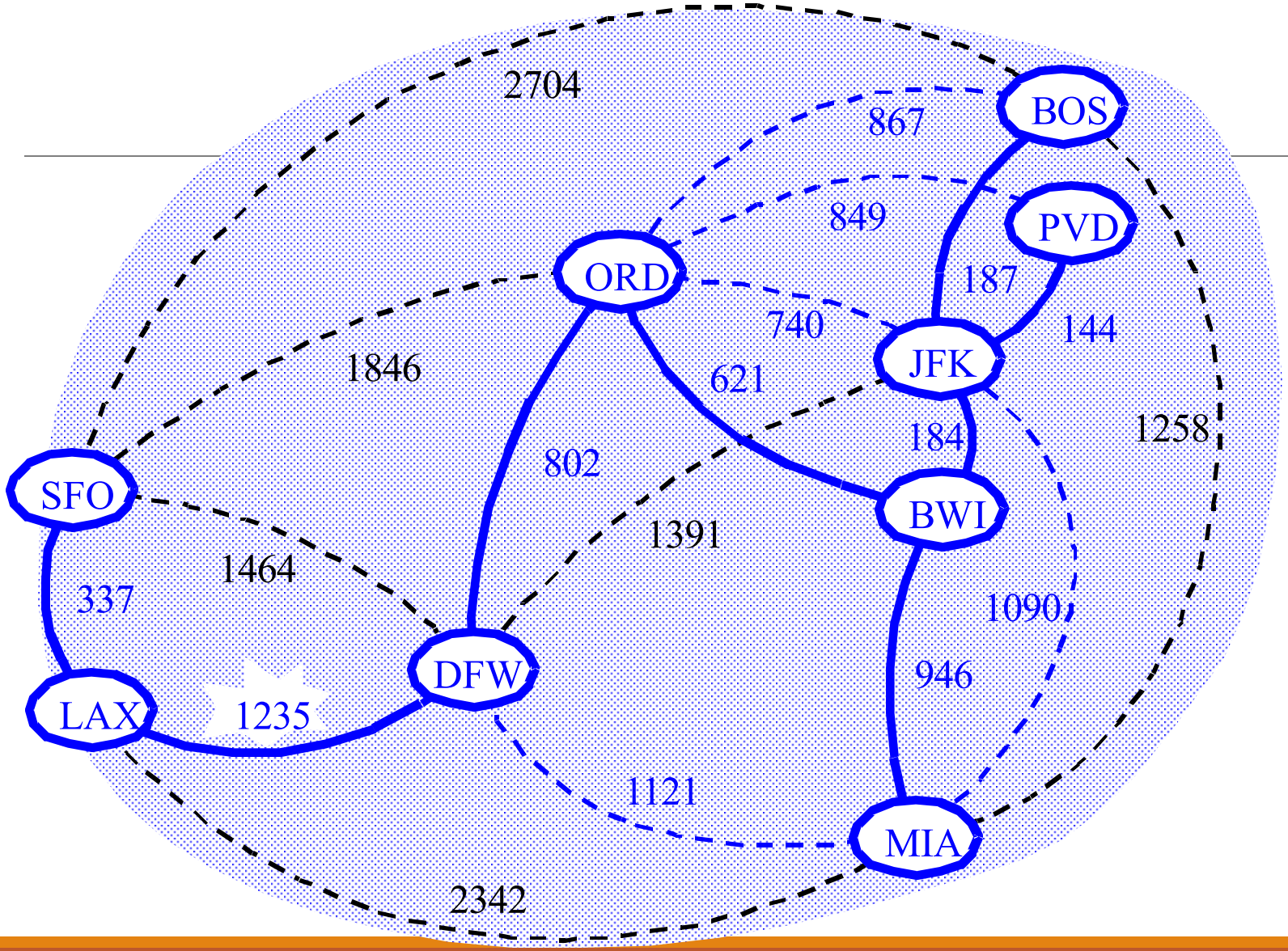

2704
867
BOS
849
PVD
ORD
187
740
144
JFK
1846
621
1258
184
802
SFO
BWI
1391
1464
337
1090
946
DFW
LAX
1235
1121
MIA
2342


2704
867
BOS
849
PVD
ORD
187
740
144
JFK
1846
621
1258
184
802
SFO
BWI
1391
1464
337
1090
946
DFW
LAX
1235
1121
MIA
2342


2704
867
BOS
849
PVD
ORD
187
740
144
JFK
1846
621
1258
184
802
SFO
BWI
1391
1464
337
1090
946
DFW
LAX
1235
1121
MIA
2342


2704
867
BOS
849
PVD
ORD
187
740
144
JFK
1846
621
1258
184
802
SFO
BWI
1391
1464
337
1090
946
DFW
LAX
1235
1121
MIA
2342


2704
867
BOS
849
PVD
ORD
187
740
144
JFK
1846
621
1258
184
802
SFO
BWI
1391
1464
337
1090
946
DFW
LAX
1235
1121
MIA
2342


2704
867
BOS
849
PVD
ORD
187
740
144
JFK
1846
621
1258
184
802
SFO
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

Costo Final

El costo es:

(BOS, JFK) + (PVD, JFK) + (JFK, BWI) + (BWI, ORD) + (BWI, MIA) + (ORD, DFW) + (DFW, LAX) + (LAX, SFO) =

187 + 144 + 184 + 946 + 621 + 802 + 1235 + 337 = 4456

Caminos más cortos

Problemas del camino más corto

- El problema del camino más corto consiste en encontrar un camino para llegar de manera más rápida (menor costo sumando las aristas recorridas) de un vértice origen a uno destino

- Se pueden encontrar varia opciones de éste tipo de problemas:
  - Camino más corto con un solo origen (Algoritmo de Dijkstra)
  - Camino más corto entre todos los nodos (Algoritmos de Floyd y Warshall)

Camino más corto con un solo origen

Suponer que se tiene un grafo dirigido *G=(V,A)* en el cual cada arista tiene una etiqueta no negativa, y donde un determinado vértice se especifica como origen

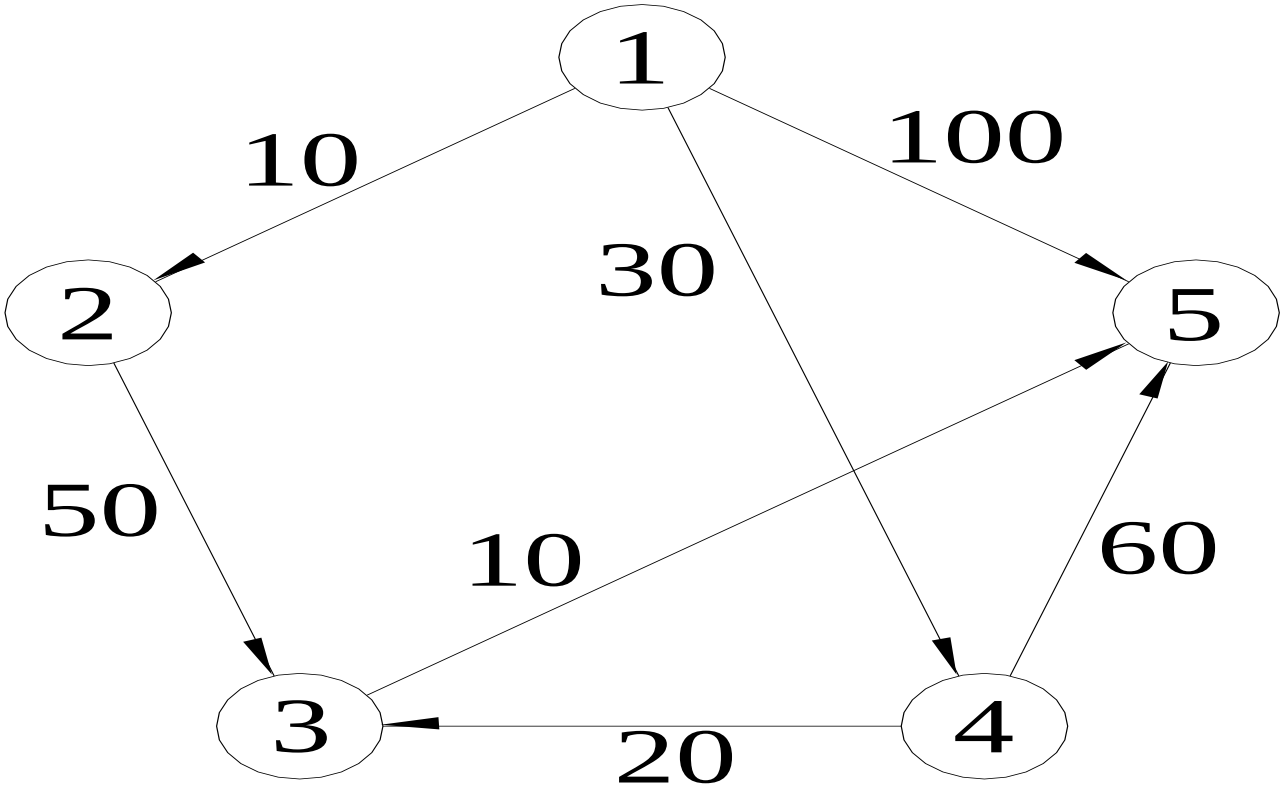
El problema es determinar el costo del camino más corto del origen a todos los demás vértices de *V*

Donde la longitud de un camino es la suma de los costos de las aristas que forman parte del camino

Algortimo de Dijkstra

- Opera a partir de un conjunto S de vértices cuya distancia más corta desde el origen es ya conocida
- En principio, *S* contiene sólo el vértice de origen
- En cada paso, se agrega algún vértice restante *v* a *S*, cuya distancia desde el origen es la más corta posible

Ejemplo



Resultado

Iteración	S	w	D[2]	D[3]	D[4]	D[5]
Inicial	$\{1\}$	—	10	∞	30	100
1	$\{1,2\}$	2	10	60	30	100
2	$\{1,2,4\}$	4	10	50	30	90
3	$\{1,2,4,3\}$	3	10	50	30	60
4	$\{1,2,4,3,5\}$	5	10	50	30	60

- Caminos
- {1,2} (10)
- {1,4,3} (50)
- {1,4} (30)
- {1,4,3,5} (60)

Camino más corto entre todos los pares

Útil cuando se desea construir una tabla que brinde el menor costo para llegar de un punto a cualquier otro

Este es el problema del camino mínimo entre un conjunto de pares (CMCP)

El problema consiste en encontrar el camino de longitud más corta entre *v* y *w* para cada par ordenado de vértices (*v,w*)

Algoritmo de R.W. Floyd

Usa una matriz A de n x n en la que se calculan las longitudes de los caminos más cortos

Inicialmente A[i][j] = G[i][j] para toda i ≠ j

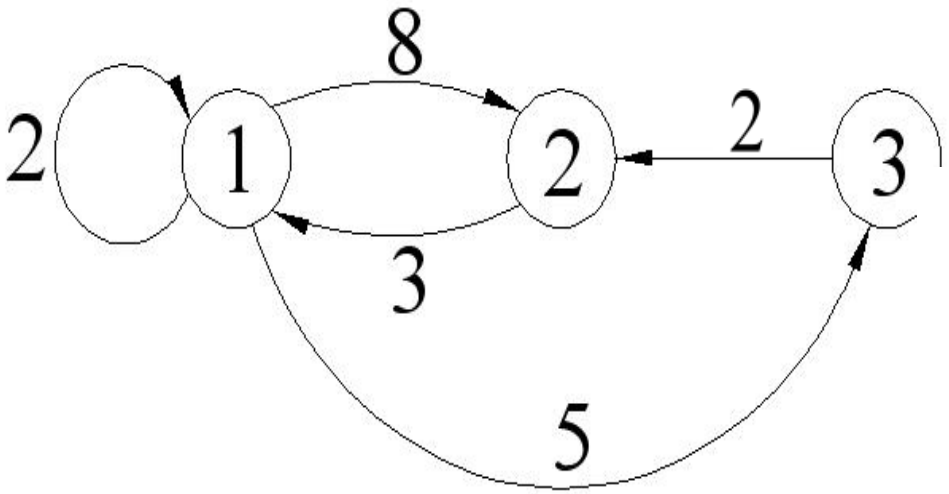
Se hacen *n* iteraciones a la matriz A, al final de la *k-ésima* iteración A[i][j] tendrá por valor la longitud más pequeña de cualquier camino que vaya desde *i* a *j* y que no pase por un número mayor de *k* vértices

Algoritmo de R.W. Floyd

En la *k-ésima* iteración se aplica la siguiente fórmula para calcular A

$$A_k[i][j] = \min \begin{cases} A_{k-1}[i][j] \\ A_{k-1}[i][k] + A_{k-1}[k][j] \end{cases}$$

Ejemplo



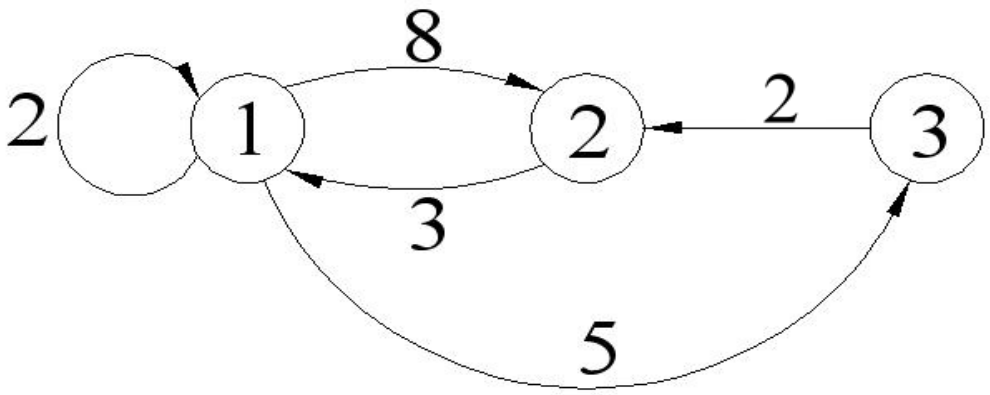
	1	2	3
1	0	7	5
2	3	0	8
3	5	2	0

Cerradura transitiva

- Variante del algoritmo del algoritmo de Floyd
- Conocido como el algoritmo de Warshall
- Se utiliza para saber si existe un camino entre un par de nodos
- Puede obtenerse con un procedimiento similar al de Floyd aplicando la siguiente fórmula en el *k-ésimo* paso en la matriz booleana A:

$$A_k[i][j] \leftarrow A_{k-1}[i][k] \text{ AND } A_{k-1}[k][j]$$

Ejemplo



	1	2	3
1	FALSE	TRUE	TRUE
2	TRUE	FALSE	TRUE
3	TRUE	TRUE	FALSE