Graphs-Spanning Trees

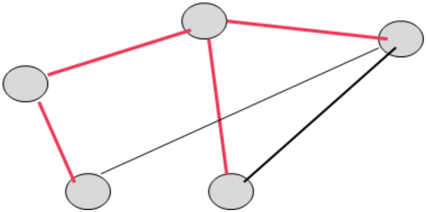
December 2, 2021

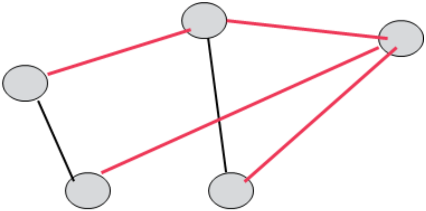
Topics

- Spanning Trees in Unweighted Graphs
- Minimum Spanning Trees in Weighted Graphs
    - Prim's Algorithm
    - Kruskal's Algorithm

Definitions

- Spanning tree: a tree that contains all vertices in the graph.
  - Sub graph of the original graph
  - Tree
  - Number of nodes: |V|
  - Number of edges: |V|-1





Spanning trees for unweighted graphs - data structures

- A table (an array) T
  - *size* = number of vertices,
  - T_v= parent of vertex v
- Adjacency lists
- A queue of vertices to be processed

Algorithm - initialization

1. Choose a vertex S and store it in a queue, set
   a *counter* $= 0$ (counts the processed nodes), and
   $T_s = 0$ (to indicate the root),
   $T_i = -1$,$i \neq s$ (to indicate vertex not processed)

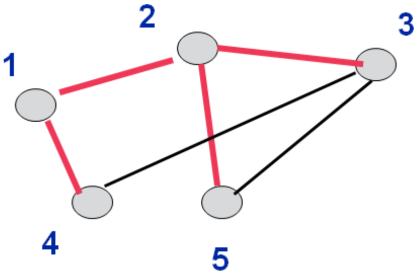
Algorithm - basic loop

2. While queue not empty and *counter* $< |V| - 1$:
   Read a vertex V from the queue
   For all adjacent vertices U :
   If $T_u = -1$ (not processed)
   $T_u \leftarrow V$
   $counter \leftarrow counter + 1$
   store U in the queue

Algorithm – results and complexity

- Result:
  - Root: S, such that Ts = 0
  - Edges in the tree: (Tv , V)
- Complexity: $O(|E| + |V|)$ - we process all edges and all nodes

Example



Table

0	1 is the root
1	edge (1,2)
2	edge (2,3)
1	edge (1,4)
2	edge (2,5)

Edge format: (Tv,V)

Definitions

- Minimum Spanning tree: a tree
  - that contains all vertices in the graph,
  - is connected,
  - is acyclic, and
  - has minimum total edge weight.

Applications of Minimum Spanning Trees

- Communication networks
- VLSI design
- Transportation systems

Minimum Spanning Tree - Prim's algorithm

- Given: Weighted graph.
  - Find a spanning tree with the minimal sum of the weights.
  - Similar to shortest paths in weighted graphs.
  - Difference: we record the weight of the current edge, not the length of the path .

Data Structures

- Three arrays:
  - cost[v] = the weight of the shortest edge connecting v to a known vertex
  - path[v] = the last vertex to cause a change in cost[v]
  - known[v] = True, if vertex v is fixed in the tree, False otherwise

Data Structures (cont.)

- Adjacency lists
- A priority queue of vertices to be processed.
  - Priority of each vertex is determined by the weight of edge that links the vertex to its parent.
  - The priority may change if we change the parent, provided the vertex is not yet fixed in the tree.

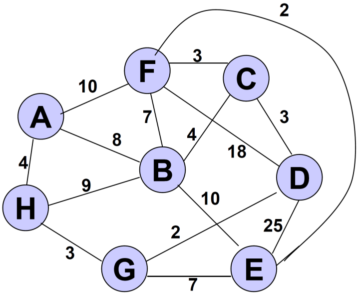
Prim's Algorithm

For all v
    $cost[v] = \infty$; $known[v] = false$; $path[v] = -$
$cost[s] = 0$
Insert s onto a priority queue that percolates vertices
    with lowest $cost[]$ to the top.
While priority queue not empty
    DeleteMin v from priority queue
    If not $known[v]$
      $known[v] = true$
      For all unknown successors w of v
        If $weight[v, w] < cost[w]$
          $cost[w] = weight[v, w]$
          $path[w] = v$
          insert w onto priority queue

Results and Complexity

- At the end of the algorithm, the tree would be represented with its edges
  - $\{(v, path[v])|v = 1, 2, \ldots, |V|\}$
- Complexity: $O(|E|log(|V|))$

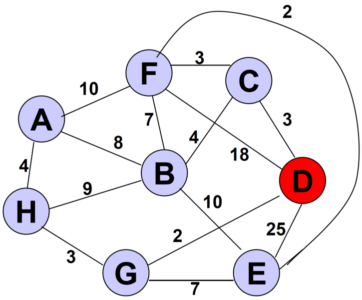
Example



Initialize array

	K	d_v	p_v
A	F	∞	–
B	F	∞	–
C	F	∞	–
D	F	∞	–
E	F	∞	–
F	F	∞	–
G	F	∞	–
H	F	∞	–

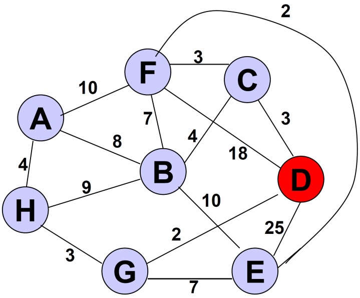
Example



Start with any node, say D

	K	d_v	p_v
A			
B			
C			
D	T	0	–
E			
F			
G			
H			

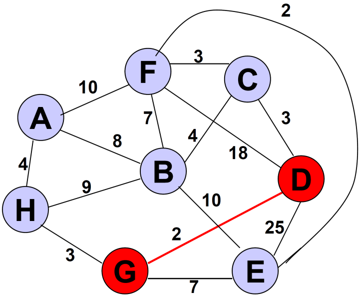
Example



Update distances of adjacent, unselected nodes

	K	d_v	p_v
A			
B			
C		3	D
D	T	0	–
E		25	D
F		18	D
G		2	D
H			

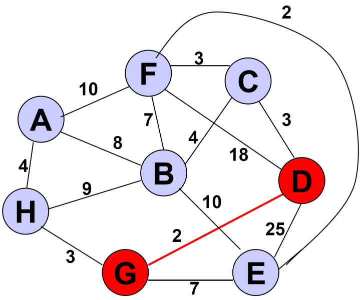
Example



Select node with minimum distance

	K	d_v	p_v
A			
B			
C		3	D
D	T	0	–
E		25	D
F		18	D
G	T	2	D
H			

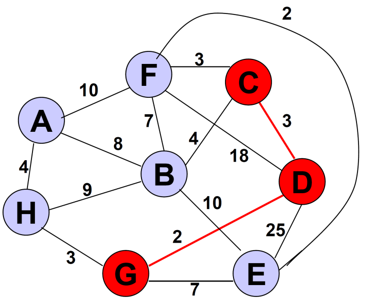
Example



Update distances of adjacent, unselected nodes

	K	d_v	p_v
A			
B			
C		3	D
D	T	0	–
E		7	G
F		18	D
G	T	2	D
H		3	G

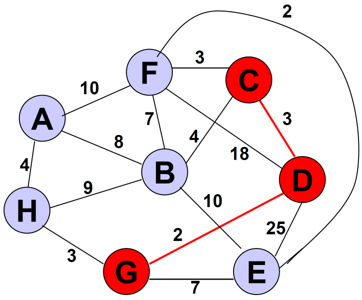
Example



Select node with minimum distance

	K	d_v	p_v
A			
B			
C	T	3	D
D	T	0	–
E		7	G
F		18	D
G	T	2	D
H		3	G

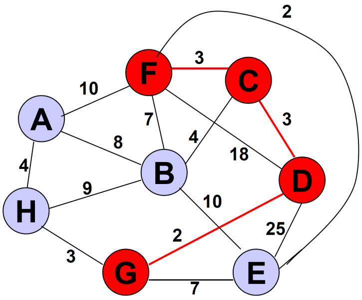
Example



Update distances of adjacent, unselected nodes

	K	d_v	p_v
A			
B		4	C
C	T	3	D
D	T	0	–
E		7	G
F		3	C
G	T	2	D
H		3	G

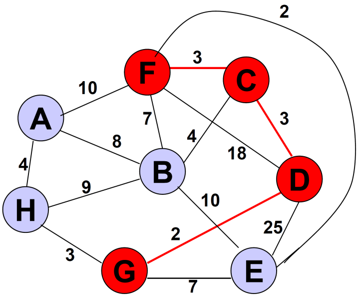
Example



Select node with minimum distance

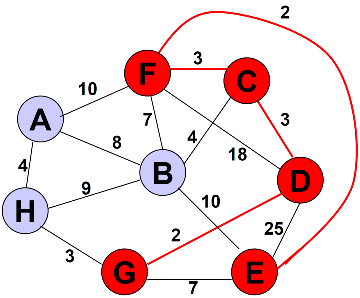
	K	d_v	p_v
A			
B		4	C
C	T	3	D
D	T	0	–
E		7	G
F	T	3	C
G	T	2	D
H		3	G

Example



Update distances of adjacent, unselected nodes

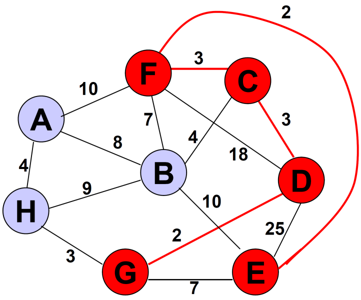
	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E		2	F
F	T	3	C
G	T	2	D
H		3	G



Select node with minimum distance

	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H		3	G

Example

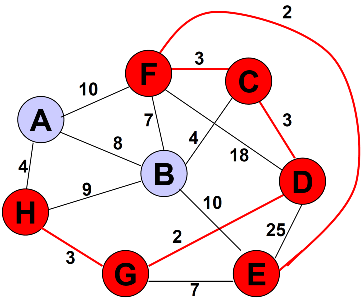


Update distances of adjacent, unselected nodes

	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H		3	G

Table entries unchanged

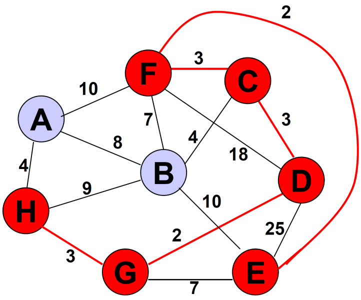
Example



Select node with minimum distance

	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

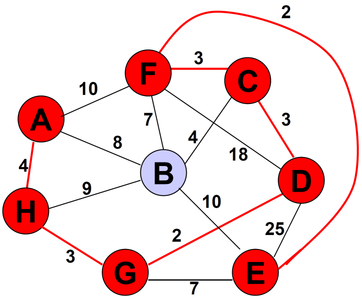
Example



Update distances of adjacent, unselected nodes

	K	d_v	p_v
A		4	H
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

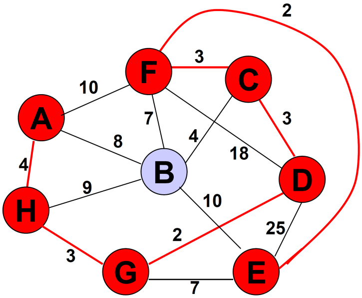
Example



Select node with minimum distance

	K	d_v	p_v
A	T	4	H
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

Example

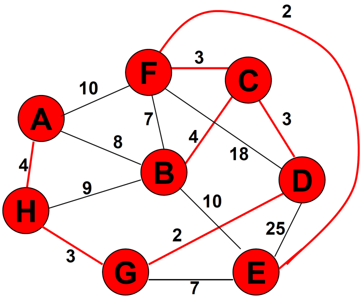


Update distances of adjacent, unselected nodes

	K	d_v	p_v
A	T	4	H
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

Table entries unchanged

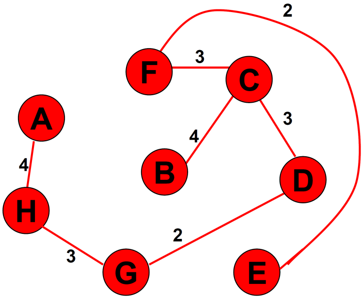
Example



Select node with minimum distance

	K	d_v	p_v
A	T	4	H
B	T	4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

Example



Cost of Minimum
Spanning Tree = Σ d_v = **21**

	K	d_v	p_v
A	T	4	H
B	T	4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

Done

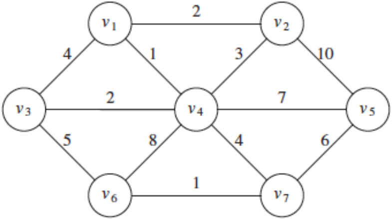
Kruskal's Algorithm

- The algorithm works with :
  - set of edges,
  - tree forests,
  - each vertex belongs to only one tree in the forest.

The Algorithm

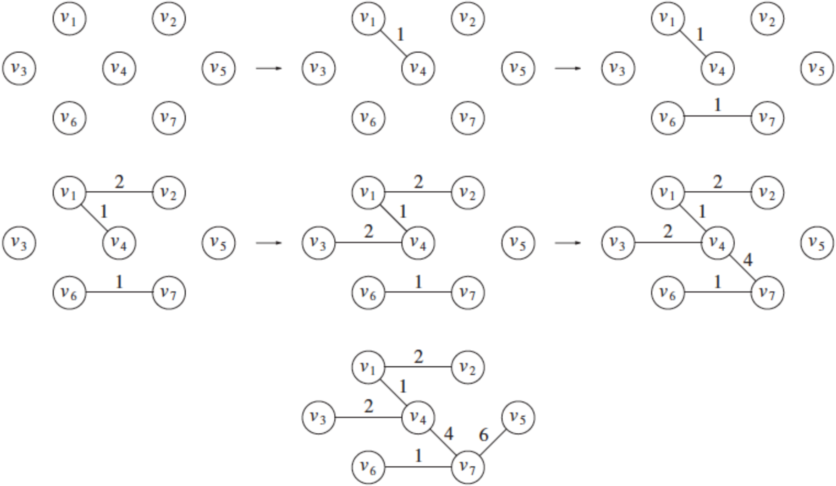
1. Build |V| trees of one vertex only - each vertex is a tree of its own. Store edges in priority queue
2. Choose an edge (u,v) with minimum weight
   if u and v belong to one and the same tree,
     do nothing
   if u and v belong to different trees,
     link the trees by the edge (u,v)
3. Perform step 2 until all trees are combined into one tree only

Example

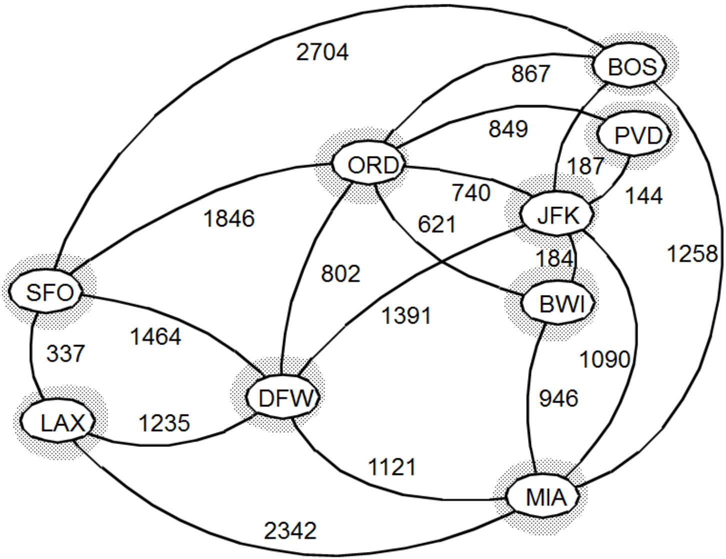


Edge	Weight	Action
(v_1, v_4)	1	Accepted
(v_6, v_7)	1	Accepted
(v_1, v_2)	2	Accepted
(v_3, v_4)	2	Accepted
(v_2, v_4)	3	Rejected
(v_1, v_3)	4	Rejected
(v_4, v_7)	4	Accepted
(v_3, v_6)	5	Rejected
(v_5, v_7)	6	Accepted

Example

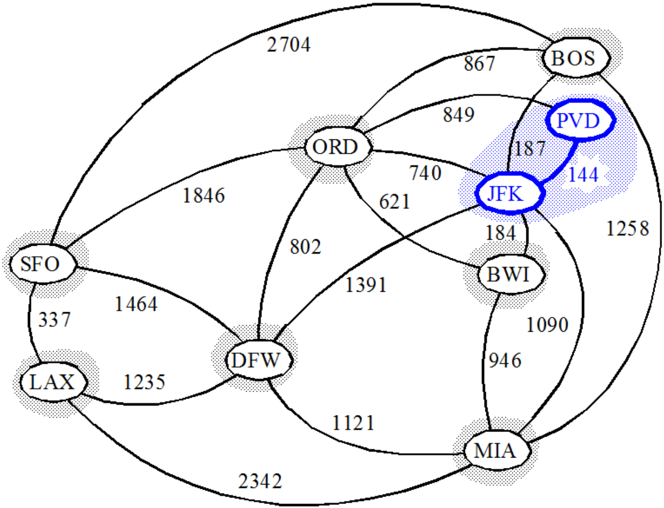


Example



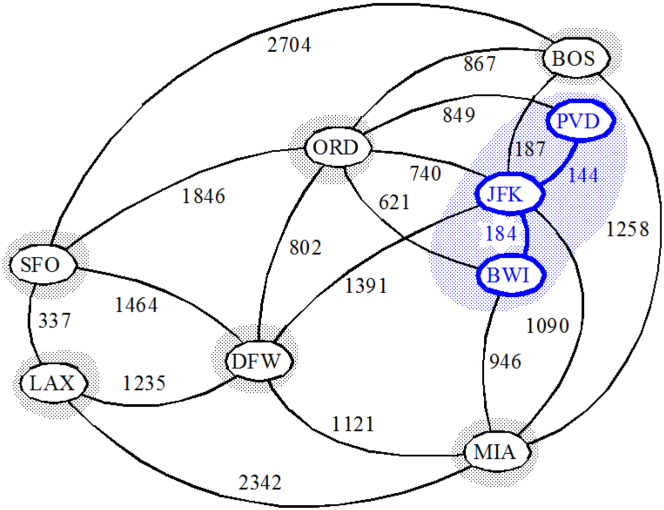
2704
867
BOS
849
PVD
ORD
187
740
144
1846
621
JFK
184
1258
SFO
802
BWI
1391
1464
337
1090
946
DFW
LAX
1235
1121
MIA
2342

Example



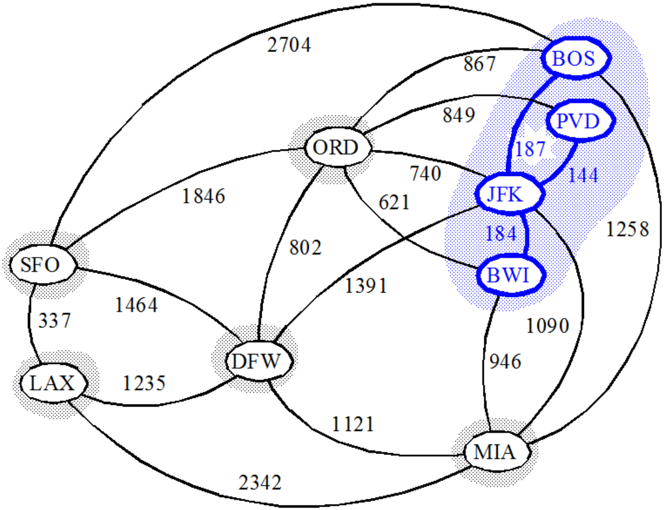
2704
BOS
867
849
PVD
ORD
187
740
144
1846
JFK
621
1258
184
802
SFO
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

Example



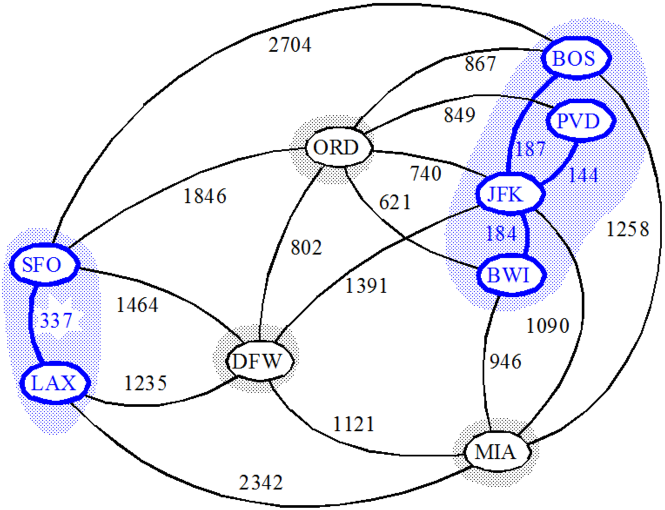
2704
BOS
867
849
PVD
ORD
187
740
144
1846
JFK
621
1258
184
SFO
802
BWI
1391
1464
337
1090
946
DFW
LAX
1235
1121
MIA
2342

Example

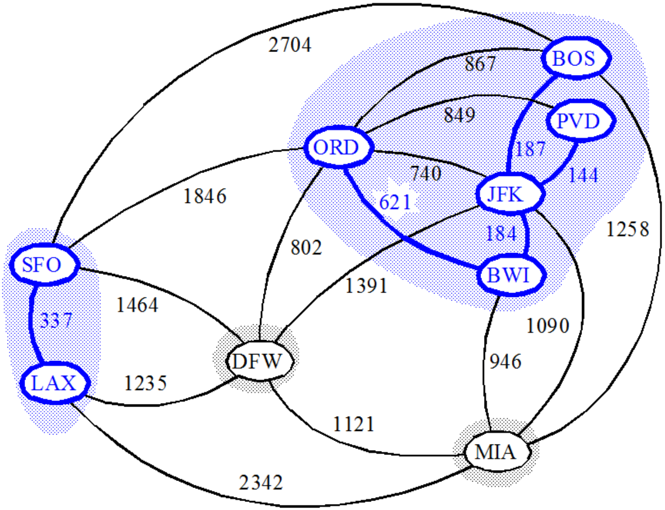


2704
867
BOS
849
PVD
187
ORD
144
740
JFK
1846
621
184
1258
802
SFO
BWI
1391
1464
337
1090
946
DFW
LAX
1235
1121
MIA
2342

Example

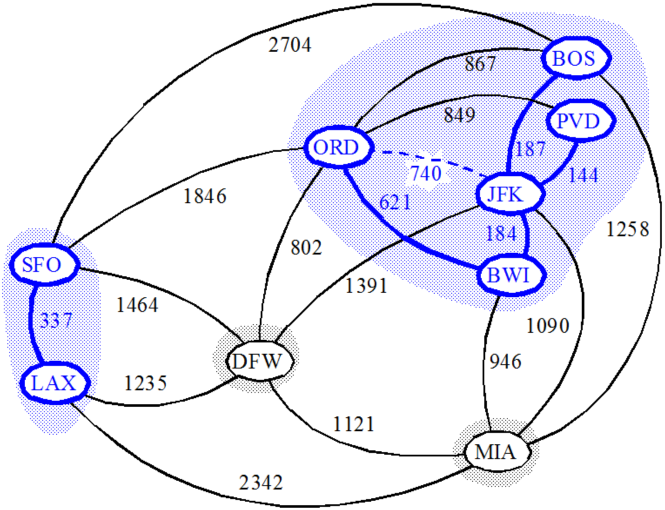

2704
867
BOS
849
PVD
187
ORD
740
144
1846
621
JFK
1258
184
802
SFO
BWI
1391
1464
337
1090
946
DFW
LAX
1235
1121
MIA
2342

Example



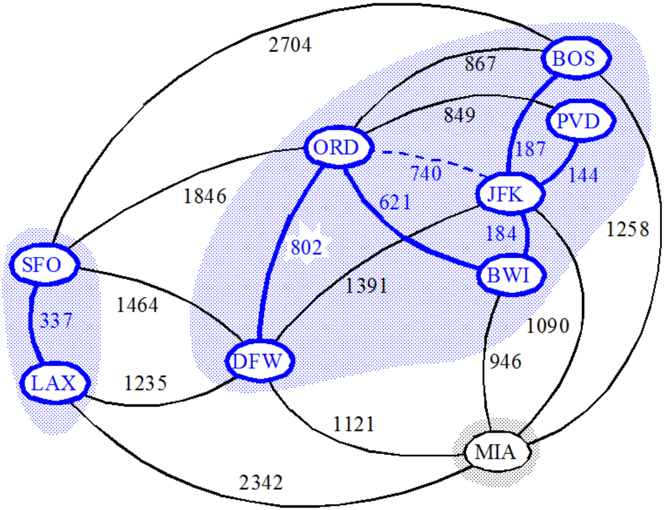
2704
BOS
867
849
PVD
ORD
187
1846
740
144
JFK
621
1258
184
802
SFO
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

Example

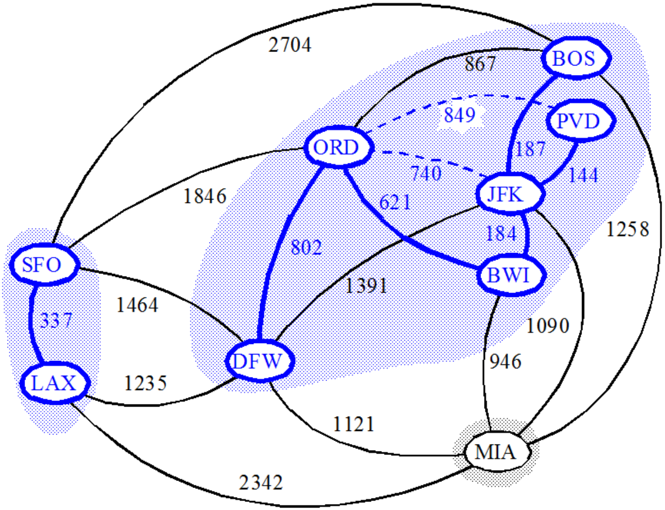


2704
867
BOS
849
PVD
ORD
187
740
144
1846
621
JFK
1258
802
184
SFO
BWI
1391
1464
1090
337
946
DFW
LAX
1235
1121
MIA
2342

Example

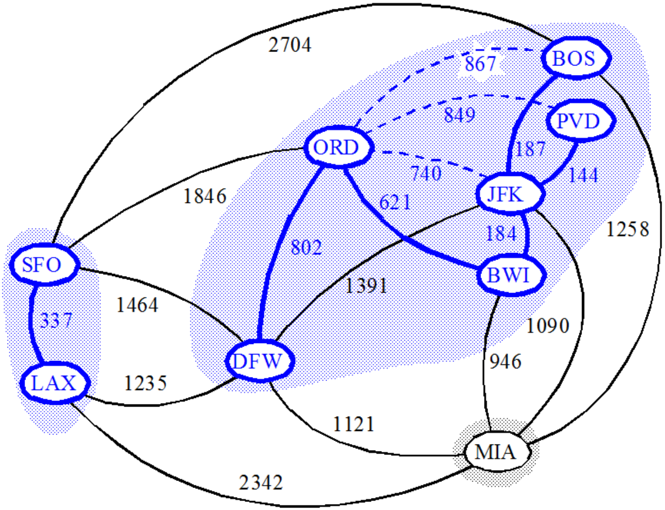

2704
867
BOS
849
PVD
ORD
187
740
144
1846
621
JFK
184
1258
802
SFO
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

Example



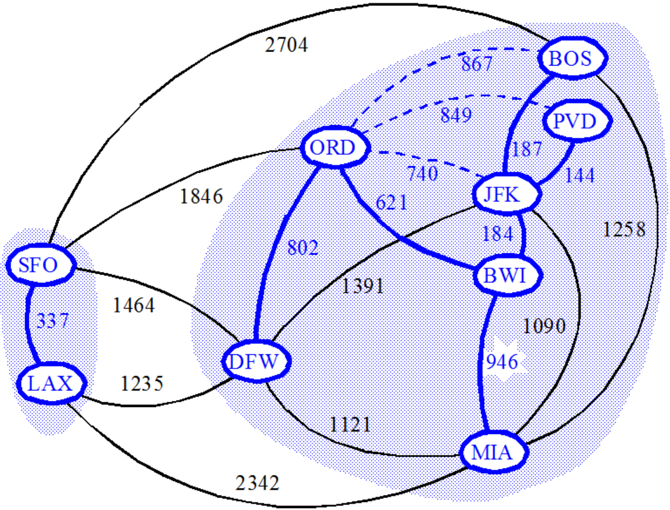
2704
867
BOS
849
PVD
ORD
187
740
144
1846
JFK
621
1258
SFO
802
184
BWI
1391
1464
337
1090
946
DFW
LAX
1235
1121
MIA
2342

Example



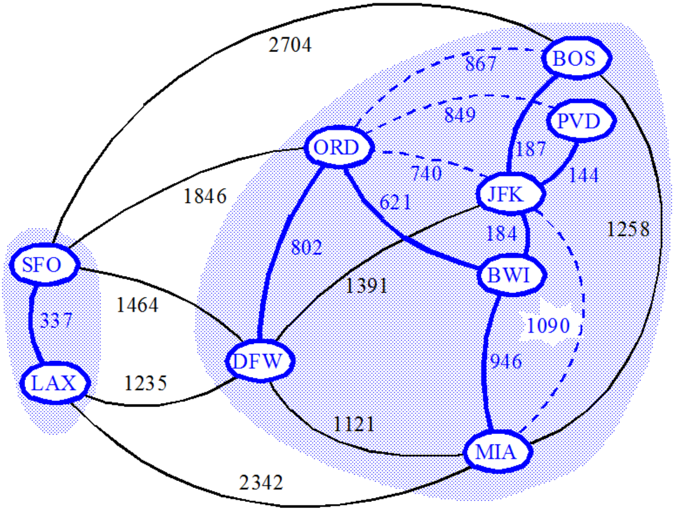
2704
867
BOS
849
PVD
ORD
187
740
144
1846
JFK
621
1258
184
SFO
802
BWI
1391
1464
1090
337
DFW
946
LAX
1235
1121
MIA
2342

Example



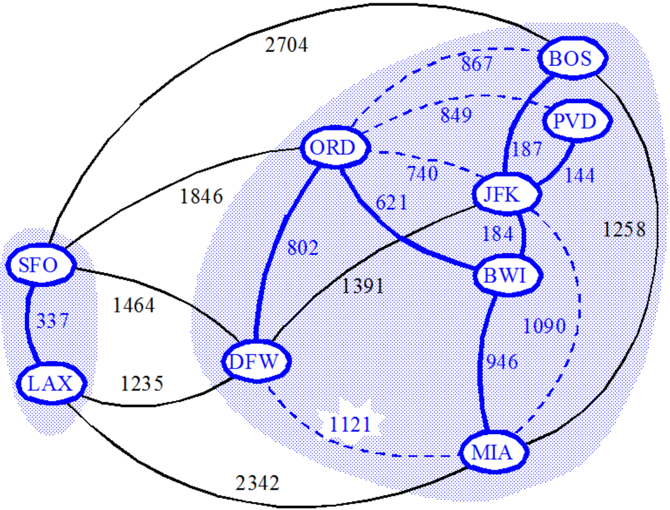
2704
867
BOS
849
PVD
ORD
187
740
144
1846
JFK
621
1258
184
802
SFO
BWI
1391
1464
1090
337
946
DFW
LAX
1235
1121
MIA
2342

Example



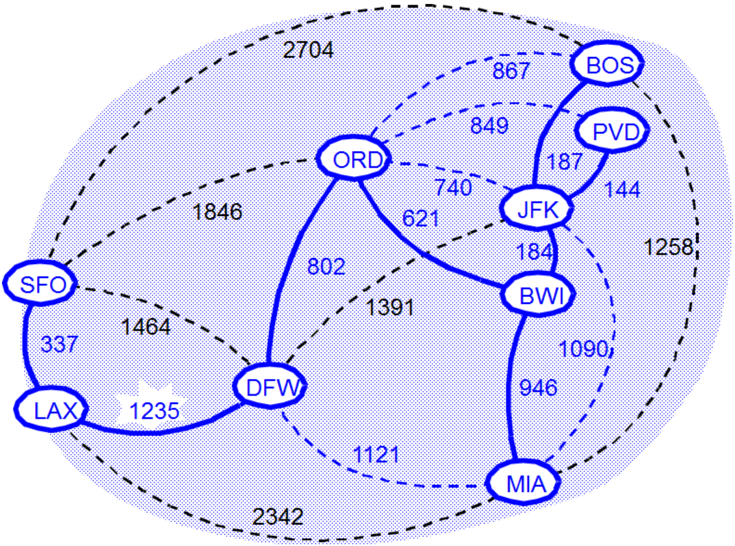
2704
867
BOS
849
PVD
ORD
187
740
144
1846
JFK
621
1258
184
802
SFO
BWI
1391
1464
1090
337
946
DFW
LAX
1235
1121
MIA
2342

Example



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

Example



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

Operations on Disjoint Sets

- Comparison: Are the vertices in the same tree?
- Union: Combine two disjoint sets to form one set - the new tree
- Implementation
  - Union/find operations: the unions are represented as trees.
  - The set of trees is implemented by an array.

Complexity of Kruskal's Algorithm

- The complexity is determined by the heap operations and the union/find operations
- Union/find operations on disjoint sets represented as trees: tree search operations, with complexity $O(log|V|)$
- DeleteMin in Binary heaps with N elements is $O(logN)$,
- When performed for N elements, it is $O(NlogN)$.

Complexity of Kruskal's Algorithm (Cont.)

- Kruskal's algorithm works with a binary heap that contains all edges: $O(|E|log(|E|))$
- However, a complete graph has
$$|E| = |V| \times (|V| - 1)/2, \text{ i.e. } |E| = O(|V|^2)$$
- Thus for the binary heap we have $O(|E|log(|E|)) = O(|E|log(|V|^2) = O(2|E|log(|V|)$ $= O(|E|log(|V|))$
- Since for each edge we perform DeleteMin operation and union/find operation, the overall complexity is:
$$O(|E|(log(|V|) + log(|V|)) = O(|E|log(|V|))$$

Discussion

- Sparse trees - Kruskal's algorithm :
  - Guided by edges
- Dense trees - Prim's algorithm :
  - The process is limited by the number of the processed vertices