Shortest Path Algorithms

Kuan-Yu Chen (陳冠宇)

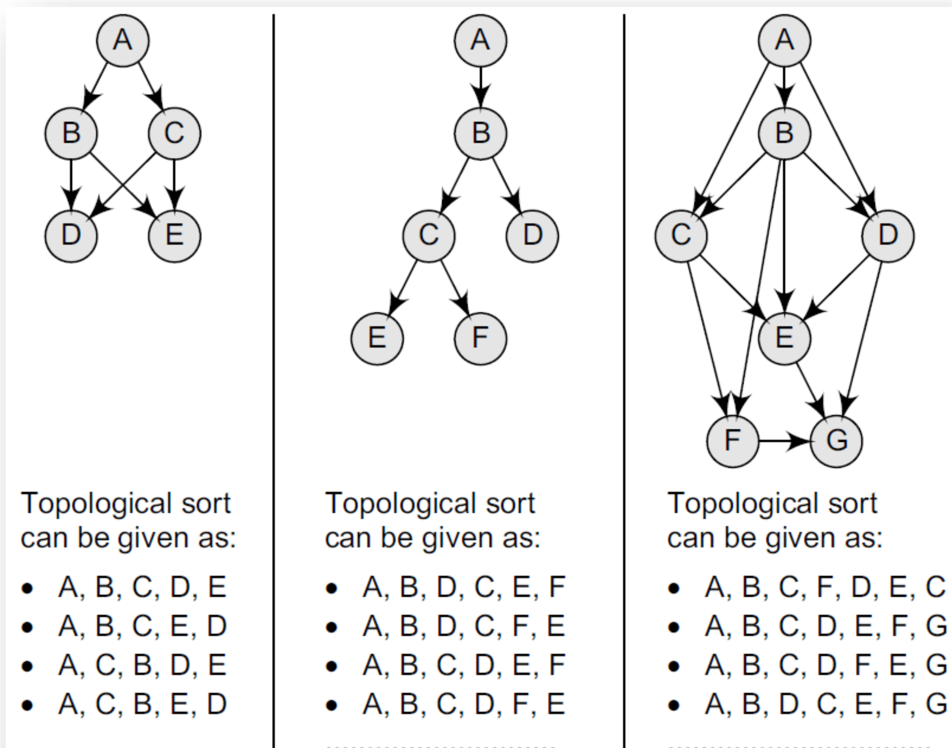
2018/12/05 @ TR-212, NTUST

Review

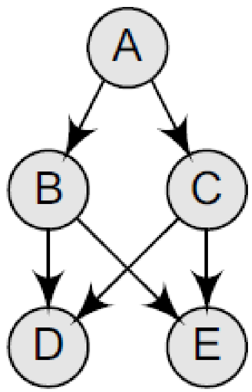
- Bi-connected components
  - Articulation Point
- Bridge
- Three common ways of storing graphs
  - Sequential representation
    - adjacency matrix
  - Linked representation
    - linked list
  - Adjacency multi-list
- Traversal algorithms
  - BFS
  - DFS

Topological Sorting.

- Topological sort of a **directed acyclic graph** (DAG) G is defined as a linear ordering of its nodes in which each node comes before all nodes to which it has outbound edges
  - Every DAG has one or more number of topological sorts
  - If G contains an edge (u, v), then u appears before v in the ordering

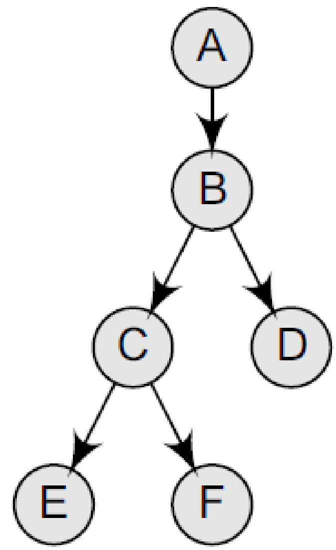


Topological Sorting..



Topological sort can be given as:

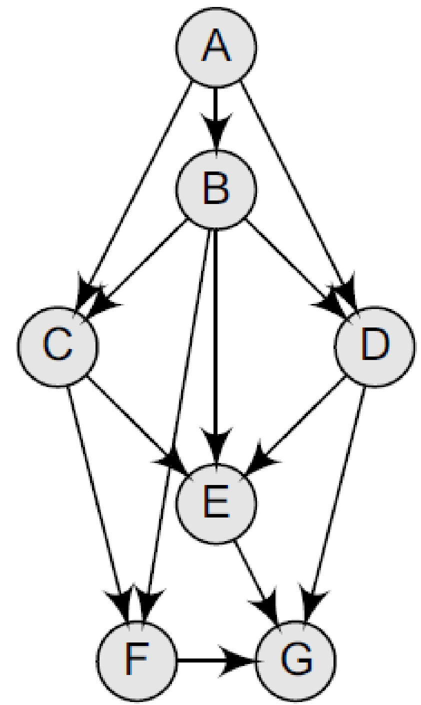
- A, B, C, D, E
- A, B, C, E, D
- A, C, B, D, E
- A, C, B, E, D



Topological sort can be given as:

- A, B, D, C, E, F
- A, B, D, C, F, E
- A, B, C, D, E, F
- A, B, C, D, F, E

....



Topological sort can be given as:

- A, B, C, F, D, E, C
- A, B, C, D, E, F, G
- A, B, C, D, F, E, G
- A, B, D, C, E, F, G

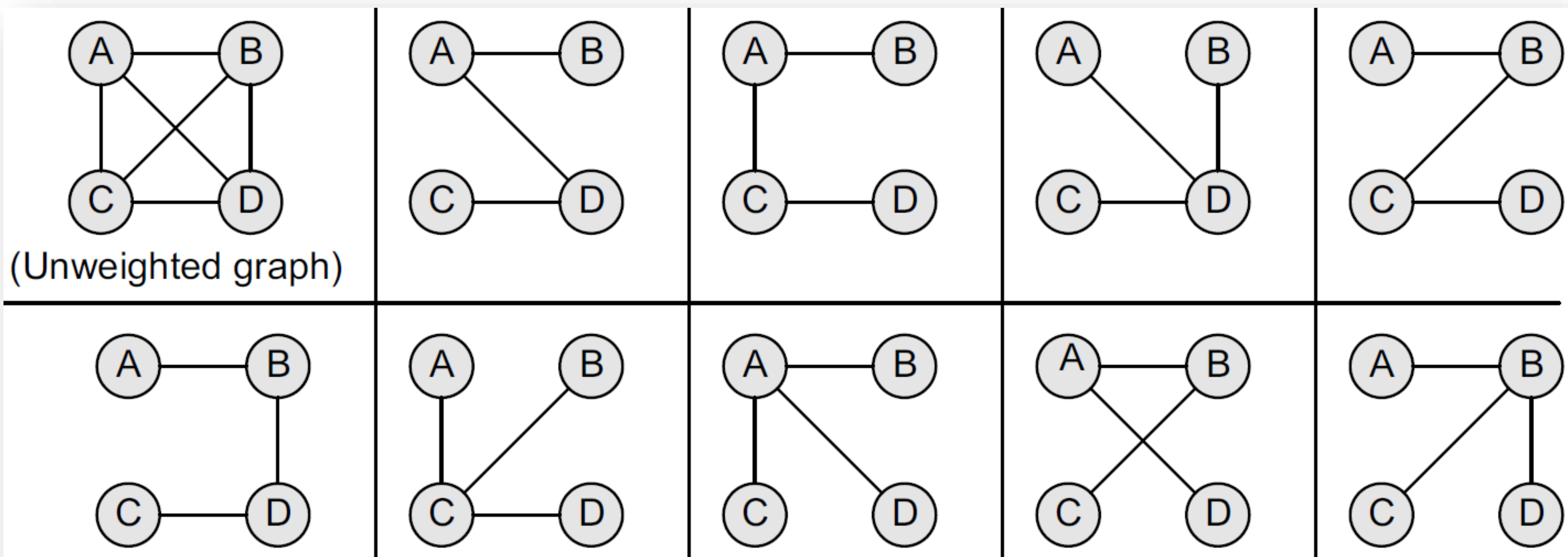
....

Shortest Path Algorithms

- The representative algorithms, which are used to calculate the shortest path, are:
  - Minimum spanning tree
    - Prim's algorithm
    - Kruskal's algorithm
  - Dijkstra's algorithm

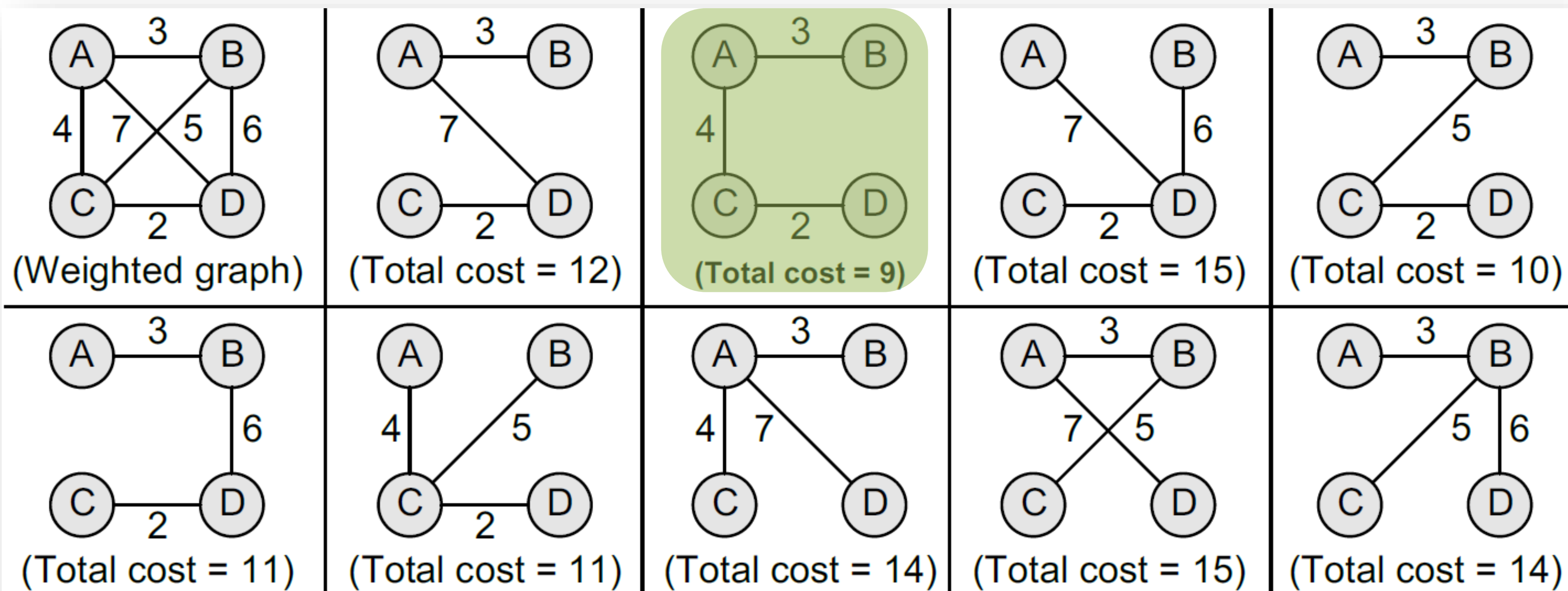
Spanning Tree

- A spanning tree of a connected, undirected graph G is a sub-graph of G which is a tree that connects all the vertices together
  - A graph G can have many different spanning trees



Minimum Spanning Tree.

- A **minimum spanning tree** (MST) is defined as a spanning tree with weight less than or equal to the weight of every other spanning tree
  - We can assign **weights** to each edge, and use it to assign a weight to a spanning tree by calculating the sum of the weights of the edges in that spanning



Minimum Spanning Tree..

- Properties
  - ***Possible multiplicity***
    - There can be multiple minimum spanning trees of the same weight
    - Particularly, if all the weights are the same, then every spanning tree will be minimum
  - ***Uniqueness***
    - When each edge in the graph is assigned a different weight, then there will be only one unique minimum spanning tree
  - ***Simplicity***
    - For an unweighted graph, any spanning tree is a minimum spanning tree

Minimum Spanning Tree…

- Minimum spanning trees can be computed quickly and easily to provide optimal solutions
  - Prim's algorithm
  - Kruskal's algorithm

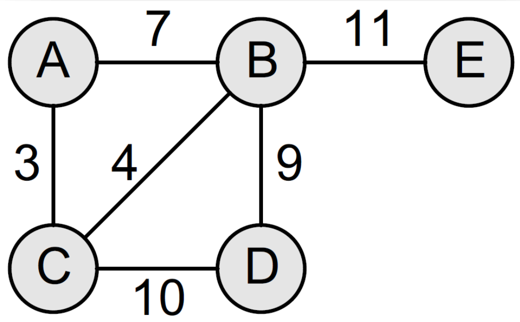
Prim's Algorithm.

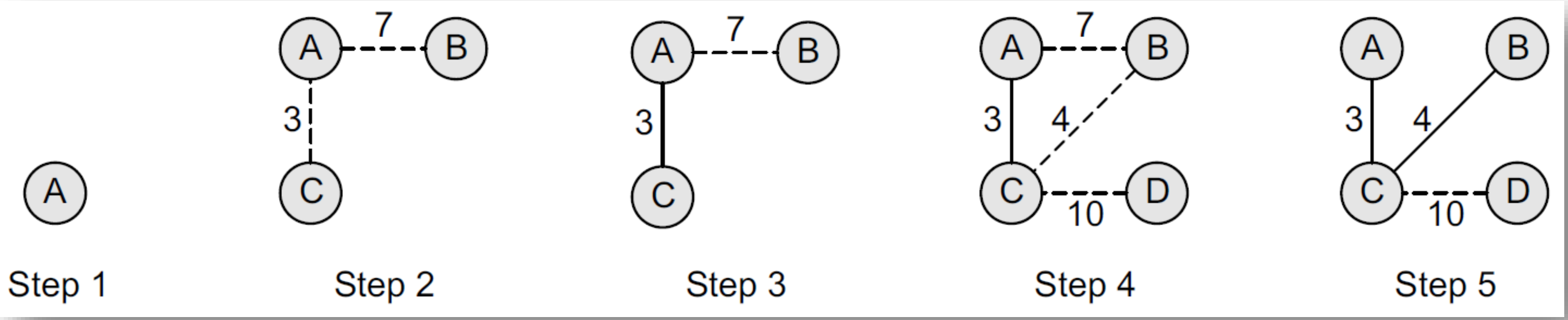
- Prim's algorithm is a greedy algorithm that is used to form a minimum spanning tree for a connected weighted undirected graph
  - **Tree vertices**
    - Vertices that are a part of the minimum spanning tree T
  - **Fringe (Neighboring) vertices**
    - Vertices that are currently not a part of T, but are adjacent to some tree vertex
  - **Unseen vertices**
    - Vertices that are neither tree vertices nor fringe vertices fall under this category

```
Step 1: Select a starting vertex
Step 2: Repeat Steps 3 and 4 until there are fringe vertices
Step 3:     Select an edge e connecting the tree vertex and
            fringe vertex that has minimum weight
Step 4:     Add the selected edge and the vertex to the
            minimum spanning tree T
        [END OF LOOP]
Step 5: EXIT
```

Prim's Algorithm..

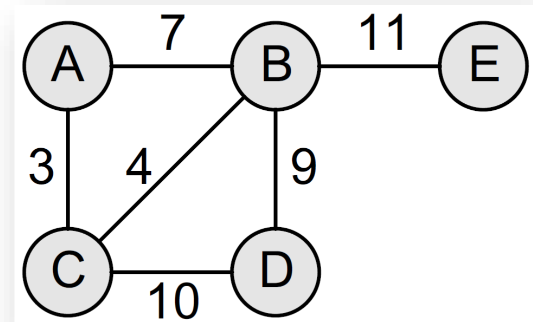
- Construct a minimum spanning tree of the graph by using Prim's algorithm
  - Step 1: Choose a starting vertex A
  - Step 2: Add the fringe vertices (that are adjacent to A)
  - Step 3: Since the edge connecting A and C has less weight, add C to the tree
  - Step 4: Add the fringe vertices (that are adjacent to C)
  - Step 5: Since the edge connecting C and B has less weight, add B to the tree

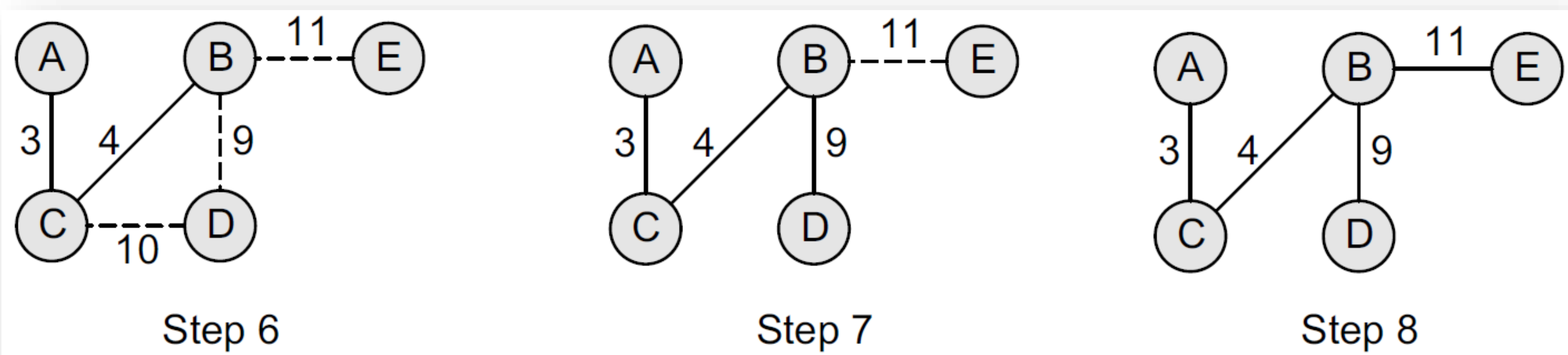




Prim's Algorithm…

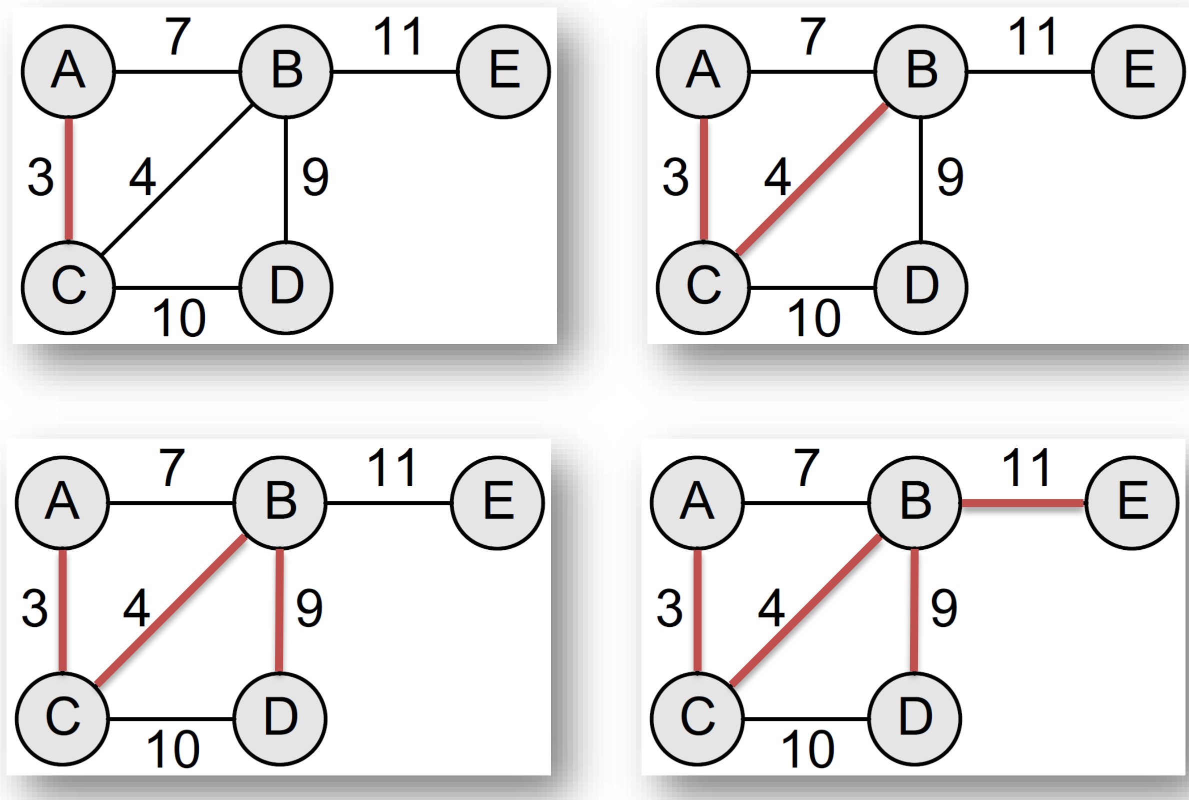
- Step 6: Add the fringe vertices (that are adjacent to B)
- Step 7: Since the edge connecting B and D has less weight, add D to the tree
- Step 8: Add E to the tree





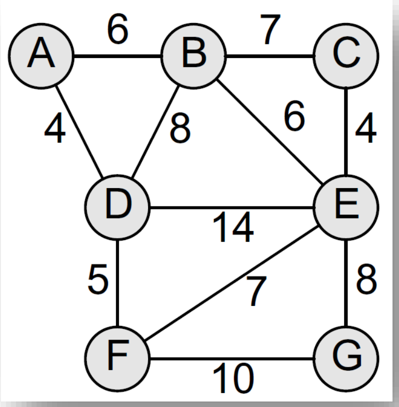
Prim's Algorithm….

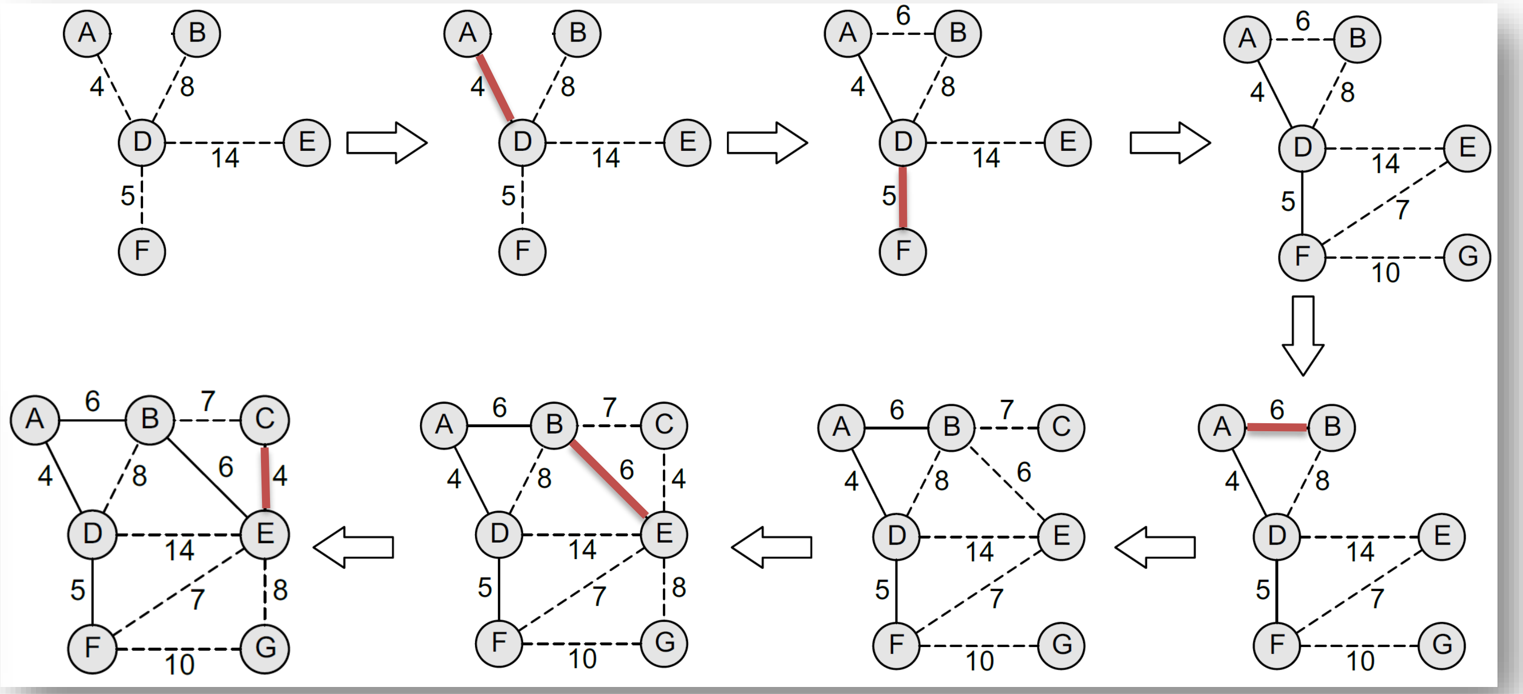
- By looking!

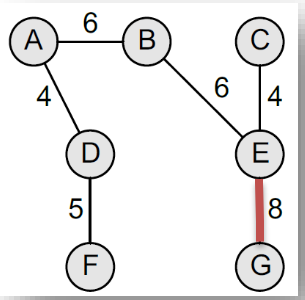


Prim’s Algorithm…..

- Construct a minimum spanning tree of the graph by using Prim’s algorithm from vertex *D*







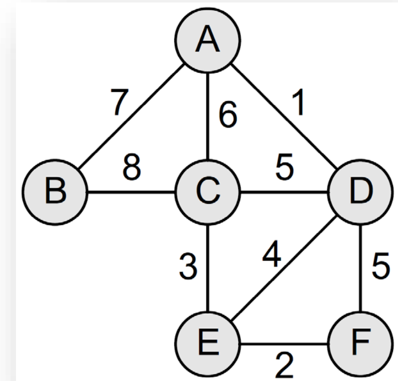
Kruskal's Algorithm.

- Kruskal's algorithm is used to find the minimum spanning tree for a connected weighted graph
  - If the graph is not connected, then it finds a **minimum spanning forest**

```
Step 1: Create a forest in such a way that each graph is a separate
        tree.
Step 2: Create a priority queue Q that contains all the edges of the
        graph.
Step 3: Repeat Steps 4 and 5 while Q is NOT EMPTY
Step 4:      Remove an edge from Q
Step 5: IF the edge obtained in Step 4 connects two different trees,
        then Add it to the forest (for combining two trees into one
        tree).
        ELSE
             Discard the edge
Step 6: END
```

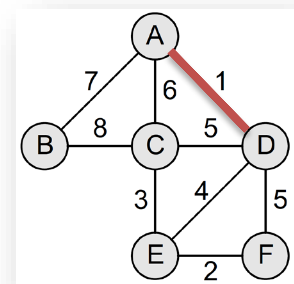
Kruskal's Algorithm..

- Apply Kruskal's algorithm on the given graph
  - Initial:
    - F = {{A}, {B}, {C}, {D}, {E}, {F}}
    - MST = {}
    - Priority Queue Q = {(A, D), (E, F), (C, E), (E, D) (C, D), (D, F), (A, C), (A, B), (B, C)}



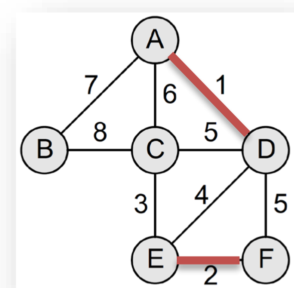
  - Step1:
    - Remove the edge (A, D) from Q

F = {{A, D}, {B}, {C}, {E}, {F}}
MST = {A, D}
Q = {(E, F), (C, E), (E, D), (C, D), (D, F), (A, C), (A, B), (B, C)}



  - Step2:
    - Remove the edge (E, F) from Q

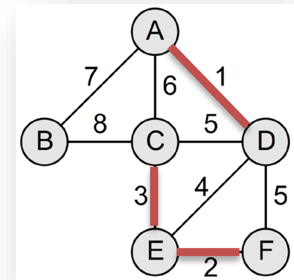
F = {{A, D}, {B}, {C}, {E, F}}
MST = {(A, D), (E, F)}
Q = {(C, E), (E, D), (C, D), (D, F), (A, C), (A, B), (B, C)}



Kruskal’s Algorithm...

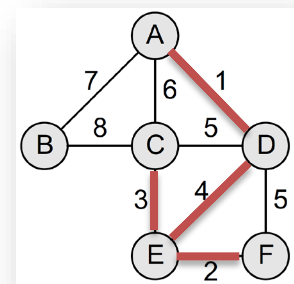
– Step3:
  - Remove the edge (C, E) from Q

F = {{A, D}, {B}, {C, E, F}}
MST = {(A, D), (C, E), (E, F)}
Q = {(E, D), (C, D), (D, F), (A, C), (A, B), (B, C)}



– Step4:
  - Remove the edge (E, D) from Q

F = {{A, C, D, E, F}, {B}}
MST = {(A, D), (C, E), (E, F), (E, D)}
Q = {(C, D), (D, F), (A, C), (A, B), (B, C)}



– Step5:
  - Remove the edge (C, D) from Q

    The edge does not connect different trees, so simply discard this edge

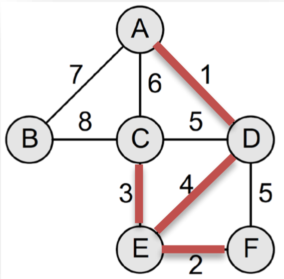
F = {{A, C, D, E, F}, {B}}
MST = {(A, D), (C, E), (E, F), (E, D)}
Q = {(D, F), (A, C), (A, B), (B, C)}

Kruskal's Algorithm….

- Step6:
  - Remove the edge (D, F) from Q

    The edge does not connect different trees, so simply discard this edge

$$
\begin{aligned}
F &= \{\{A, C, D, E, F\}, \{B\}\} \\
MST &= \{(A, D), (C, E), (E, F), (E, D)\} \\
Q &= \{(A, C), (A, B), (B, C)\}
\end{aligned}
$$



- Step7:
  - Remove the edge (A, C) from Q

    The edge does not connect different trees, so simply discard this edge

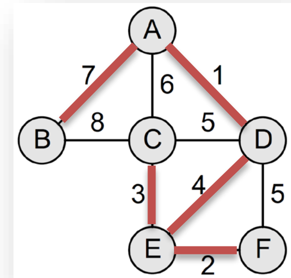
$$
\begin{aligned}
F &= \{\{A, C, D, E, F\}, \{B\}\} \\
MST &= \{(A, D), (C, E), (E, F), (E, D)\} \\
Q &= \{(A, B), (B, C)\}
\end{aligned}
$$

Kruskal’s Algorithm…..

– Step8:

- Remove the edge (A, B) from Q

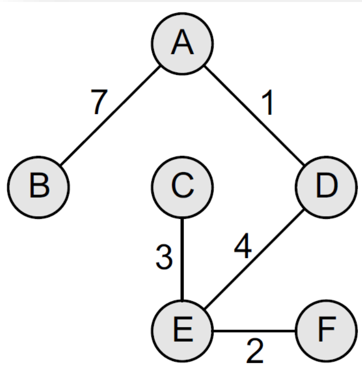
F = {A, B, C, D, E, F}
MST = {(A, D), (C, E), (E, F), (E, D), (A, B)}
Q = {(B, C)}



– Step8:

- Remove the edge (B, C) from Q

The edge does not connect different trees, so simply discard this edge



F = {A, B, C, D, E, F}
MST = {(A, D), (C, E), (E, F), (E, D), (A, B)}
Q = {}

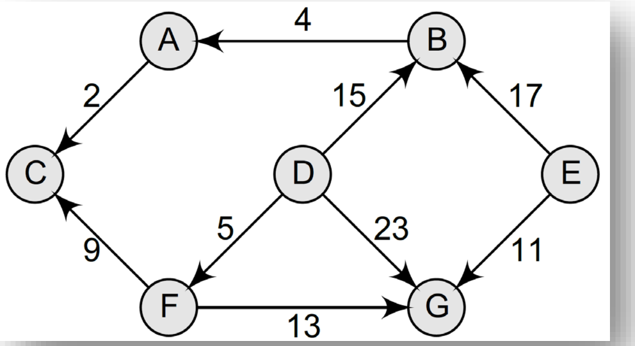
Dijkstra's Algorithm.

- Dijkstra's algorithm, given by a Dutch scientist Edsger Dijkstra in 1959, is used to find the shortest path tree
  - Given a graph *G* and a source node *A*, the algorithm is used to find the shortest path (one having the lowest cost) between *A* (source node) and every other node

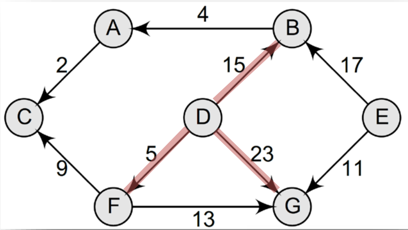
```
1. Select the source node also called the initial node
2. Define an empty set N that will be used to hold nodes to which a shortest
   path has been found.
3. Label the initial node with 0, and insert it into N.
4. Repeat Steps 5 to 7 until the destination node is in N or there are no more
   labelled nodes in N.
5. Consider each node that is not in N and is connected by an edge from
   the newly inserted node.
6. (a) If the node that is not in N has no label then SET the label of the
       node = the label of the newly inserted node + the length of the edge.
   (b) Else if the node that is not in N was already labelled, then SET its new
       label = minimum (label of newly inserted vertex + length of edge, old
       label)
7. Pick a node not in N that has the smallest label assigned to it and add it
   to N.
```

Dijkstra's Algorithm..

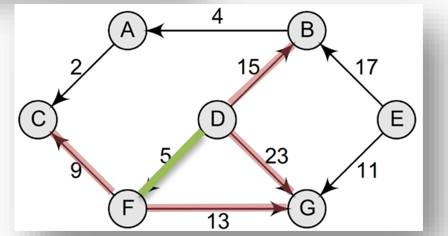
- Given a graph G, please take D as the initial node, and execute the Dijkstra's algorithm on it



  - Step 1:
    - Set the label of D = 0 and N = {D}



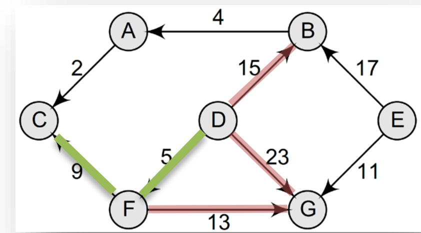
  - Step 2:
    - Label of B = 15, G = 23, and F = 5
    - Therefore, N = {D, F}



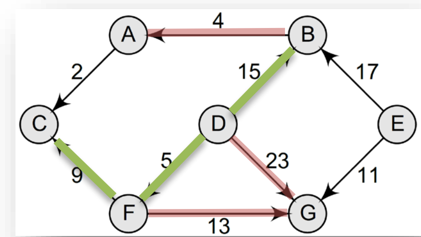
  - Step 3:
    - Label of B = 15, G has been re-labelled 18 because minimum(5 + 13, 23) = 18, C has been re-labelled 14 (5 + 9)
    - Therefore, N = {D,F, C}

Dijkstra's Algorithm...

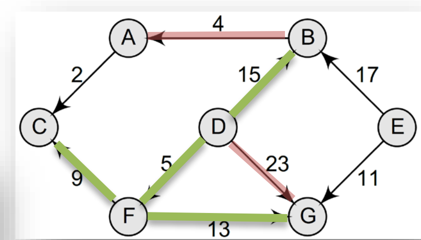
– Step 4:
  - Label of B = 15, G = 18
  - Therefore, N = {D, F, C, B}

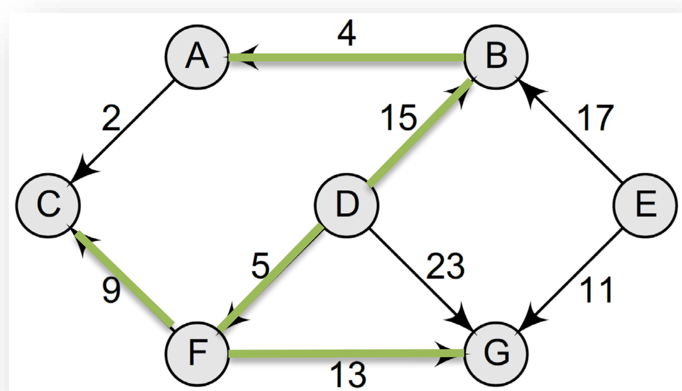


– Step 5:
  - Label of G = 18 and A = 19 (15 + 4)
  - Therefore, N = {D, F, C, B, G}



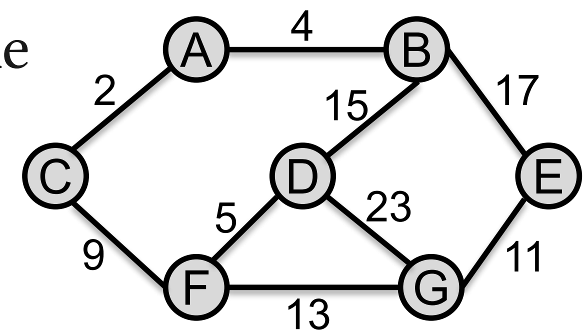
– Step 6: Label of A = 19
  - Therefore, N = {D, F, C, B, G, A}



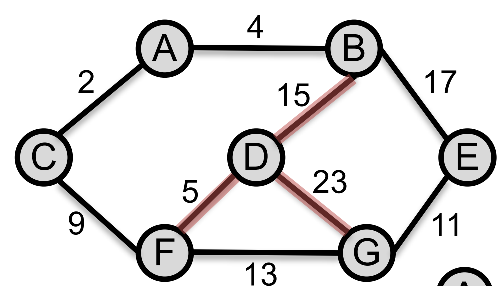


Dijkstra's Algorithm….

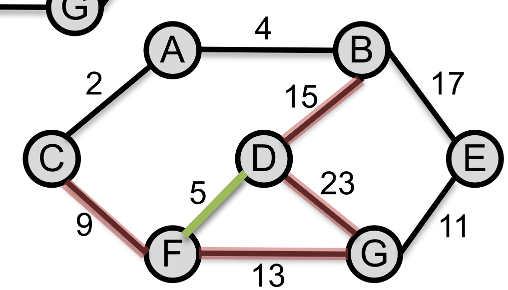
- Given a **undirected** graph G, please take D as the initial node, and execute the Dijkstra's algorithm on it



  - Step 1:
    - Set the label of D = 0 and N = {D}



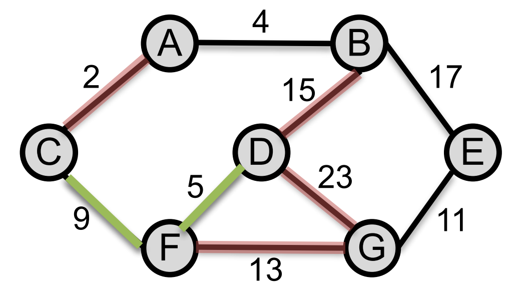
  - Step 2:
    - Label of B = 15, G = 23, and F = 5
    - Therefore, N = {D, F}



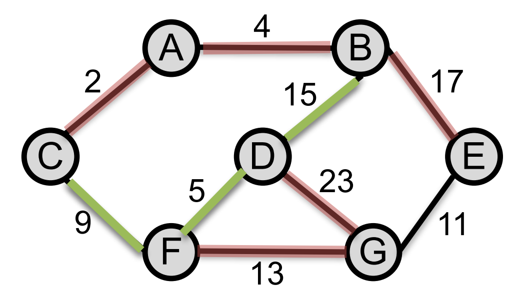
  - Step 3:
    - Label of B = 15, G has been re-labelled 18 because minimum(5 + 13, 23) = 18, C has been re-labelled 14 (5 + 9)
    - Therefore, N = {D,F, C}

Dijkstra's Algorithm…..

- Step 3:
  - Label of B = 15, G has been re-labelled 18 because minimum(5+13, 23) = 18, A = 16 (5+9+2)
  - Therefore, N = {D, F, C, B}

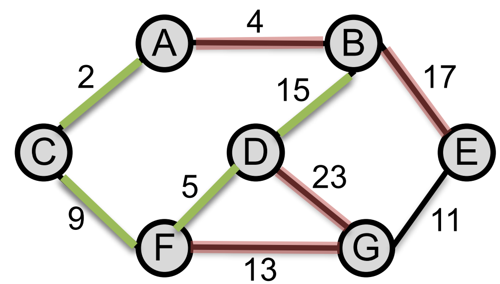


- Step 3:
  - Label of G has been re-labelled 18 because minimum(5+13, 23) = 18, A = minimum(5+9+2=16,15+4=19)=16, E = 32 (15+17)
  - Therefore, N = {D, F, C, B, A}

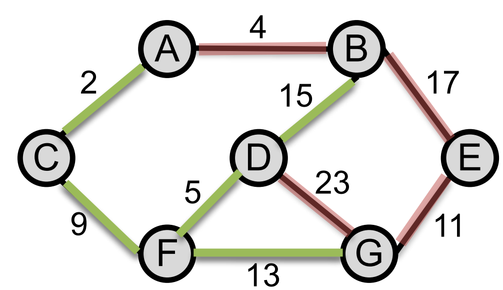


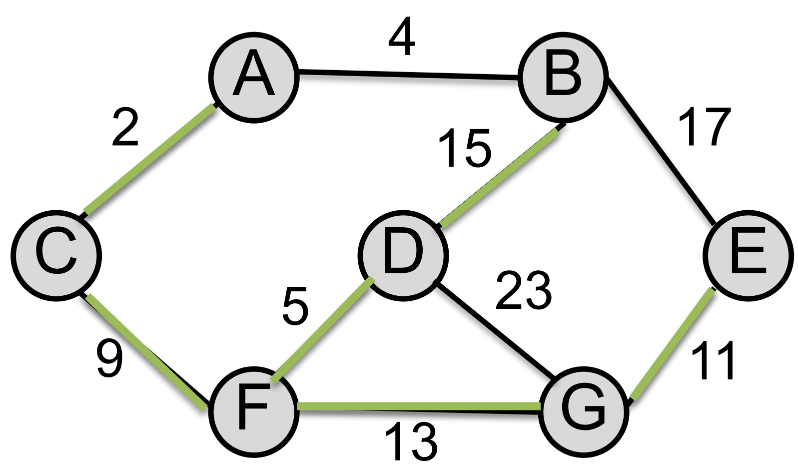
Dijkstra's Algorithm……

– Step 3:
  - Label of G has been re-labelled 18 because minimum(5+13, 23) = 18, E = 32 (15+17)
  - Therefore, N = {D, F, C, B, A, G}



– Step 3:
  - Label of E has been re-labelled 29 because minimum(5+13+11,15+17)
  - Therefore, N = {D, F, C, B, A, G, E}





Dijkstra's & MST Algorithms

- Minimum spanning tree algorithm is used to traverse a graph in the most efficient manner, but Dijkstra's algorithm calculates the distance from a given vertex to every other vertex in the graph

- Dijkstra's algorithm is very similar to Prim's algorithm
  - Both the algorithms begin at a specific node and extend outward within the graph, until all other nodes in the graph have been reached
  - The difference is while Prim's algorithm stores a minimum cost edge, Dijkstra's algorithm stores the total cost from a source node to the current node

Questions?

kychen@mail.ntust.edu.tw