Chapter 11: Graph Algorithms Part 2

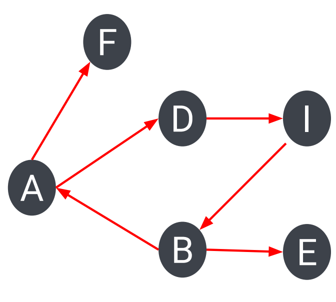
Dr. Sirasit Lochanachit

Outline

Graph Algorithms

- Minimum Spanning Tree Algorithms
  - Prim-Jarnik Algorithm
  - Kruskal's Algorithm

Minimum Spanning Tree



- A **spanning tree** is a tree that contains all of the nodes in a graph *G*.
- A **minimum spanning tree (MST)** is a spanning tree that has the smallest total weight of the edges.
- Given an undirected, weight graph *G*, we are interested in finding a tree *T* that contains all the nodes in *G* and minimises the sum
  - $w(T) = \sum_{(u, v) \text{ in } T} w(u, v).$

Minimum Spanning Tree

- Two classics **greedy** algorithms for solving the MST problem.
- Greedy method is based on choosing objects to join a growing collection by iteratively picking an object that minimises some cost function.
- Prim-Jarnik algorithm which is similar to Dijkstra's shortest path algorithm.
- Kruskal's algorithm, using the idea of clusters.

Prim’s Algorithm

Informal steps of Prim’s Algorithm:

1. Initialise a tree with a single node, chosen randomly from the graph.
2. Grow the tree by considering edges that connect the tree to nodes not yet in the tree.
3. Amongst considered edges, find the minimum-weight edge, and transfer it to the tree.
4. Repeat step 2 (until all nodes are in the tree).

Prim’s Algorithm

Pseudocode:

1. Associate with each node v of the graph a number *C*[*v*] and an edge *E*[*v*].
   a. *C*[*v*] is the cheapest cost of a connection to node v
   b. *E*[*v*] is the edge that provides the cheapest connection.
   c. Initialise all *C*[*v*] to Inf or very large number, except starting node.
   d. Initialise all *E*[*v*] to None indicating no edge connecting to node *v*.
2. Initialise an empty tree *T* and a set of Queue *Q* as unvisited nodes.
   a. Initially, *Q* has all nodes.

Prim’s Algorithm

Pseudocode:

3. Repeat the following steps until Q is empty:

   3.1 Find and remove a node *v* from Q that has the smallest value of *C*[*v*].

   3.2 Add node *v* to T and add edge *E*[*v*] to T.

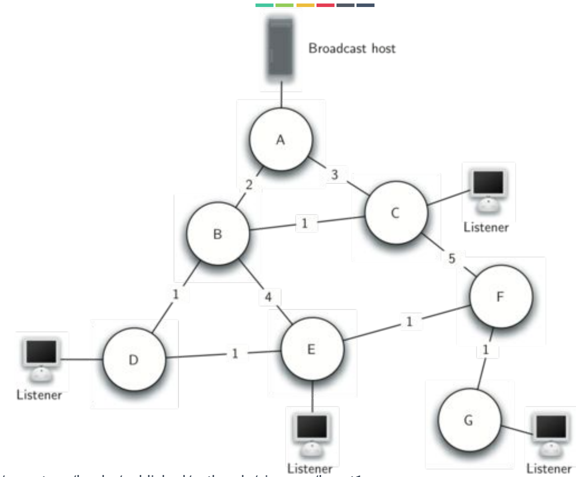
   3.3 Loop over the edges that connect node *v* to other nodes *w*.

   - For each edge, if node *w* belongs to *Q*, and edge (*v*, *w*) has smaller weight than *C*[*w*] (initially, Inf), perform the following steps:

     3.3.1 Set *C*[*w*] to the cost of edge (*v*, *w*).

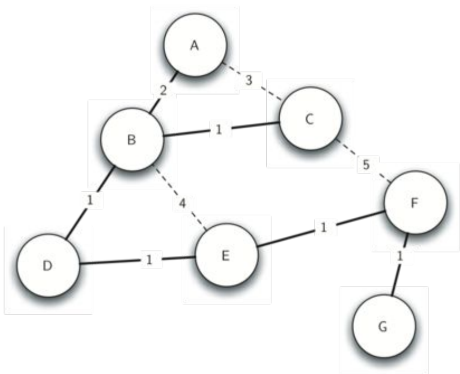
     3.3.2 Set *E*[*w*] to point to edge (*v*, *w*).

Broadcast Problem



Ref: https://runestone.academy/runestone/books/published/pythonds/_images/bcast1.png

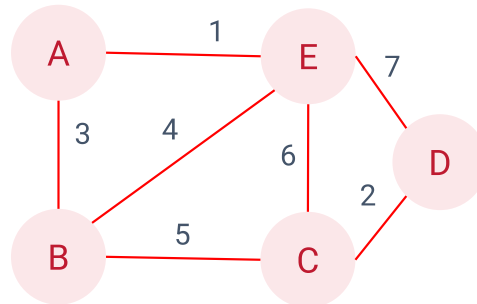
Prim's Algorithm



- A minimum spanning tree highlighting the edges that produce the smallest total weight.
- The broadcast host simply sends a single message into the network.
- Each router forwards the message to any neighbour that is part of the spanning tree.
- No router sees more than one copy of any message.

Ref: https://runestone.academy/runestone/books/published/pythonds/_images/mst1.png

Prim's Algorithm



Step 1 & 2:

	A	B	C	D	E
C	0	Inf	Inf	Inf	Inf
E	-	-	-	-	-
Q	{A, B, C, D, E}				

T	

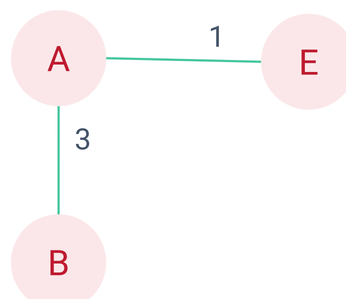
Prim's Algorithm

Step 3.1 & 3.2:

	A	B	C	D	E
C	0	Inf	Inf	Inf	Inf
E	-	-	-	-	-
Q	{B, C, D, E}				

T	(A, 0)

Prim's Algorithm



Step 3.3:

	A	B	C	D	E
C	0	3	Inf	Inf	1
E	E	-	-	-	-
Q	{B, C, D, E}				

T	(A, 0)

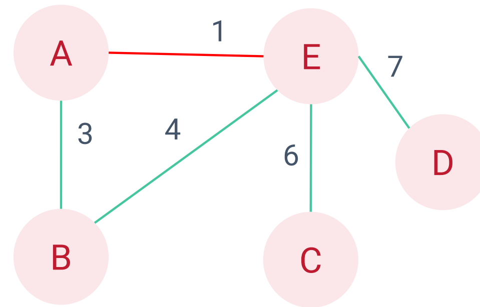
Prim's Algorithm

Step 3.1 & 3.2:

	A	B	C	D	E
C	0	3	Inf	Inf	1
E	E	-	-	-	-
Q	{B, C, D}				

T	(A, E, 1)

Prim's Algorithm

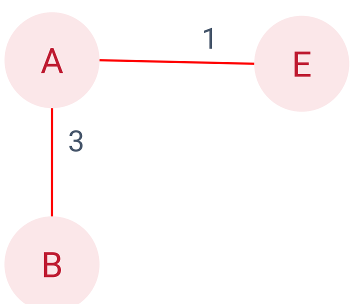


Step 3.3:

	A	B	C	D	E
C	0	3	6	7	1
E	E	-	-	-	B
Q	{B, C, D}				

T	(A, E, 1)

Prim's Algorithm

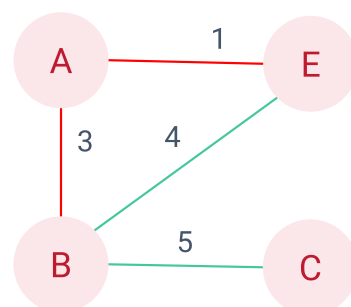


Step 3.1 & 3.2:

	A	B	C	D	E
C	0	3	6	7	1
E	B, E	-	-	-	-
Q	{C, D}				

T	(A, E, 1) (A, B, 3)

Prim's Algorithm

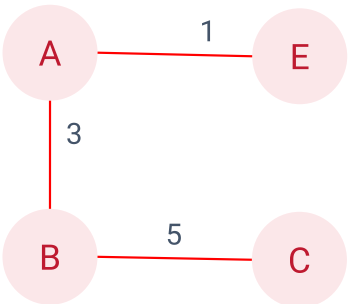


Step 3.3:

	A	B	C	D	E
C	0	3	5	7	1
E	B, E	C	-	-	-
Q	{C, D}				

T	(A, E, 1) (A, B, 3)

Prim's Algorithm

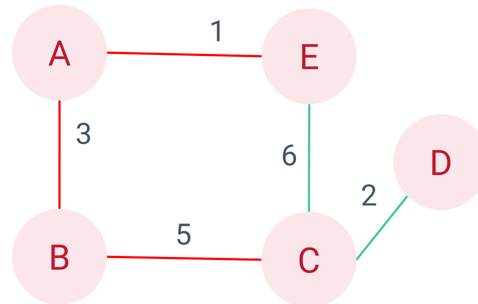


Step 3.1 & 3.2:

	A	B	C	D	E
C	0	3	5	7	1
E	B, E	C	-	-	-
Q	{D}				

T **(A, E, 1) (A, B, 3) (B, C, 5)**

Prim's Algorithm

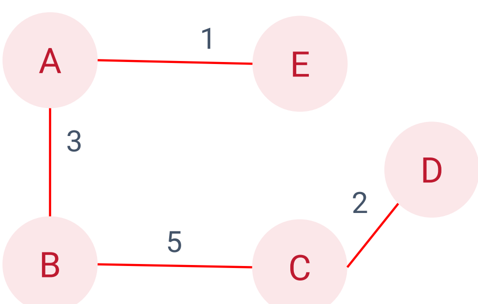


Step 3.3:

	A	B	C	D	E
C	0	3	5	2	1
E	B, E	C	D	-	-
Q	{D}				

T **(A, E, 1) (A, B, 3) (B, C, 5)**

Prim's Algorithm

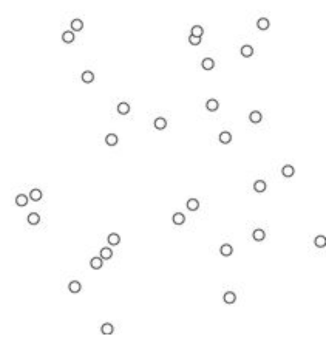


Step 3.1 & 3.2 & 3.3:

	A	B	C	D	E
C	0	3	5	2	1
E	B, E	C	D	-	-
Q	{}				

T **(A, E, 1) (A, B, 3) (B, C, 5) (C, D, 2)**

Kruskal's Algorithm



- Prim's algorithm builds the MST by growing a single tree until it spans the graph.
- **Kruskal's algorithm** maintains a **forest** of clusters, repeatedly merging pairs of clusters until a single cluster spans the graph.
- Initially, each node is a cluster.
- The algorithm then considers each edge ordered by increasing weight.
- If edge *e* connects 2 clusters, then 2 clusters

Kruskal's Algorithm

- Initially, each node is a cluster.
- The algorithm then considers each edge ordered by increasing weight.
- If edge *e* connects 2 different clusters, then 2 clusters are merged into a single cluster.
- On the other hand, if *e* connects 2 nodes in the same cluster, edge *e* is disregarded.
- The algorithm continues until there is only a single cluster which is the minimum spanning tree.

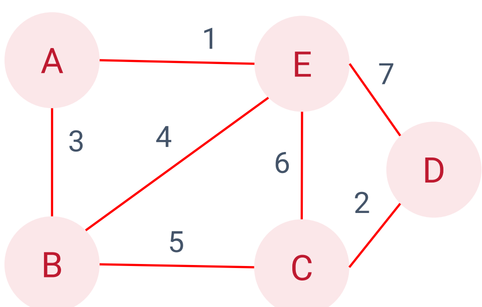
Kruskal's Algorithm

Informal steps of Kruskal's Algorithm:

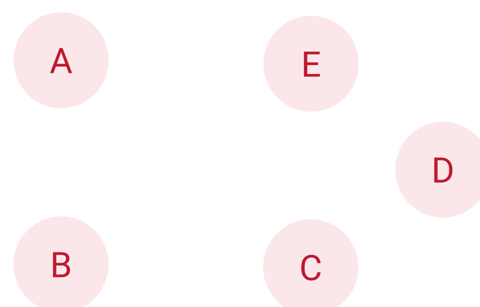
1. Create a forest *F* (a set of trees), where each node is a separate tree.
2. Create a set *S* containing all the edges in the graph.
3. While set *S* is not empty and *F* is not yet spanning (has all the nodes):
    a. Remove an edge with minimum weight from *S*.
    b. If the removed edge connects 2 different trees
        i. Add it to the forest *F*, combining two trees into a single tree.

Note: If the graph is connected, the forest has a single minimum spanning tree.

Kruskal's Algorithm



Kruskal's Algorithm

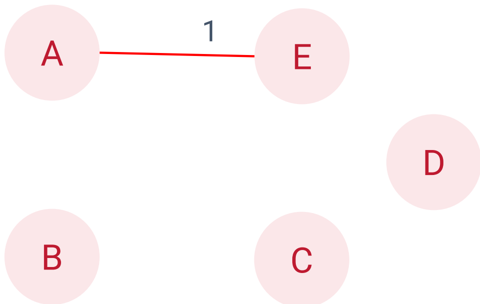


Step 1: Create a forest *F* (a set of trees), where each node is a separate tree.

Step 2: Create a set *S* containing all the edges in the graph.

S	**{(A, E, 1), (A, B, 3), (B, E, 4), (B, C, 5), (E, C, 6), (C, D, 2), (E, D, 7)}**

Kruskal's Algorithm



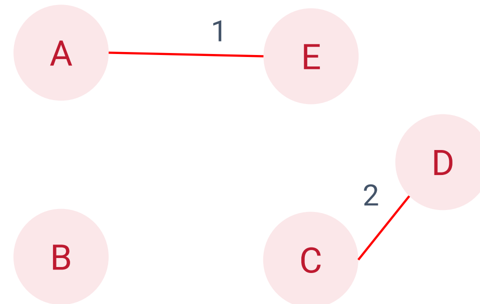
Step 3:

While set *S* is not empty and *F* is not yet spanning (has all the nodes):

1. Remove an edge with minimum weight from *S*.
2. If the removed edge connects 2 different trees
   a. Add it to the forest *F*, combining two trees into a single tree.

S | **{(A, B, 3), (B, E, 4), (B, C, 5), (E, C, 6), (C, D, 2), (E, D, 7)}**

Kruskal's Algorithm



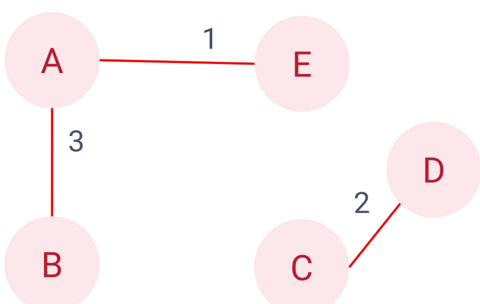
Step 3:

While set *S* is not empty and *F* is not yet spanning (has all the nodes):

1. Remove an edge with minimum weight from *S*.
2. If the removed edge connects 2 different trees
   a. Add it to the forest *F*, combining two trees into a single tree.

S | **{(A, B, 3), (B, E, 4), (B, C, 5), (E, C, 6), (E, D, 7)}**

Kruskal's Algorithm



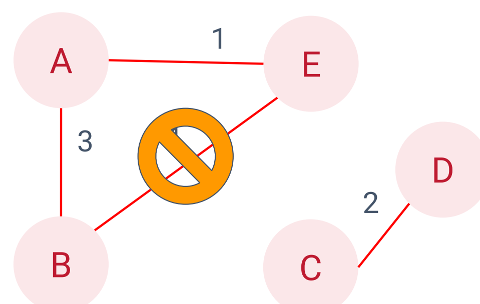
Step 3:

While set *S* is not empty and *F* is not yet spanning (has all the nodes):

1. Remove an edge with minimum weight from *S*.
2. If the removed edge connects 2 different trees
   a. Add it to the forest *F*, combining two trees into a single tree.

S | **{(B, E, 4), (B, C, 5), (E, C, 6), (E, D, 7)}**

Kruskal's Algorithm



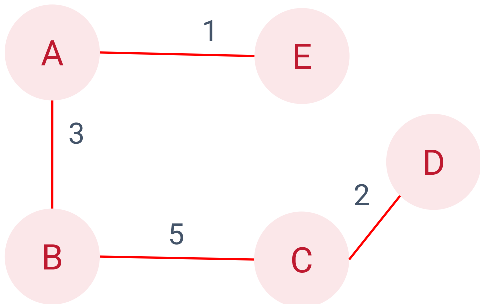
Step 3:

While set *S* is not empty and *F* is not yet spanning (has all the nodes):

1. Remove an edge with minimum weight from *S*.
2. If the removed edge connects 2 different trees
   a. Add it to the forest *F*, combining two trees into a single tree.

S | **{(B, C, 5), (E, C, 6), (E, D, 7)}**

Kruskal's Algorithm



Step 3:

While set *S* is not empty and *F* is not yet spanning (has all the nodes):

1. Remove an edge with minimum weight from *S*.
2. If the removed edge connects 2 different trees
   a. Add it to the forest *F*, combining two trees into a single tree.

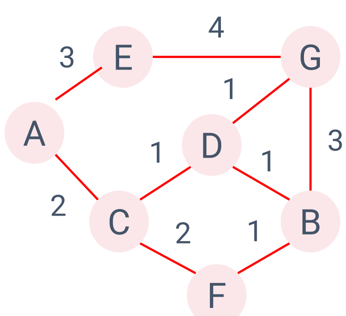
S	{(E, C, 6), (E, D, 7)}

Difference between MST and Shortest Path Algorithm

Although the computation of the minimum spanning tree (MST) and the shortest path (SP) algorithms looks similar, they focus on two different requirements as follows:

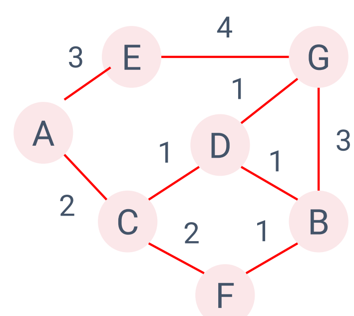
- MST's requirement is to select a set of edges so that there is a path between each node. Its goal is the sum of the edge lengths is minimised.
- SP's requirement is to reach destination node from source node with lowest possible cost (smallest weight).
- MST does not necessarily gives shortest path between two nodes.

MST Example 1

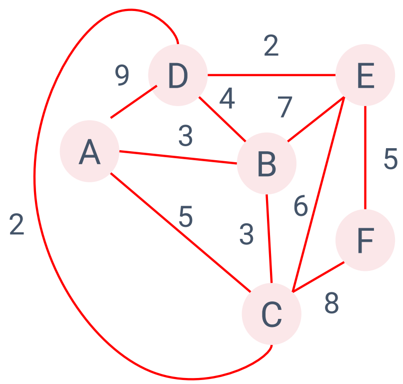


Shortest Path Example 1

Start from node A.

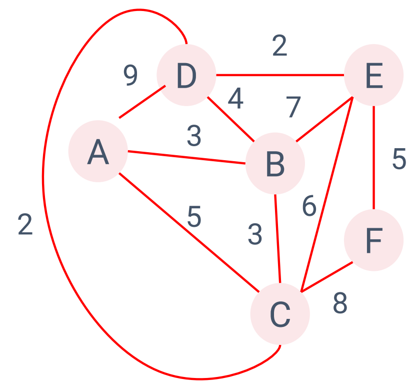


MST Example 2



Shortest Path Example 2

Start from node A.



Individual Assignment

- Assignment#9: Shortest Path and MST
- Due 09.00 am, Tuesday 10/11/2020.
- Submission
  - Email: sirasit@it.kmitl.ac.th
  - Paper: in classroom next week
- Can be either written by hand or typing.
- Make sure to submit on time!!
  - Late submission has penalty on the score.
- If unable to submit on time for reasonable reasons, let me know asap.