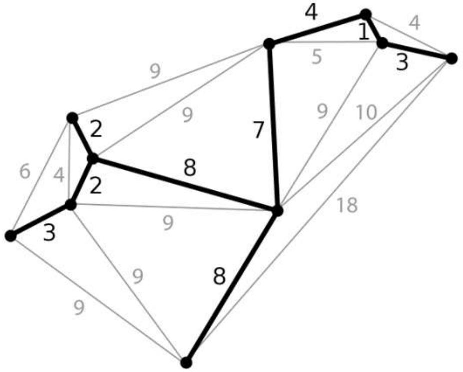
Algorithms Analysis and Design

Chapter 5

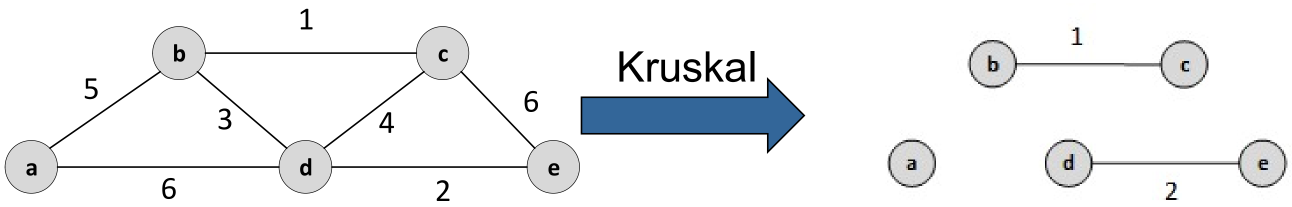
Greedy Technique
Part 2



Prim’s Algorithm

Prim’s algorithm

- Prim’s algorithm constructs a minimum spanning tree through a sequence of expanding subtrees.
- Unlike Kruskal’s <u>whose partial solutions are not necessarily connected</u>, a **partial solution in prim’s algorithm is a tree** (of course, trees are always connected).



A partial solution (disconnected)

Prim's algorithm

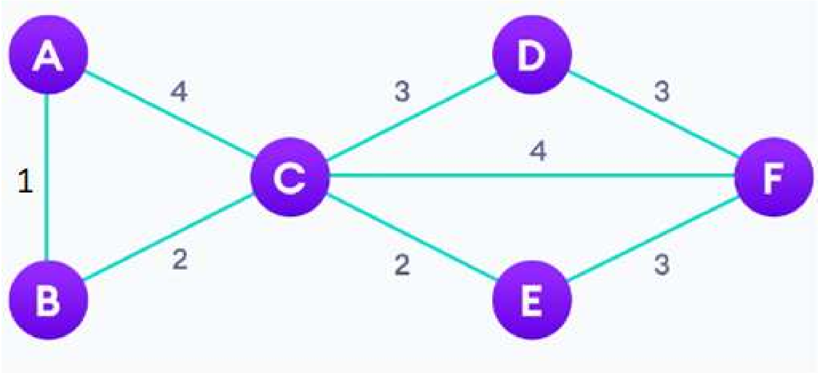
- Begin with a start vertex and no edges
  - Initial subtree consists of a single vertex selected arbitrarily from the set V of the graph's vertices.
- Apply the greedy rule at each iteration to expands the current tree
  - Add an edge of min weight that has one vertex in the current tree and the other not in the current tree
- Continue until you get a single tree T
  - The algorithm stops after all the graph's vertices have been included in the tree being constructed.

Since the algorithm expands a tree by exactly one vertex on each of its iterations, the total number of such iterations is $n - 1$, where n is the number of vertices in the graph

Application of Prim's algorithm

Example: Apply Kruskal's algorithm to find the MST of the following graph. Start with vertex **A**

vertex	known	cost	previous
A	0	**0**	-
B	0	∞	-
C	0	∞	-
D	0	∞	-
E	0	∞	-
F	0	∞	-

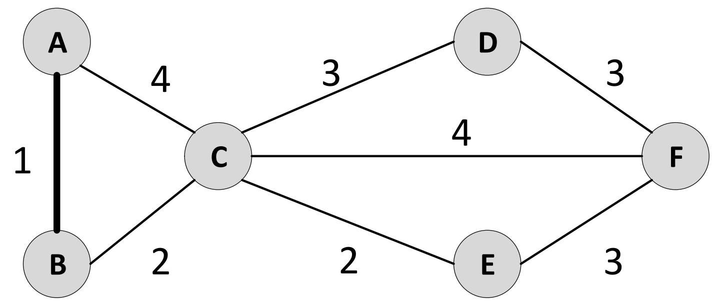


0: False
1: True

Initially, the cheapest unknown vertex is (**A**)

Application of Prim's algorithm

vertex	known	cost	previous	
A	1	0	-	
B	1	1	A	1<∞
C	0	4	-	4<∞
D	0	∞	-	
E	0	∞	-	
F	0	∞	-	



Update cost of neighbors of vertex A

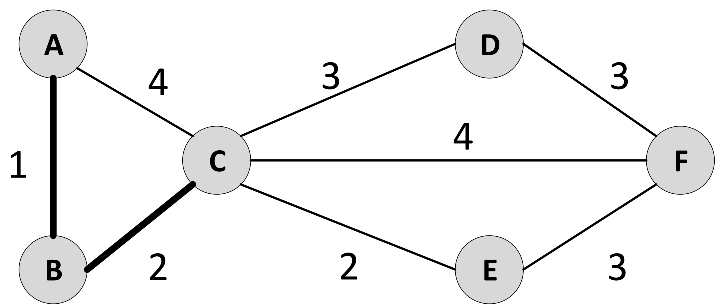
Find the cheapest unknown vertex (**B**)

Set known field of **B** to True

Application of Prim's algorithm

vertex	known	cost	previous
A	1	0	-
B	1	1	A
C	1	~~4~~ 2	B
D	0	∞	-
E	0	∞	-
F	0	∞	-

2<4



Update cost of neighbors of vertex B

Find cheapest unknown vertex (**C**)

Set known field of **c** to True

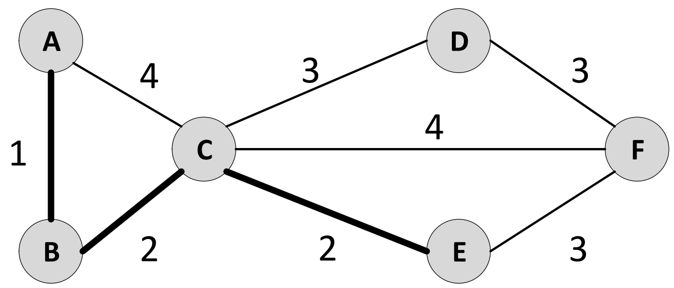
Application of Prim's algorithm

vertex	known	cost	previous
A	1	0	-
B	1	1	A
C	1	~~4~~ 2	B
D	0	3	-
E	1	2	C
F	0	4	-

3<∞

2<∞

4<∞



Update cost of neighbors of vertex C

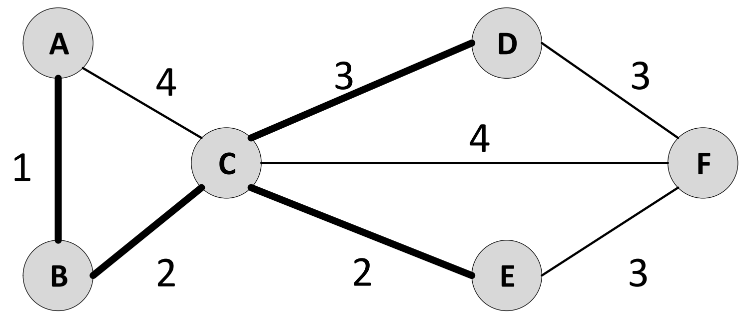
Find cheapest unknown vertex (**E**)

Set known field of **E** to True

Application of Prim's algorithm

vertex	known	cost	previous
A	1	0	-
B	1	1	A
C	1	~~4~~ 2	B
D	1	$\boxed{3}$	C
E	1	2	C
F	0	$\boxed{\cancel{4}\ 3}$	-

3<4



Update cost of neighbors of vertex E

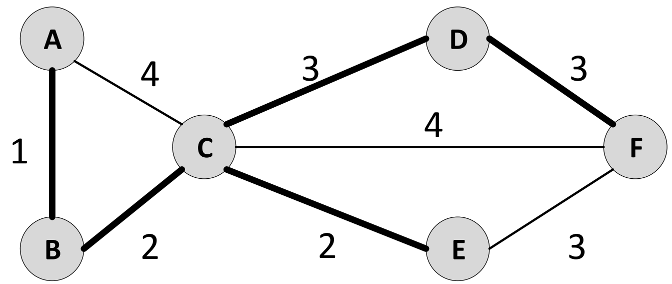
Find cheapest unknown vertex (**D**)

Set known field of **D** to True

Note: a tie can be resolved arbitrarily

Application of Prim's algorithm

vertix	known	cost	previou s
A	1	0	-
B	1	1	A
C	1	~~4~~ 2	B
D	1	3	C
E	1	2	C
F	1	~~4~~ 3	D

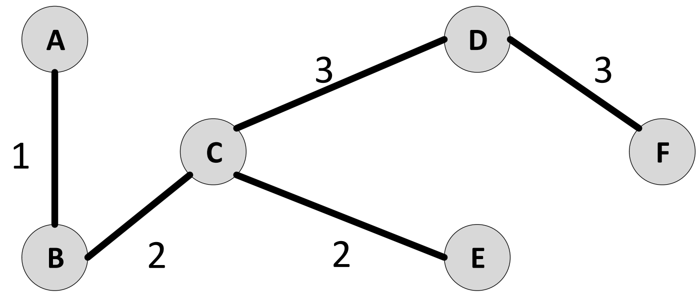


Update cost of neighbors of vertex D

Find cheapest unknown vertex (**F**)

Set known field of **F** to True

Stop: All vertices are known



MST cost = 11

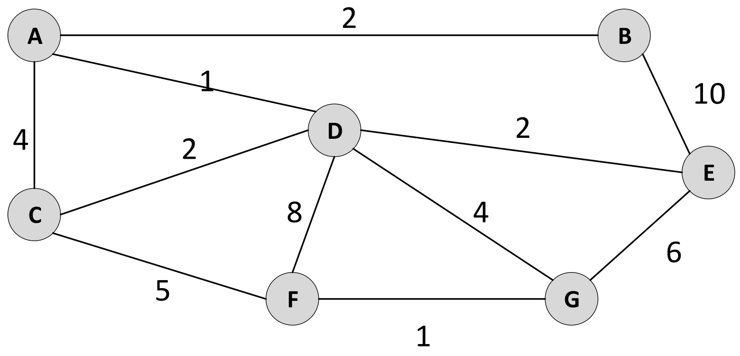
Application of Prim's algorithm

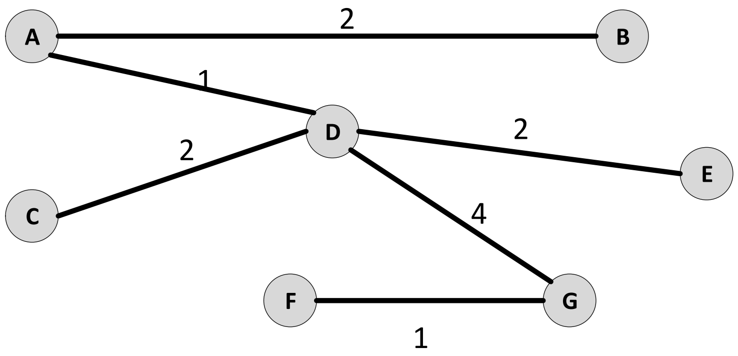
vertex	known	cost	previous
A	1	0	-
B	1	1	A
C	1	~~4~~ 2	B
D	1	3	C
E	1	2	C
F	1	~~4~~ 3	D

We can create the tree from the table as the cost and path are known

Application of Prim's algorithm

Example: Apply Prim's algorithm to the following graph:





MST cost = 12

Prim’s algorithm

- Implementation:
  - It necessary to provide each vertex not in the current tree with the **information about the shortest edge connecting the vertex to a tree vertex**.
  - We can provide such information by **attaching two labels to a vertex**: the <u>name of the nearest tree vertex</u> and the length (<u>the weight</u>) of the corresponding edge.
  - Vertices that are not adjacent to any of the tree vertices can be given the ∞ label indicating their “infinite” distance to the tree vertices and a null label for the name of the nearest tree vertex.
  - With such labels, finding the next vertex to be added to the current tree $T = \langle V_T, E_T \rangle$ becomes a simple task of **finding a vertex with the smallest distance label** in the set $V - VT$.

Pseudocode of Prim's algorithm

ALGORITHM *Prim*(G)

//Prim's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph $G = \langle V, E \rangle$
//Output: E_T, the set of edges composing a minimum spanning tree of G
$V_T \leftarrow \{v_0\}$ //the set of tree vertices can be initialized with any vertex
$E_T \leftarrow \varnothing$
for $i \leftarrow 1$ **to** $|V| - 1$ **do**
    find a minimum-weight edge $e^* = (v^*, u^*)$ among all the edges (v, u)
    such that v is in V_T and u is in $V - V_T$
    $V_T \leftarrow V_T \cup \{u^*\}$
    $E_T \leftarrow E_T \cup \{e^*\}$
return E_T

Time Complexity of Prim's algorithms

- How efficient is Prim's algorithm?
  - The answer **depends on the data structures chosen for the graph itself** and for **the priority queue of the set $\mathbf{V - V_T}$** whose vertex priorities are the distances to the nearest tree vertices

> If a graph is represented by its **adjacency lists** and the **priority queue is implemented as a min-heap**, the running time of the algorithm is in $O(|E| \log |V|)$.

Search

> A min-heap is a complete binary tree in which every element is less than or equal to its children. **Deletion** of the smallest element from and **insertion** of a new element into a min-heap of size n are $O(\log n)$

Animations of MST algorithms

For a better understanding of Kruskal's, Prim's algorithms and visualizing its operation through animation:

https://visualgo.net/en/mst?slide=1

https://www.cs.usfca.edu/~galles/visualization/Kruskal.html

https://www.cs.usfca.edu/~galles/visualization/Prim.html

Exercise

Indicate whether the following statements are true or false. **Justify your answer with examples**.

1. If edge weights of a connected weighted graph are all distinct, the graph must have exactly one minimum spanning tree.
2. If e is a minimum-weight edge in a connected weighted graph, it must be among edges of at least one minimum spanning tree of the graph.
3. If e is a minimum-weight edge in a connected weighted graph, it must be among edges of each minimum spanning tree of the graph.
4. If edge weights of a connected weighted graph are not all distinct, the graph must have more than one minimum spanning tree.