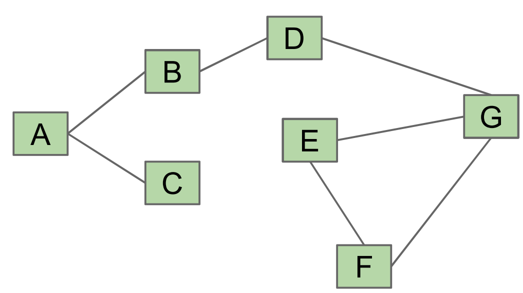
Minimum Spanning Tree (MST) in an undirected graph

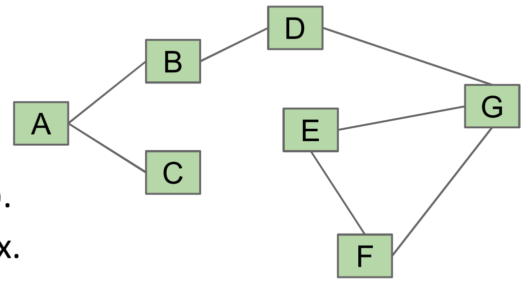
Warm-up Problem

Given a undirected graph, determine if it contains any cycles.



Warm-up Problem

Given a undirected graph, determine if it contains any cycles.

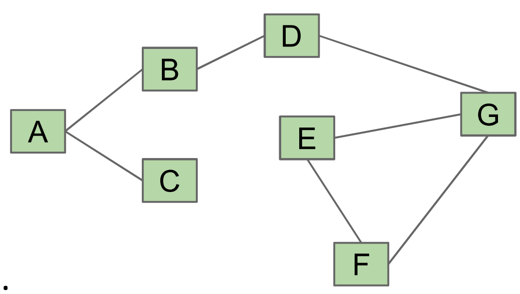


Approach 1: Do DFS from 0 (arbitrary vertex).

- Keep going until you see a marked vertex.
- Potential danger:
  - 1 looks back at 0 and sees marked.
  - Solution: Just don't count the node you came from. (father node)

Warm-up Problem

Given a undirected graph, determine if it contains any cycles.



Approach 2: Use an Union/Finding algorithm.

- For each edge, check if the two vertices are in the same union (set).
  - If not, union them.
  - If so, there is a cycle.
  - Union=[A-C]=>[A-C-B]=>[A-C-B-D]=> [A-C-B-D-G] => [A-C-B-D-G-E]=> [A-C-B-D-G-E-F]=> consider EF edge. Both E and F are in Union => there is a cycle

Spanning Trees

Given an **undirected** graph, a ***spanning tree*** T is a subgraph of G, where T:

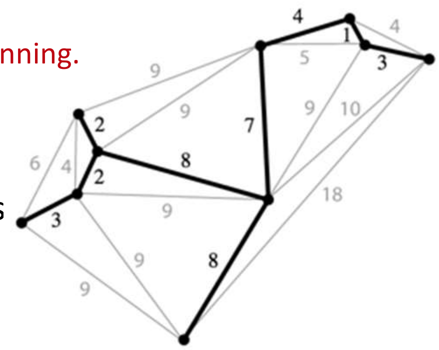
- Is connected.
- Is acyclic.
- Includes all of the vertices.

These two properties make it a tree.

This makes it spanning.

Example:

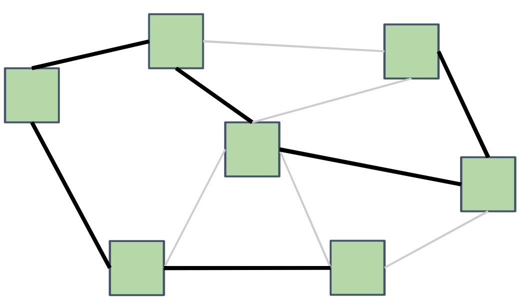
- Spanning tree is the black edges and vertices

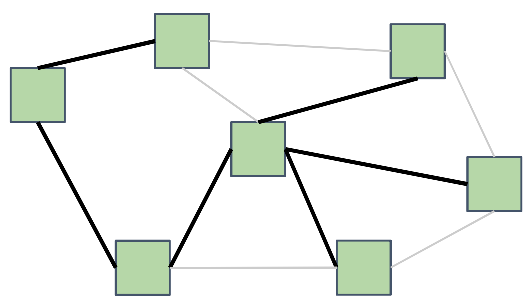


A ***minimum spanning tree*** is a spanning tree of minimum total weight.

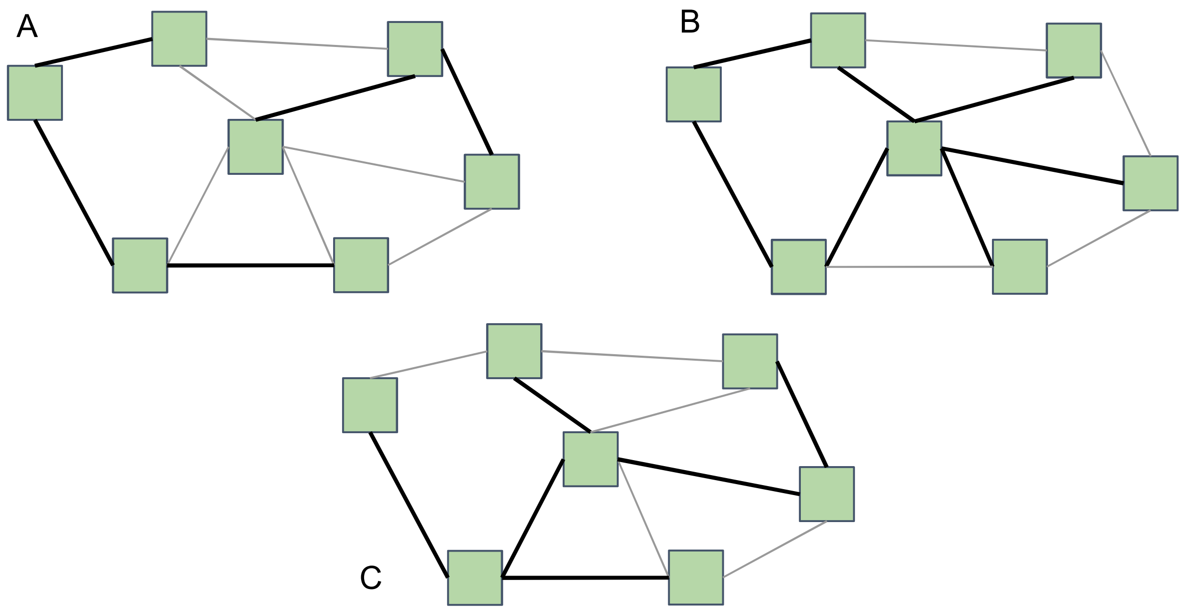
- Example: Network of power lines that connect a bunch of buildings.

Spanning Trees



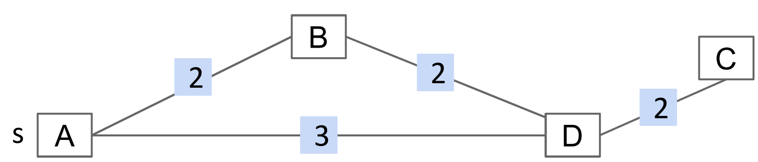


Which are Spanning Trees?



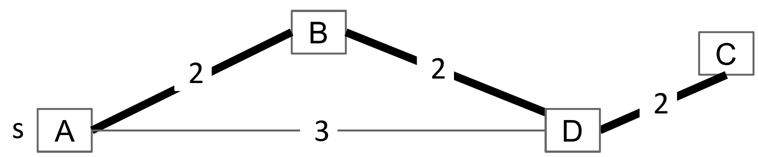
MST

Find the MST for the graph.



MST

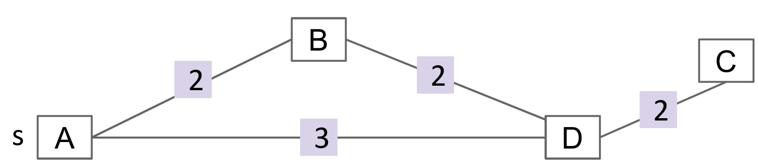
Find the MST for the graph. What is the number of edges of MST?



MST vs. SPT

Is the MST for this graph also a shortest paths tree? If so, using which node as the starting node for this SPT?

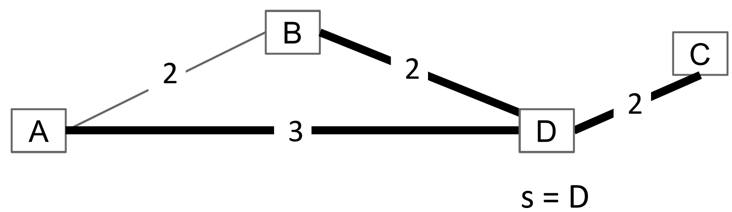
A. A
B. B
C. C
D. D
E. No SPT is an MST.



MST vs. SPT

Is the MST for this graph also a shortest paths tree? If so, using which node as the starting node for this SPT?

A. A
B. B
C. C
D. D
E. No SPT is an MST.

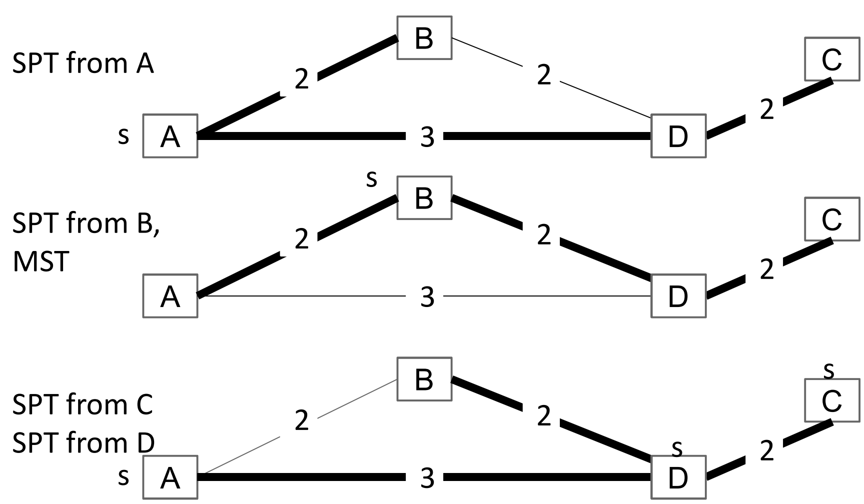


Doesn't work for s = D!

MST vs. SPT

Is the MST for this graph also a shortest paths tree? If so, using which node as the starting node for this SPT?

A. A
B. B
C. C
D. D
E. No SPT is an MST.



MST vs. SPT

A shortest paths tree depends on the start vertex:

- Because it tells you how to get from a source to EVERYTHING.

There is no source for a MST.

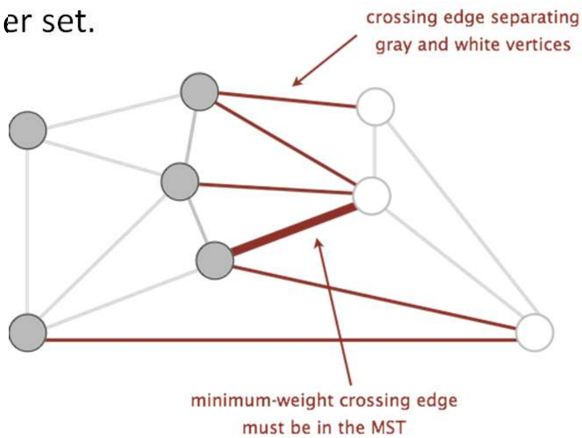
Nonetheless, the MST sometimes happens to be an SPT for a specific vertex.

For a weighted undirected graph, there is only 1 MST and the number of edges of a MST is V-1 if its edge weights are unique.

The Cut Property

A Useful Tool for Finding the MST: Cut Property

- A ***cut*** is an assignment of a graph's nodes to two non-empty sets.
- A ***crossing edge*** is an edge which connects a node from one set to a node from the other set.

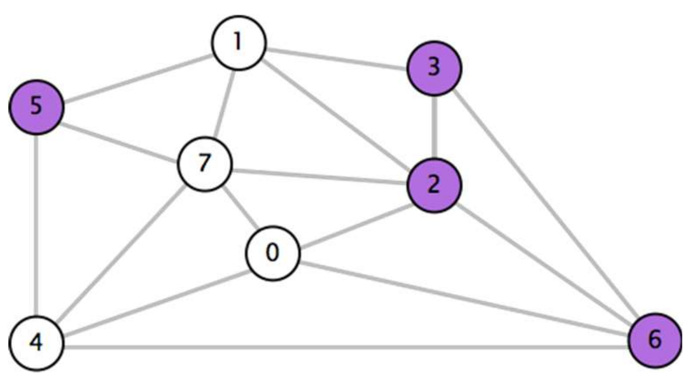


Cut property: Given any cut, minimum weight crossing edge is in the MST.

- For rest of today, we'll assume edge weights are unique.

Cut Property in Action

Which edge is the minimum weight edge crossing the cut {2, 3, 5, 6}?

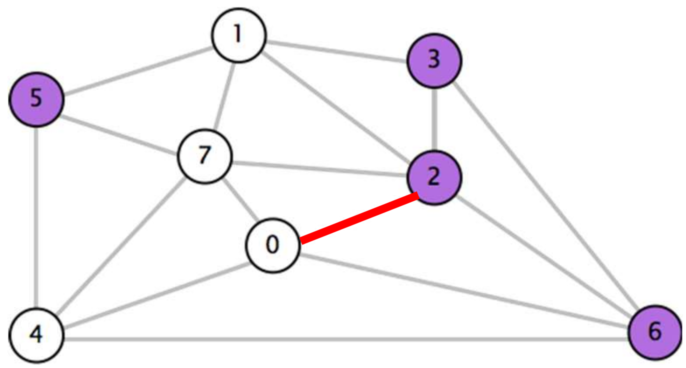


0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Cut Property in Action

Which edge is the minimum weight edge crossing the cut {2, 3, 5, 6}?

- 0-2. Must be part of the MST!

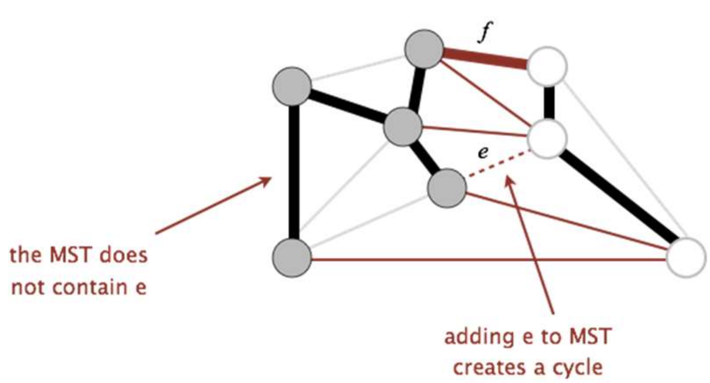


0-7	0.16
2-3	0.17
1-7	0.19
0-2	**0.26**
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Cut Property Proof

Suppose that the minimum crossing edge *e* were not in the MST.

- Adding *e* to the MST creates a cycle.
- Some other edge *f* must also be a crossing edge.
- Removing f and adding e is a lower weight spanning tree.
- Contradiction!



Generic MST Finding Algorithm

Start with no edges in the MST.

- Find a cut that has no crossing edges in the MST.
- Add smallest crossing edge to the MST.
- Repeat until MST has V-1 edges.

This should work, but we need some way of finding a cut with no crossing edges!

- Random isn't a very good idea.

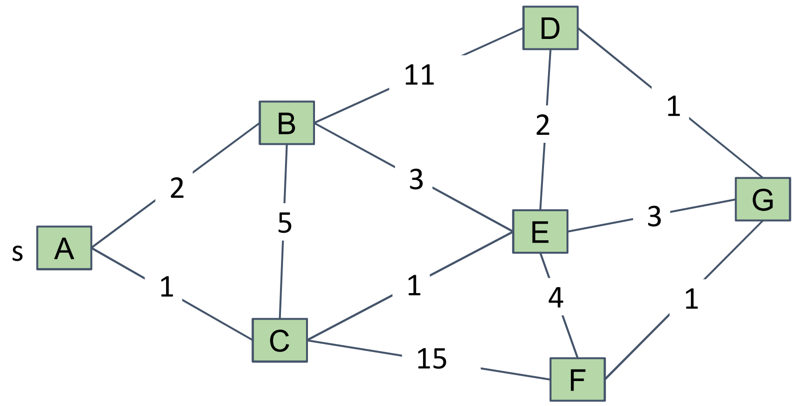
Basic Prim's (Demo)

Prim's Demo (Conceptual)

Start from some arbitrary start node.

- Add shortest edge (mark black) that has one node inside the MST under construction. Repeat until V-1 edges.

Node	edgeTo
A	-
B	-
C	-
D	-
E	-
F	-
G	-

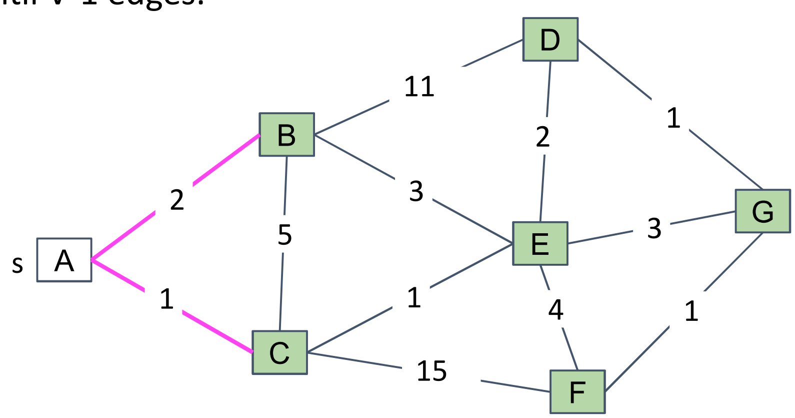


Prim's Demo (Conceptual)

Start from some arbitrary start node.

- Add shortest edge (mark black) that has one node inside the MST under construction. Repeat until V-1 edges.

Node	edgeTo
A	-
B	-
C	-
D	-
E	-
F	-
G	-

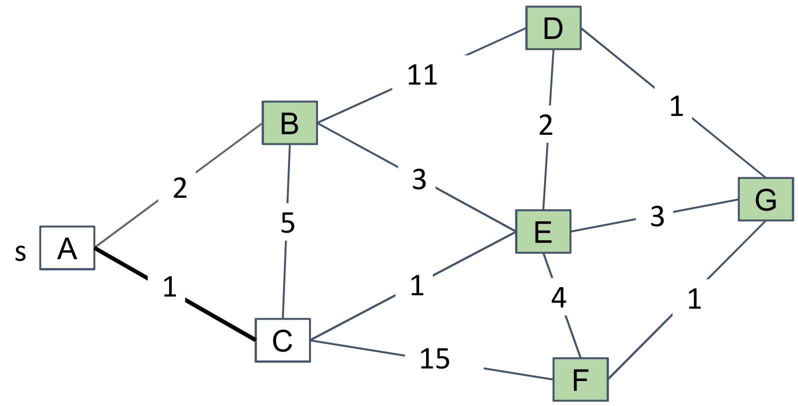


Prim's Demo (Conceptual)

Start from some arbitrary start node.

- Add shortest edge (mark black) that has one node inside the MST under construction. Repeat until V-1 edges.

Node	edgeTo
A	-
B	-
C	A
D	-
E	-
F	-
G	-

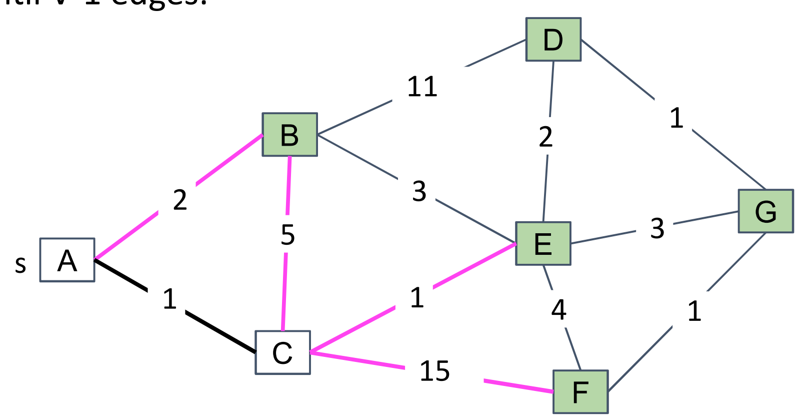


Prim's Demo (Conceptual)

Start from some arbitrary start node.

- Add shortest edge (mark black) that has one node inside the MST under construction. Repeat until V-1 edges.

Node	edgeTo
A	-
B	-
C	A
D	-
E	-
F	-
G	-

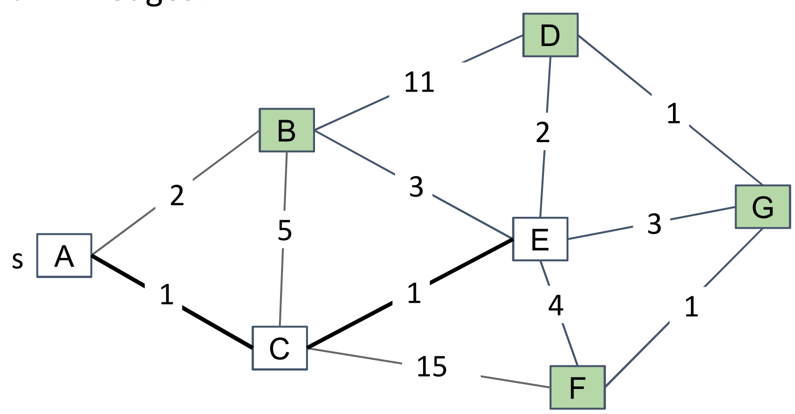


Prim's Demo (Conceptual)

Start from some arbitrary start node.

- Add shortest edge (mark black) that has one node inside the MST under construction. Repeat until V-1 edges.

Node	edgeTo
A	-
B	-
C	A
D	-
E	C
F	-
G	-

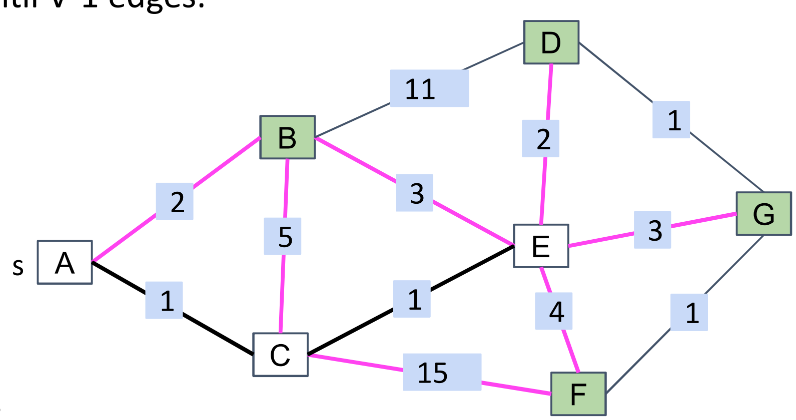


Prim's Demo (Conceptual)

Start from some arbitrary start node.

- Add shortest edge (mark black) that has one node inside the MST under construction. Repeat until V-1 edges.

Node	edgeTo
A	-
B	-
C	A
D	-
E	C
F	-
G	-



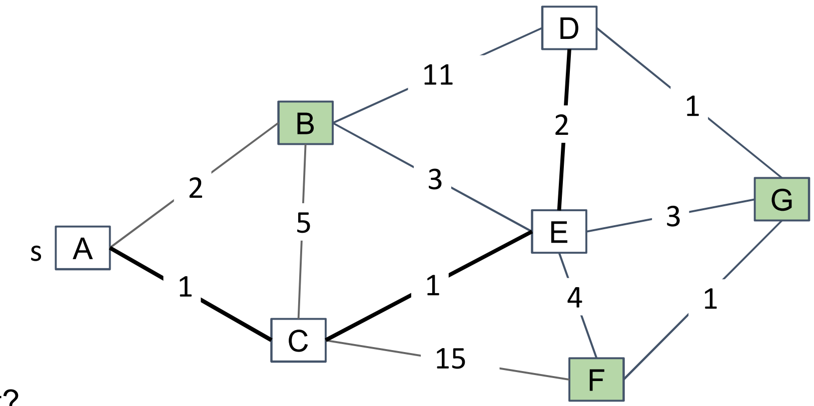
Which edge is added next?

Prim's Demo (Conceptual)

Start from some arbitrary start node.

- Add shortest edge (mark black) that has one node inside the MST under construction. Repeat until V-1 edges.

Node	edgeTo
A	-
B	-
C	A
D	E
E	C
F	-
G	-



Which edge is added next?

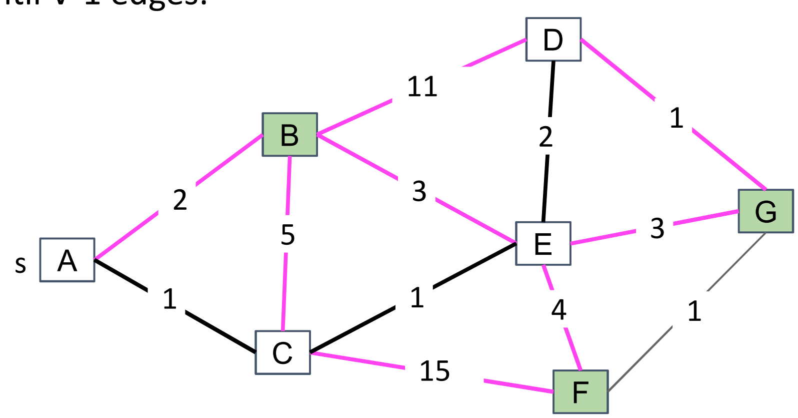
- Either A-B or D-E are guaranteed to work (see exercises for proof)!
- Note: They are not both guaranteed to be in the MST.

Prim's Demo (Conceptual)

Start from some arbitrary start node.

- Add shortest edge (mark black) that has one node inside the MST under construction. Repeat until V-1 edges.

Node	edgeTo
A	-
B	-
C	A
D	E
E	C
F	-
G	-

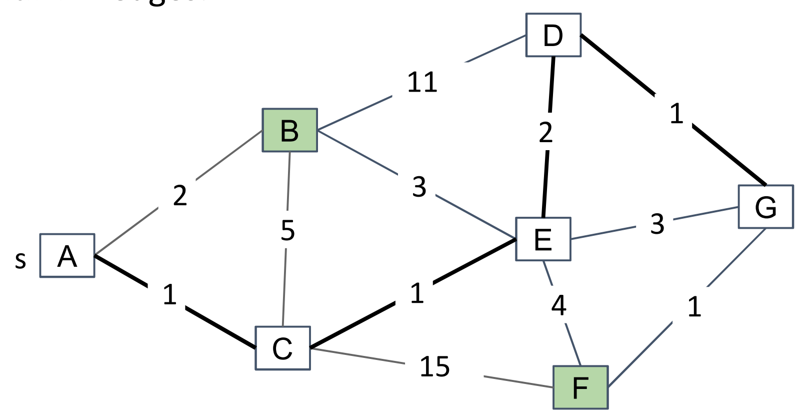


Prim's Demo (Conceptual)

Start from some arbitrary start node.

- Add shortest edge (mark black) that has one node inside the MST under construction. Repeat until V-1 edges.

Node	edgeTo
A	-
B	-
C	A
D	E
E	C
F	-
G	D

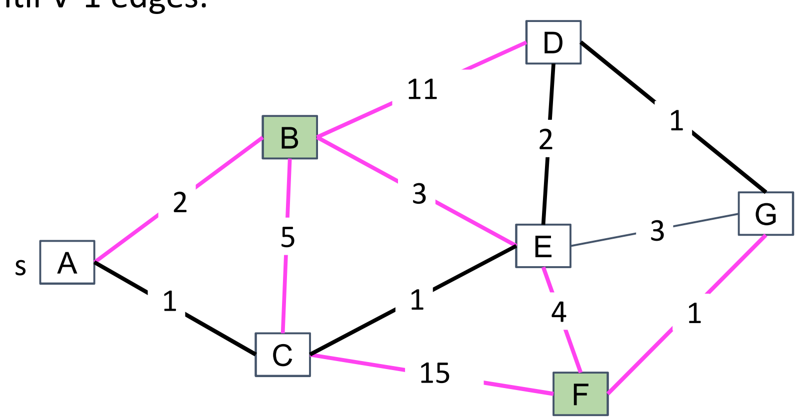


Prim's Demo (Conceptual)

Start from some arbitrary start node.

- Add shortest edge (mark black) that has one node inside the MST under construction. Repeat until V-1 edges.

Node	edgeTo
A	-
B	-
C	A
D	E
E	C
F	-
G	D

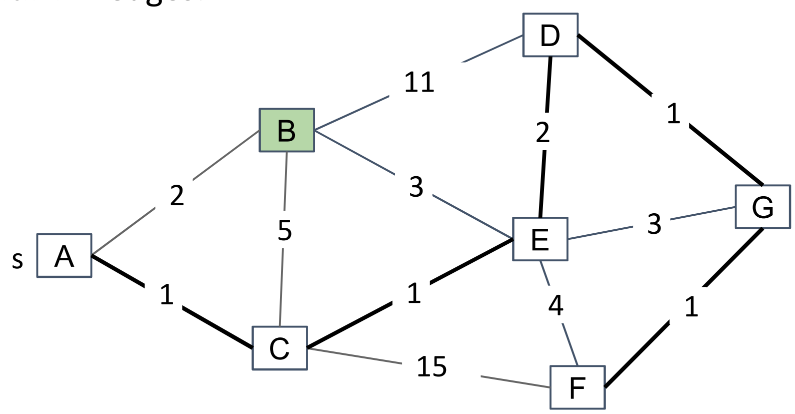


Prim’s Demo (Conceptual)

Start from some arbitrary start node.

- Add shortest edge (mark black) that has one node inside the MST under construction. Repeat until V-1 edges.

Node	edgeTo
A	-
B	-
C	A
D	E
E	C
F	G
G	D

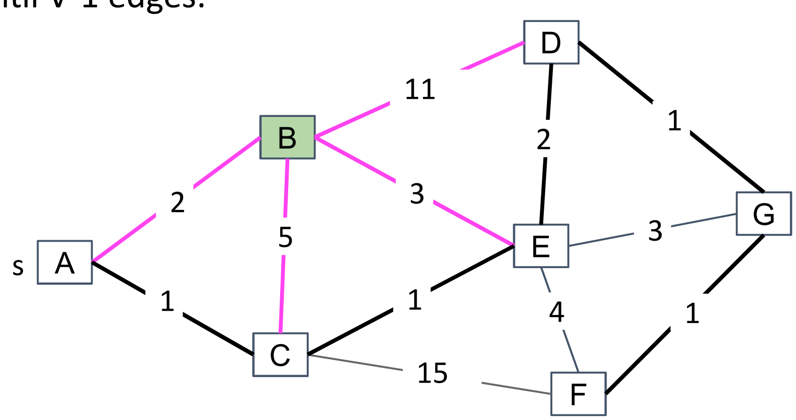


Prim’s Demo (Conceptual)

Start from some arbitrary start node.

- Add shortest edge (mark black) that has one node inside the MST under construction. Repeat until V-1 edges.

Node	edgeTo
A	-
B	-
C	A
D	E
E	C
F	G
G	D

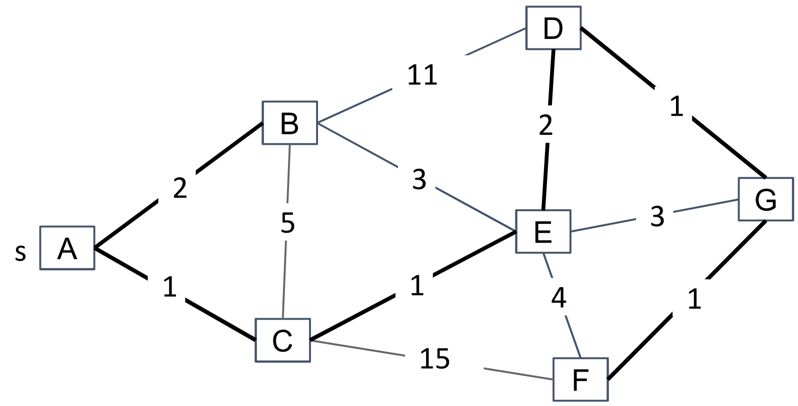


Prim's Demo (Conceptual)

Start from some arbitrary start node.

- Add shortest edge (mark black) that has one node inside the MST under construction. Repeat until V-1 edges.

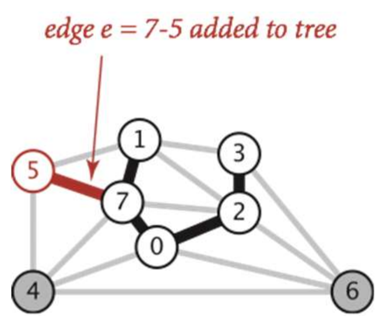
Node	edgeTo
A	-
B	A
C	A
D	E
E	C
F	G
G	D



Prim's Algorithm

Start from some arbitrary start node.

- Repeatedly add shortest edge (mark black) that has one node inside the MST under construction.
- Repeat until V-1 edges.



Optimized Prim's (Demo)

Prim's Algorithm Implementation

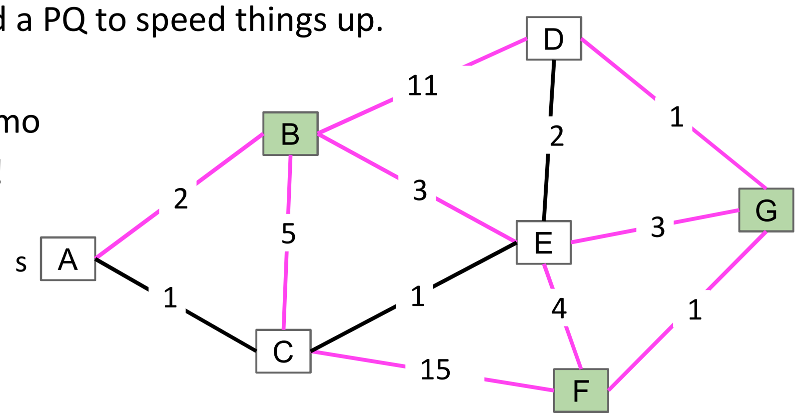
The natural implementation of the conceptual version of Prim's algorithm is highly inefficient.

- Example: Iterating over all purple edges shown is unnecessary and slow.

Can use some cleverness and a PQ to speed things up.

Realistic Implementation Demo

- Very similar to Dijkstra's!

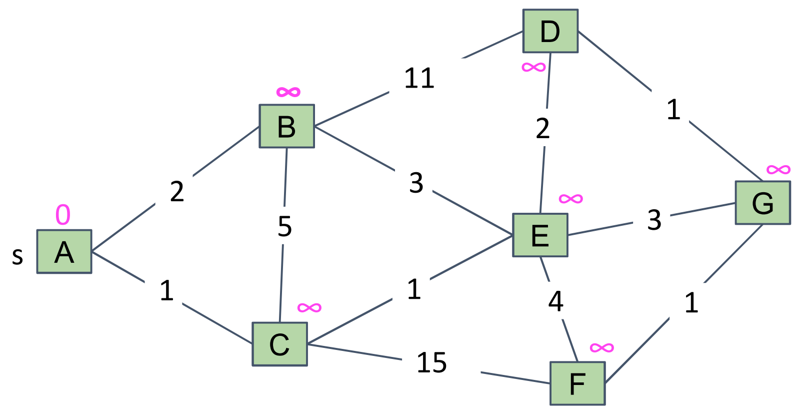


Prim's Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.

Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A	0	-
B	∞	-
C	∞	-
D	∞	-
E	∞	-
F	∞	-
G	∞	-



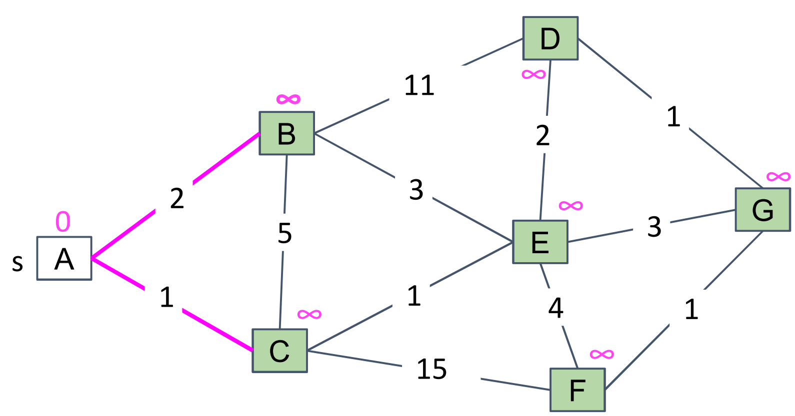

```
PQ: [(A: 0), (B: ∞), (C: ∞), (D: ∞), (E: ∞), (F: ∞), (G: ∞)]
```

Prim's Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.

Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A	0	-
B	∞	-
C	∞	-
D	∞	-
E	∞	-
F	∞	-
G	∞	-



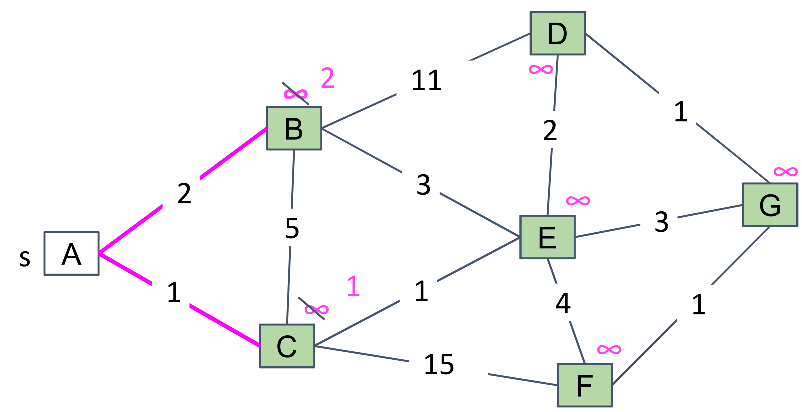

```
PQ: [(B: ∞), (C: ∞), (D: ∞), (E: ∞), (F: ∞), (G: ∞)]
```

Prim's Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.

Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A		-
B	2	A
C	1	A
D	∞	-
E	∞	-
F	∞	-
G	∞	-



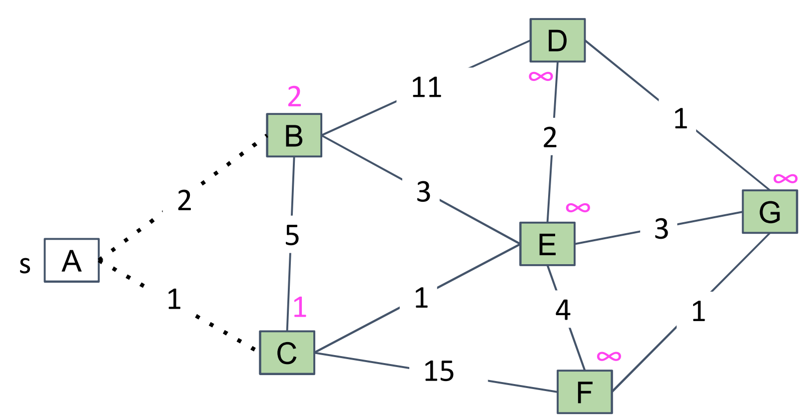
PQ: [(C: 1), (B: 2), (D: ∞), (E: ∞), (F: ∞), (G: ∞)]

Prim's Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.

Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A	0	-
B	2	A
C	1	A
D	∞	-
E	∞	-
F	∞	-
G	∞	-

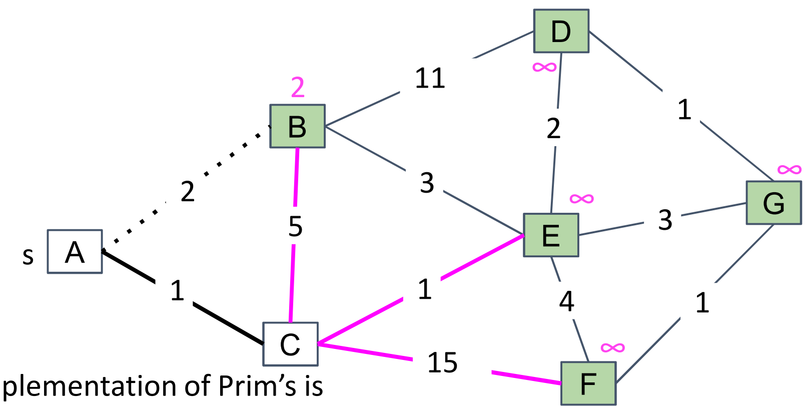


PQ: [(C: 1), (B: 2), (D: ∞), (E: ∞), (F: ∞), (G: ∞)]

Prim's Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.
Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A		-
B	2	A
C		**A**
D	∞	-
E	∞	-
F	∞	-
G	∞	-



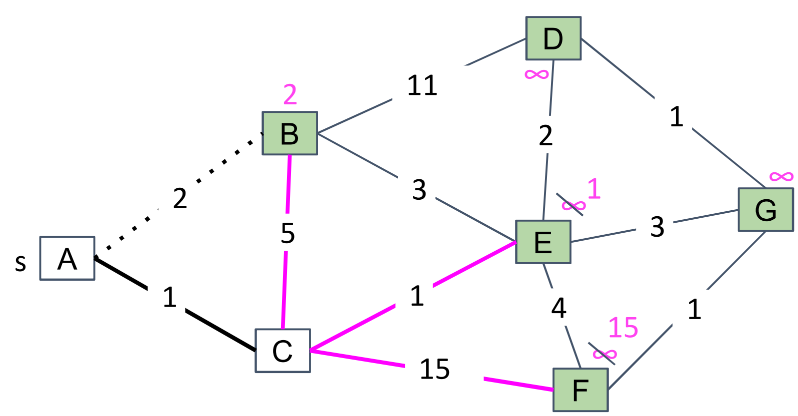
Note: Vertex removal in this implementation of Prim's is equivalent to edge addition in the conceptual version of Prim's.

`PQ: [(B: 2), (D: ∞), (E: ∞), (F: ∞), (G: ∞)]`

Prim's Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.
Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A		-
B	2	A
C		**A**
D	∞	-
E	1	C
F	15	C
G	∞	-



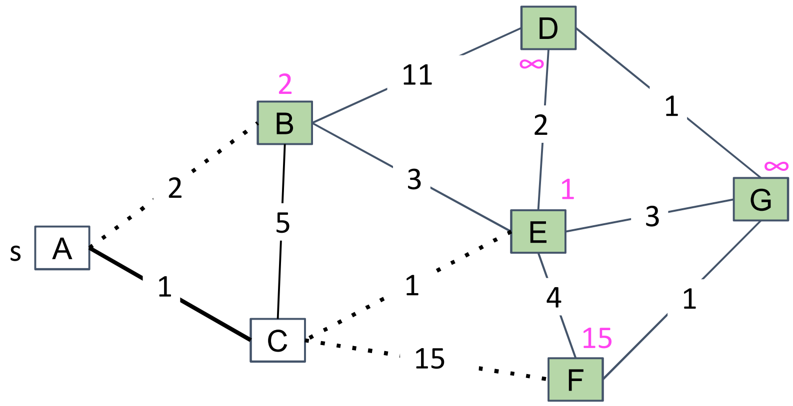
`PQ: [(E: 1), (B: 2), (F: 15), (D: ∞), (G: ∞)]`

Prim's Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.

Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A		-
B	2	A
C		**A**
D	∞	-
E	1	C
F	15	C
G	∞	-



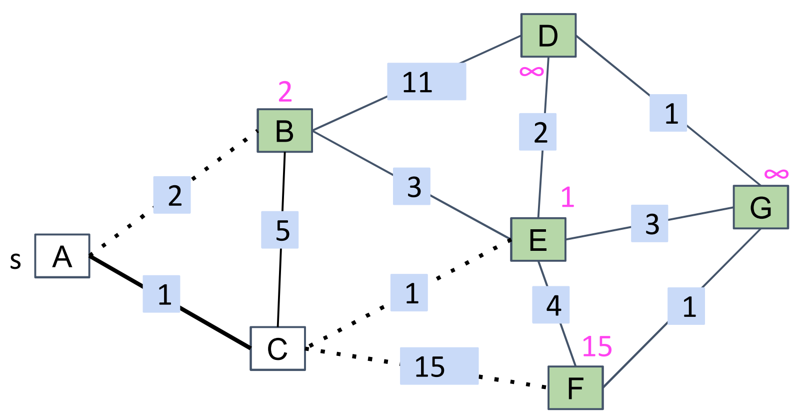
`PQ: [(E: 1), (B: 2), (F: 15), (D: ∞), (G: ∞)]`

Prim's Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.

Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A		-
B	2	A
C		**A**
D	∞	-
E	1	C
F	15	C
G	∞	-



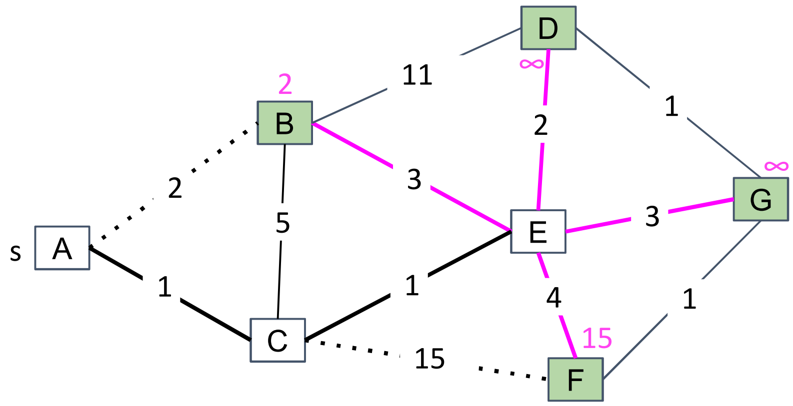
`PQ: [(E: 1), (B: 2), (F: 15), (D: ∞), (G: ∞)]`

Prim's Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.

Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A		-
B	2	A
C		**A**
D	∞	-
E		**C**
F	15	C
G	∞	-



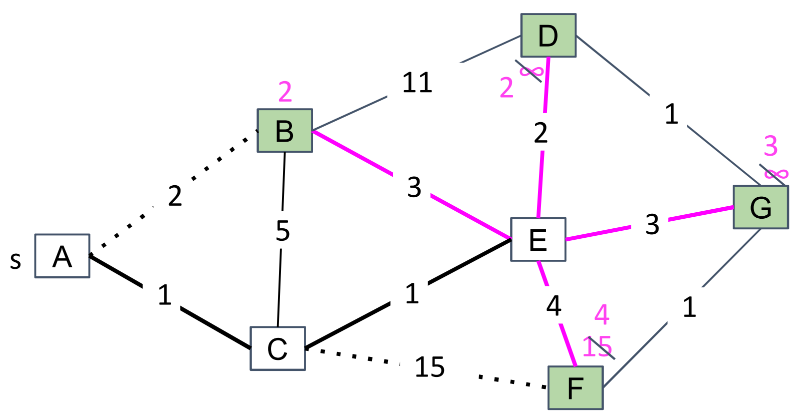
PQ: [(B: 2), (F: 15), (D: ∞), (G: ∞)]

Prim's Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.

Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A		-
B	2	A
C		**A**
D	2	E
E		**C**
F	4	E
G	3	E

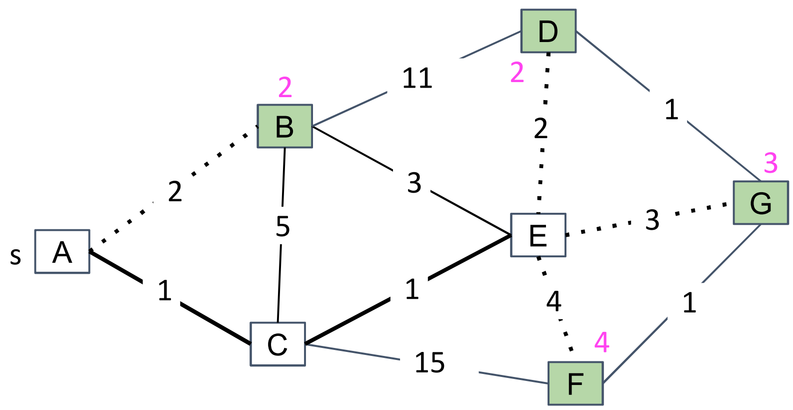


PQ: [(B: 2), $\boxed{\text{(D: 2), (G: 3), (F: 4)}}$]

Prim's Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.
Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A		-
B	2	A
C		**A**
D	2	E
E		**C**
F	4	E
G	3	E

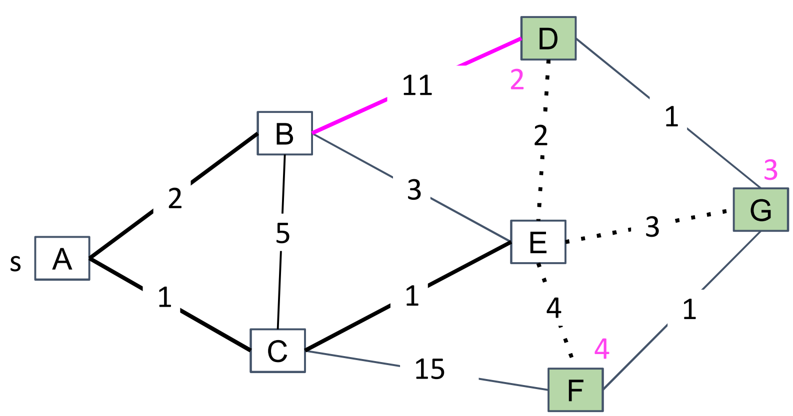


PQ: [(B: 2), (D: 2), (G: 3), (F: 4)]

Prim's Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.
Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A		-
B		**A**
C		**A**
D	2	E
E		**C**
F	4	E
G	3	E



PQ: [(D: 2), (G: 3), (F: 4)]

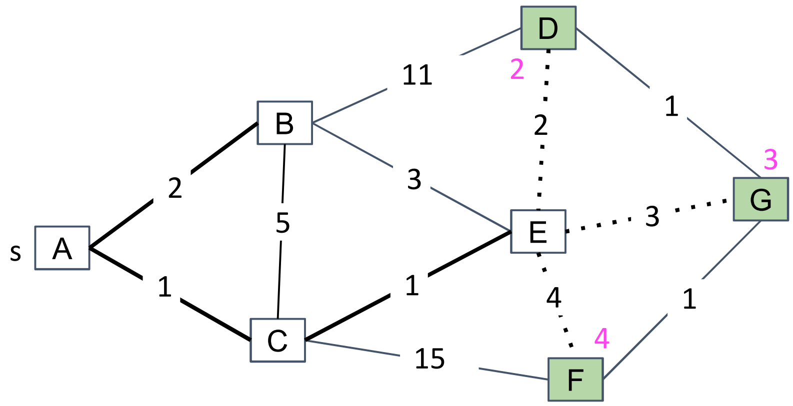
No need to consider edges
with weight 5 and 3 since
other side is already marked!

Prim's Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.

Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A		-
B		**A**
C		**A**
D	2	E
E		**C**
F	4	E
G	3	E



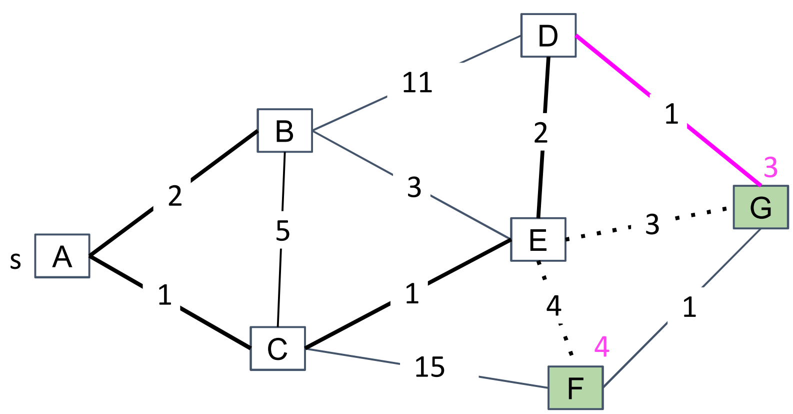
PQ: [(D: 2), (G: 3), (F: 4)]

Prim's Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.

Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A		-
B		**A**
C		**A**
D		**E**
E		**C**
F	4	E
G	3	E



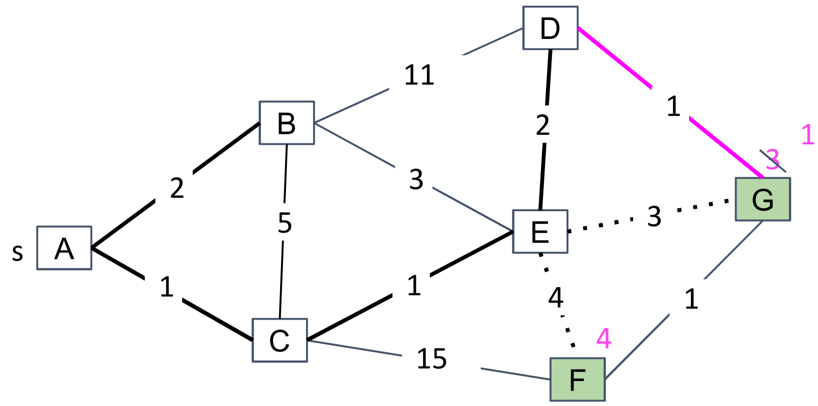
PQ: [(G: 3), (F: 4)]

Prim's Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.

Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A		-
B		**A**
C		**A**
D		**E**
E		**C**
F	4	E
G	1	D



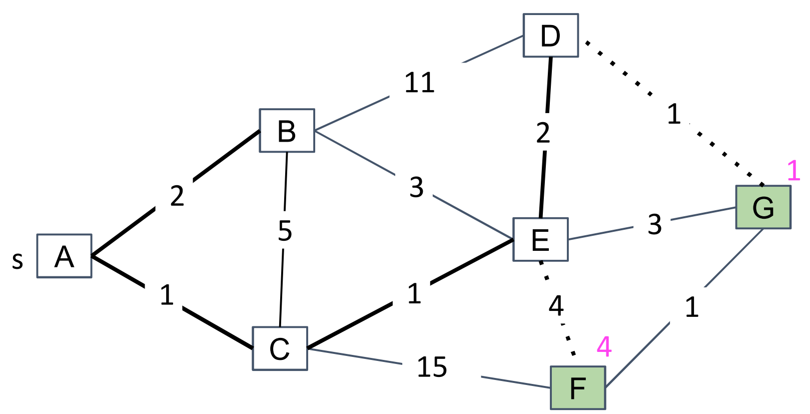
PQ: [$\boxed{\text{(G: 1)}}$, (F: 4)]

Prim's Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.

Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A		-
B		**A**
C		**A**
D		**E**
E		**C**
F	4	E
G	1	D



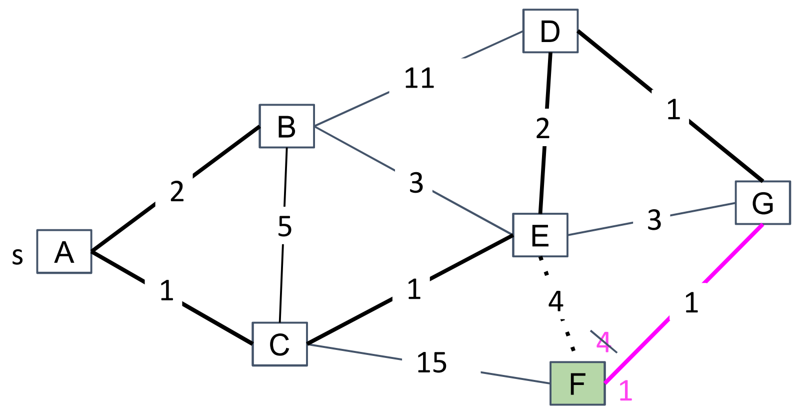
PQ: [(G: 1), (F: 4)]

Prim’s Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.

Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A		-
B		**A**
C		**A**
D		**E**
E		**C**
F	1	G
G		**D**



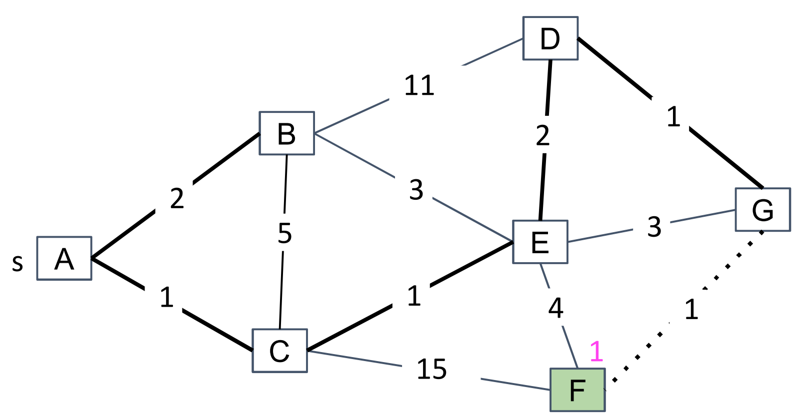
PQ: [(F: 1)]

Prim’s Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.

Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A		-
B		**A**
C		**A**
D		**E**
E		**C**
F	1	G
G		**D**



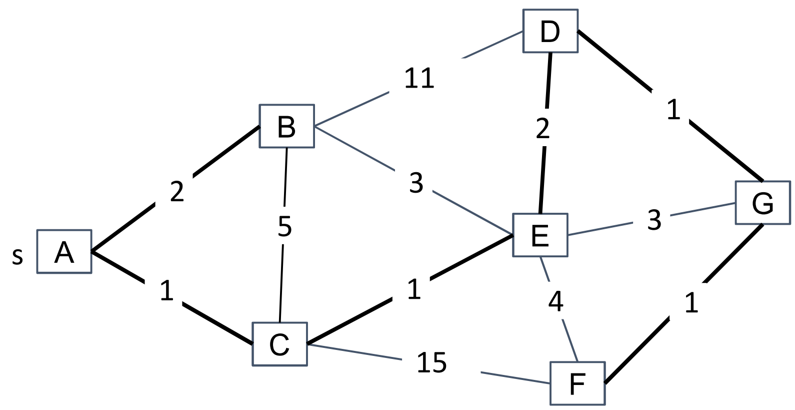
PQ: [(F: 1)]

Prim’s Demo

Insert all vertices into fringe PQ, storing vertices in order of distance from tree.

Repeat: Remove (closest) vertex v from PQ, and relax all edges pointing from v.

Node	distTo	edgeTo
A		**-**
B		**A**
C		**A**
D		**E**
E		**C**
F		**G**
G		**D**



PQ: []

Prim’s vs. Dijkstra’s

Prim’s and Dijkstra’s algorithms are exactly the same, except Dijkstra’s considers “distance from the source”, and Prim’s considers “distance from the tree.”

Visit order:

- Dijkstra’s algorithm visits vertices in order of distance from the source.
- Prim’s algorithm visits vertices in order of distance from the MST under construction.

Relaxation:

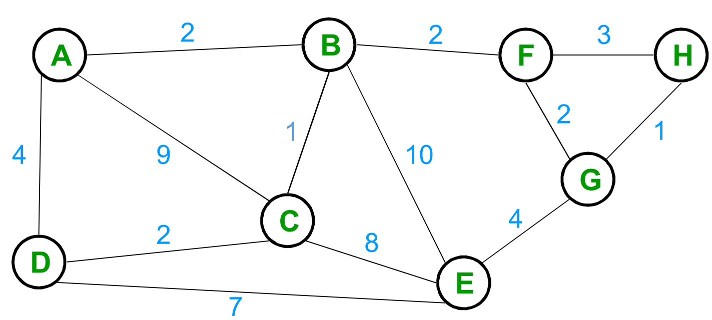
- Relaxation in Dijkstra’s considers an edge better based on distance to source.
- Relaxation in Prim’s considers an edge better based on distance to tree.

Prim’s Algorithm Runtime

The running time is $O(V^2)$ without heaps, and $O(E \ln(V))$ using binary heaps.

Quiz

- Finding MST of this graph using Prim’s Algorithm

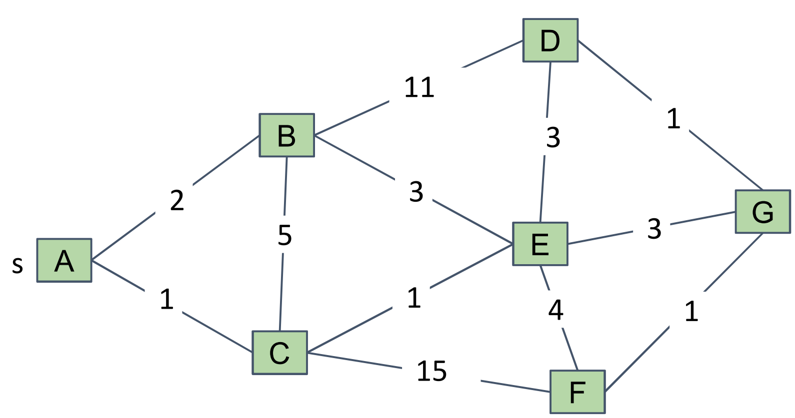


Basic Kruskal's (Demo)

Kruskal's Demo

Consider edges in order of increasing weight. Add to MST unless a cycle is created.
Repeat until V-1 edges.

A-C	1
C-E	1
D-G	1
F-G	1
A-B	2
E-B	3
D-E	3
G-E	3
E-F	4
B-C	5
B-D	11
C-F	15



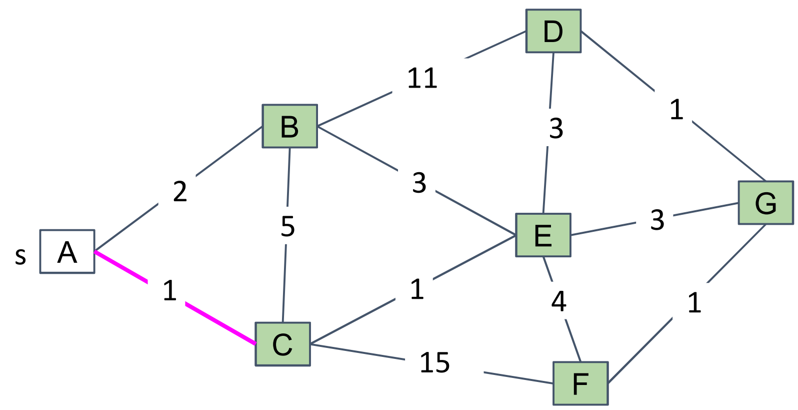
MST: []

Kruskal's Demo

Consider edges in order of increasing weight. Add to MST unless a cycle is created.
Repeat until V-1 edges.

A-C	1
C-E	1
D-G	1
F-G	1
A-B	2
E-B	3
D-E	3
G-E	3
E-F	4
B-C	5
B-D	11
C-F	15

Cycle
?



White and green colorings
for vertices show cut being

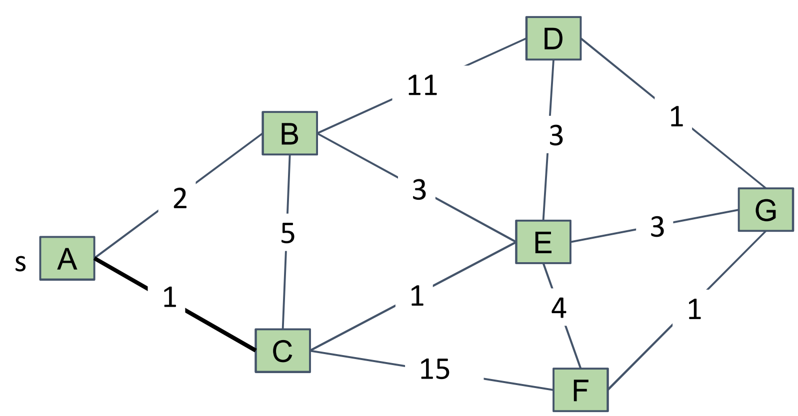
MST: []

Kruskal's Demo

Consider edges in order of increasing weight. Add to MST unless a cycle is created.
Repeat until V-1 edges.

~~A-C~~	~~1~~
C-E	1
D-G	1
F-G	1
A-B	2
E-B	3
D-E	3
G-E	3
E-F	4
B-C	5
B-D	11
C-F	15

Cycle?
No.



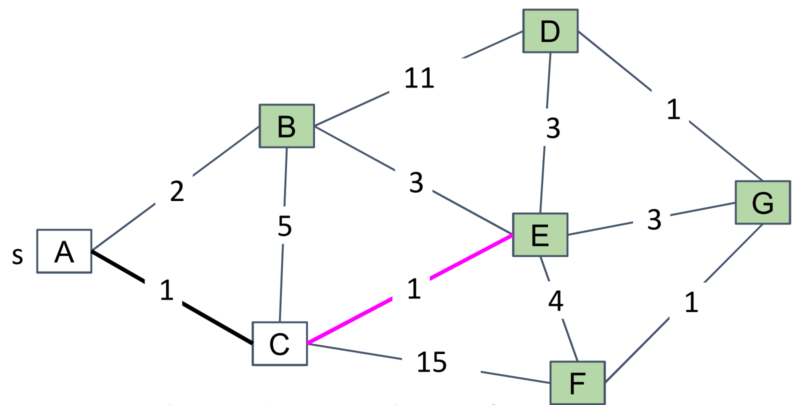
MST: [A-C]

Kruskal's Demo

Consider edges in order of increasing weight. Add to MST unless a cycle is created.
Repeat until V-1 edges.

~~A-C~~	~~1~~
C-E	1
D-G	1
F-G	1
A-B	2
E-B	3
D-E	3
G-E	3
E-F	4
B-C	5
B-D	11
C-F	15

Cycle? No.



White and green colorings for vertices show cut being implicitly utilized by Kruskal's algorithm.

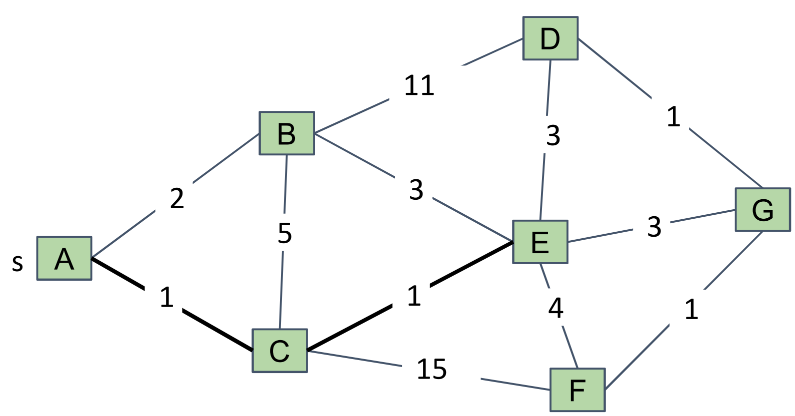
`MST: [A-C]`

Kruskal's Demo

Consider edges in order of increasing weight. Add to MST unless a cycle is created.
Repeat until V-1 edges.

~~A-C~~	~~1~~
~~C-E~~	~~1~~
D-G	1
F-G	1
A-B	2
E-B	3
D-E	3
G-E	3
E-F	4
B-C	5
B-D	11
C-F	15

Cycle? No.



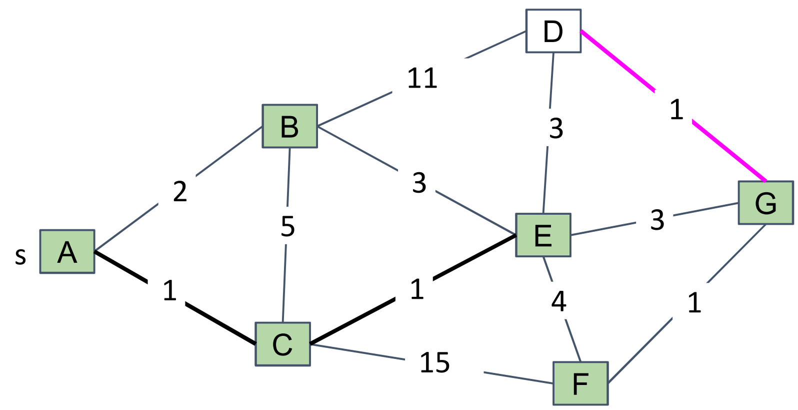
`MST: [A-C, C-E]`

Kruskal's Demo

Consider edges in order of increasing weight. Add to MST unless a cycle is created.
Repeat until V-1 edges.

~~A-C 1~~
~~C-E 1~~
D-G 1
F-G 1
A-B 2
E-B 3
D-E 3
G-E 3
E-F 4
B-C 5
B-D 11
C-F 15

Cycle?



White and green colorings for vertices show cut being implicitly utilized by Kruskal's algorithm.

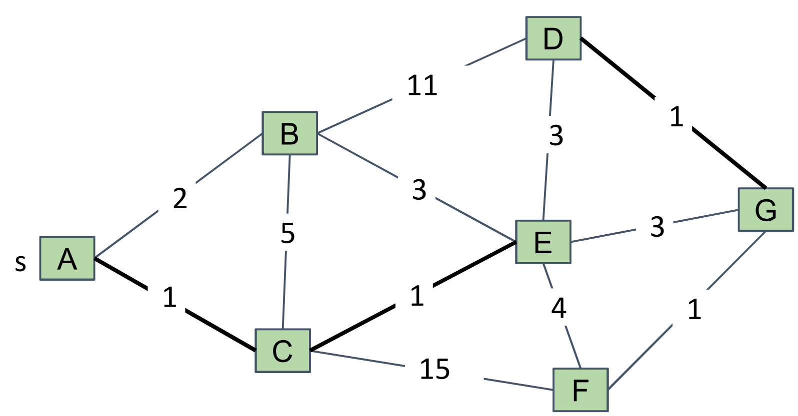
`MST: [A-C, C-E]`

Kruskal's Demo

Consider edges in order of increasing weight. Add to MST unless a cycle is created.
Repeat until V-1 edges.

~~A-C 1~~
~~C-E 1~~
~~D-G 1~~
F-G 1
A-B 2
E-B 3
D-E 3
G-E 3
E-F 4
B-C 5
B-D 11
C-F 15

Cycle? No.



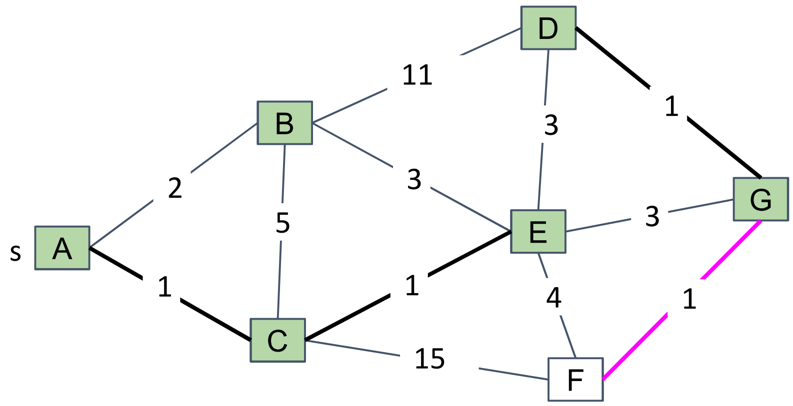
`MST: [A-C, C-E, D-G]`

Kruskal's Demo

Consider edges in order of increasing weight. Add to MST unless a cycle is created.
Repeat until V-1 edges.

~~A-C~~	~~1~~
~~C-E~~	~~1~~
~~D-G~~	~~1~~
F-G	1
A-B	2
E-B	3
D-E	3
G-E	3
E-F	4
B-C	5
B-D	11
C-F	15

Cycle?



White and green colorings for vertices show cut being implicitly utilized by Kruskal's algorithm.

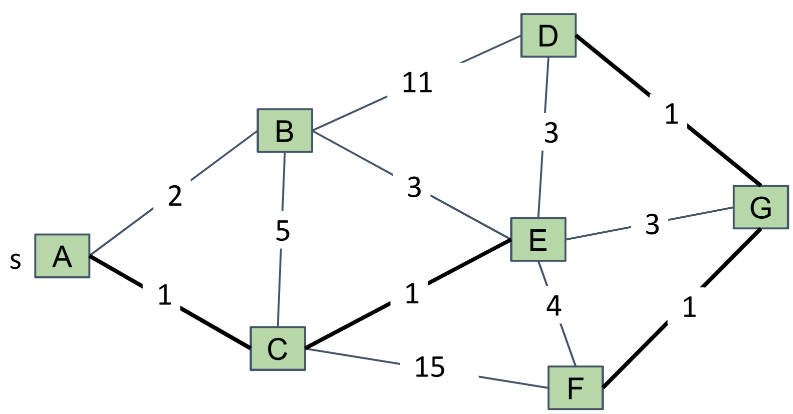
MST: [A-C, C-E, D-G]

Kruskal's Demo

Consider edges in order of increasing weight. Add to MST unless a cycle is created.
Repeat until V-1 edges.

~~A-C~~	~~1~~
~~C-E~~	~~1~~
~~D-G~~	~~1~~
~~F-G~~	~~1~~
A-B	2
E-B	3
D-E	3
G-E	3
E-F	4
B-C	5
B-D	11
C-F	15

Cycle? No.



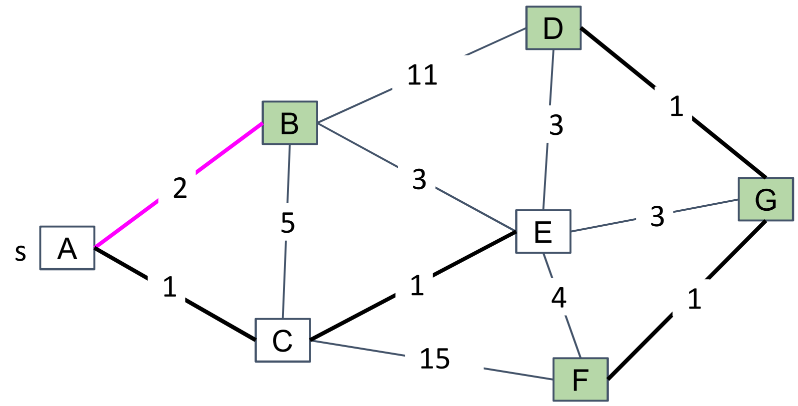
MST: [A-C, C-E, D-G, F-G]

Kruskal's Demo

Consider edges in order of increasing weight. Add to MST unless a cycle is created.
Repeat until V-1 edges.

~~A-C 1~~
~~C-E 1~~
~~D-G 1~~
~~F-G 1~~
A-B 2
E-B 3
D-E 3
G-E 3
E-F 4
B-C 5
B-D 11
C-F 15

Cycle?



White and green colorings for vertices show cut being implicitly utilized by Kruskal's algorithm.

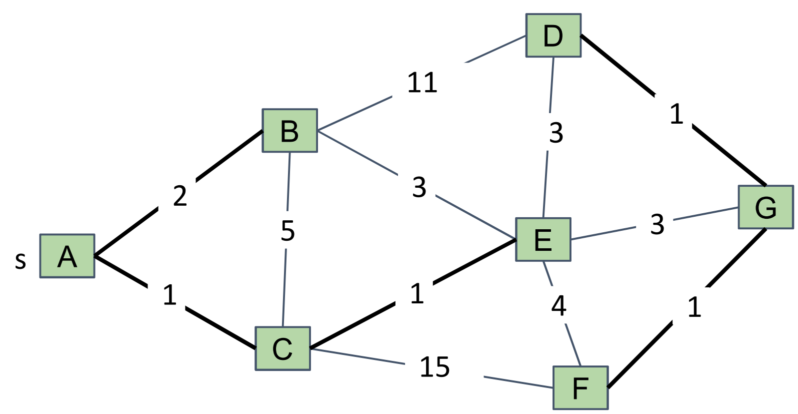
`MST: [A-C, C-E, D-G, F-G]`

Kruskal's Demo

Consider edges in order of increasing weight. Add to MST unless a cycle is created.
Repeat until V-1 edges.

~~A-C 1~~
~~C-E 1~~
~~D-G 1~~
~~F-G 1~~
~~A-B 2~~
E-B 3
D-E 3
G-E 3
E-F 4
B-C 5
B-D 11
C-F 15

Cycle? No.



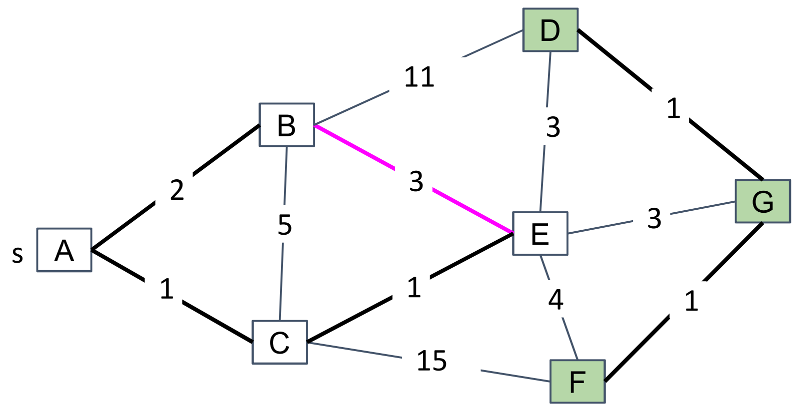
`MST: [A-C, C-E, D-G, F-G, A-B]`

Kruskal's Demo

Consider edges in order of increasing weight. Add to MST unless a cycle is created.
Repeat until V-1 edges.

~~A-C 1~~
~~C-E 1~~
~~D-G 1~~
~~F-G 1~~
~~A-B 2~~
E-B 3
D-E 3
G-E 3
E-F 4
B-C 5
B-D 11
C-F 15

Cycle?



White and green colorings for vertices show cut being implicitly utilized by Kruskal's algorithm.

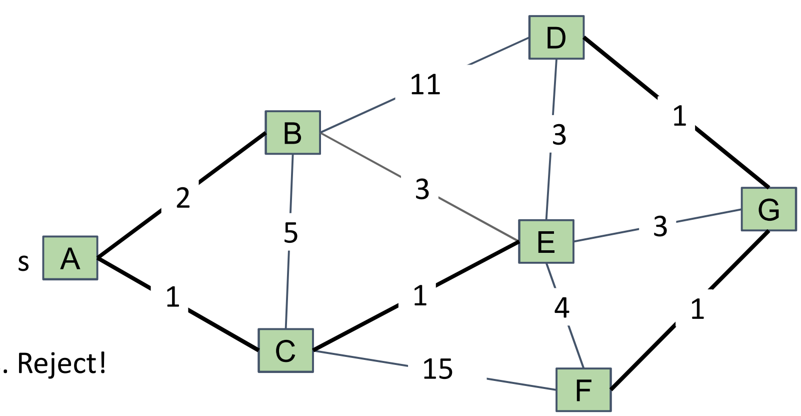
MST: [A-C, C-E, D-G, F-G, A-B]

Kruskal's Demo

Consider edges in order of increasing weight. Add to MST unless a cycle is created.
Repeat until V-1 edges.

~~A-C 1~~
~~C-E 1~~
~~D-G 1~~
~~F-G 1~~
~~A-B 2~~
~~E-B 3~~
D-E 3
G-E 3
E-F 4
B-C 5
B-D 11
C-F 15

Cycle? Yes. Reject!



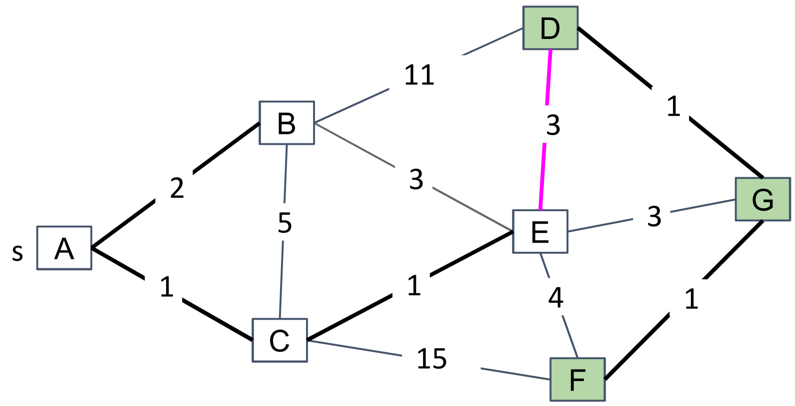
MST: [A-C, C-E, D-G, F-G, A-B]

Kruskal's Demo

Consider edges in order of increasing weight. Add to MST unless a cycle is created.
Repeat until V-1 edges.

~~A-C~~	~~1~~
~~C-E~~	~~1~~
~~D-G~~	~~1~~
~~F-G~~	~~1~~
~~A-B~~	~~2~~
~~E-B~~	~~3~~
D-E	3
G-E	3
E-F	4
B-C	5
B-D	11
C-F	15

Cycle?



White and green colorings for vertices show cut being implicitly utilized by Kruskal's algorithm.

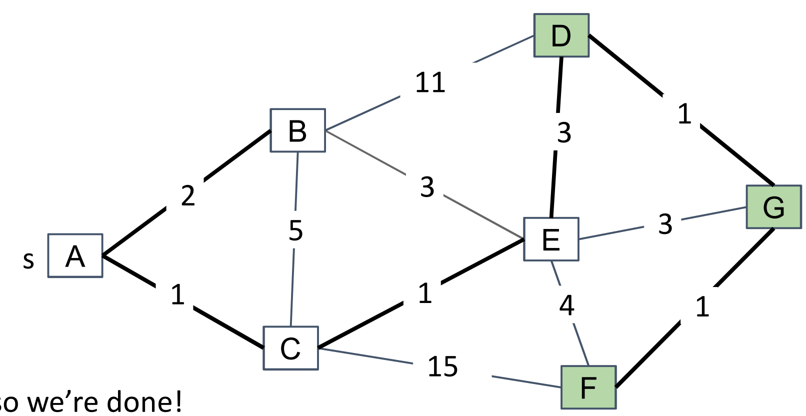
MST: [A-C, C-E, D-G, F-G, A-B]

Kruskal's Demo

Consider edges in order of increasing weight. Add to MST unless a cycle is created.
Repeat until V-1 edges.

~~A-C~~	~~1~~
~~C-E~~	~~1~~
~~D-G~~	~~1~~
~~F-G~~	~~1~~
~~A-B~~	~~2~~
~~E-B~~	~~3~~
~~D-E~~	~~3~~
G-E	3
E-F	4
B-C	5
B-D	11
C-F	15

Cycle? No.
V-1 edges, so we're done!

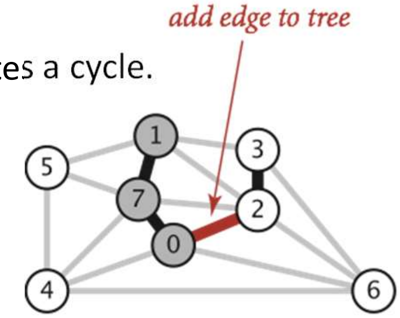


MST: [A-C, C-E, D-G, F-G, A-B, D-E]

Kruskal's Algorithm

Initially mark all edges gray.

- Consider edges in increasing order of weight.
- Add edge to MST (mark black) unless doing so creates a cycle.
- Repeat until V-1 edges.



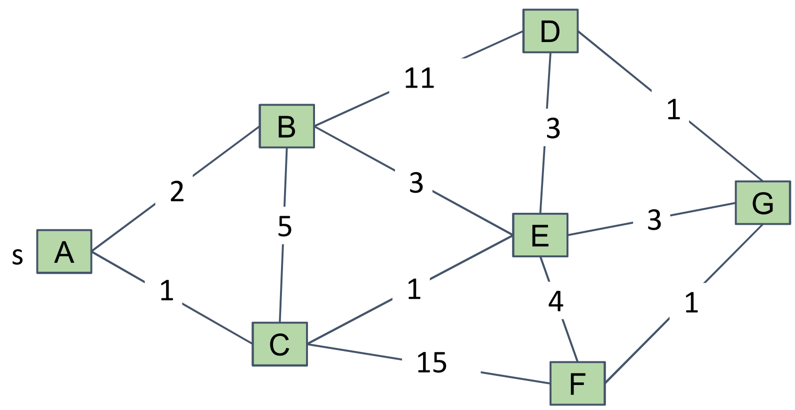
Optimized Kruskal's (Demo)

Kruskal's Demo

Insert all edges into PQ.

Repeat: Remove smallest weight edge. Add to MST if no cycle created.

```
PQ: (A-C: 1), (C-E: 1),
(D-G: 1), (F-G: 1),
(A-B: 2), (E-B: 3),
(D-E: 3), (G-E: 3),
(E-F: 4), (B-C: 5),
(B-D: 11), (C-F: 15)
```



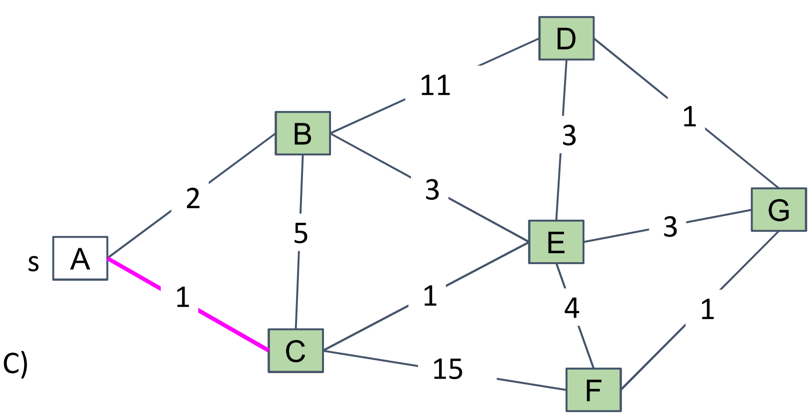

```
U: []
MST: []
```

Kruskal's Demo

Insert all edges into PQ.

Repeat: Remove smallest weight edge. Add to MST if no cycle created.

PQ: ~~(A-C: 1),~~ (C-E: 1),
(D-G: 1), (F-G: 1),
(A-B: 2), (E-B: 3),
(D-E: 3), (G-E: 3),
(E-F: 4), (B-C: 5),
(B-D: 11), (C-F: 15)



Removed edge: A-C

Cycle? isConnected(A, C)

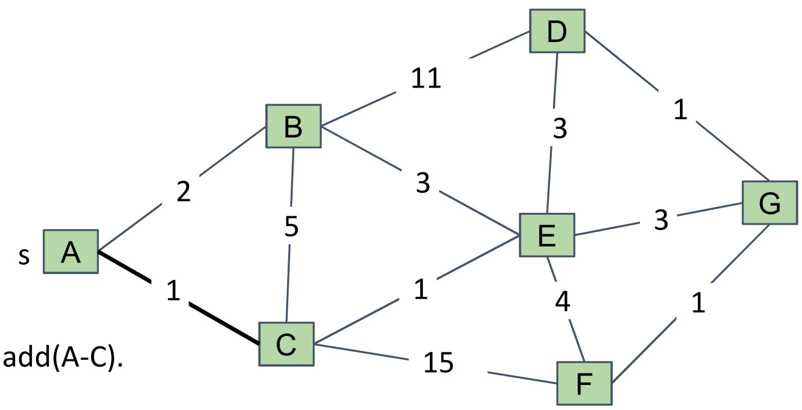
```
U: []
MST: []
```

Kruskal's Demo

Insert all edges into PQ.

Repeat: Remove smallest weight edge. Add to MST if no cycle created.

PQ: ~~(A-C: 1),~~ (C-E: 1),
(D-G: 1), (F-G: 1),
(A-B: 2), (E-B: 3),
(D-E: 3), (G-E: 3),
(E-F: 4), (B-C: 5),
(B-D: 11), (C-F: 15)



Removed edge: A-C

Cycle? No. union(A, C). add(A-C).

U: [$\boxed{\text{A-C}}$]

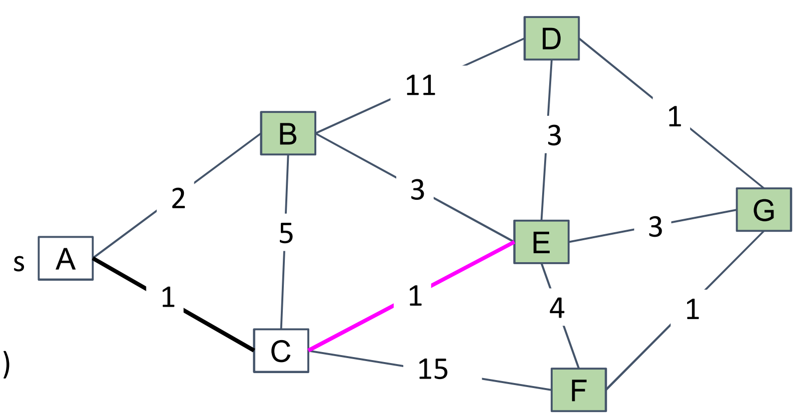
MST: [$\boxed{\text{A-C}}$]

Kruskal's Demo

Insert all edges into PQ.

Repeat: Remove smallest weight edge. Add to MST if no cycle created.

PQ: ~~(A-C: 1),~~ ~~(C-E: 1),~~
(D-G: 1), (F-G: 1),
(A-B: 2), (E-B: 3),
(D-E: 3), (G-E: 3),
(E-F: 4), (B-C: 5),
(B-D: 11), (C-F: 15)



Removed edge: C-E

Cycle? isConnected(C, E)

U: [A-C]

MST: [A-C]

Kruskal's Demo

Insert all edges into PQ.

Repeat: Remove smallest weight edge. Add to MST if no cycle created.

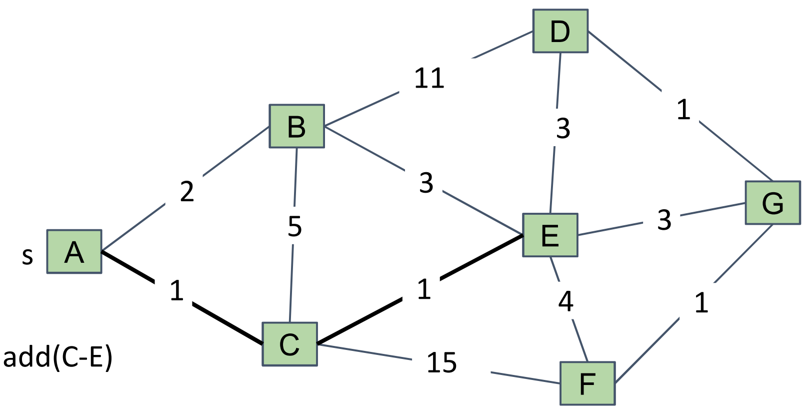
PQ: ~~(A-C: 1), (C-E: 1),~~
(D-G: 1), (F-G: 1),
(A-B: 2), (E-B: 3),
(D-E: 3), (G-E: 3),
(E-F: 4), (B-C: 5),
(B-D: 11), (C-F: 15)

Removed edge: C-E

Cycle? No. union(C, E). add(C-E)

U: [A-C-E]

MST: [A-C, C-E]



Kruskal's Demo

Insert all edges into PQ.

Repeat: Remove smallest weight edge. Add to MST if no cycle created.

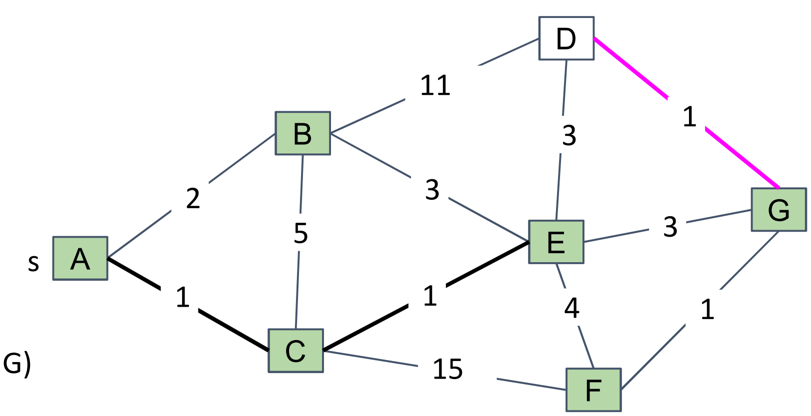
PQ: ~~(A-C: 1), (C-E: 1),~~
~~(D-G: 1),~~ (F-G: 1),
(A-B: 2), (E-B: 3),
(D-E: 3), (G-E: 3),
(E-F: 4), (B-C: 5),
(B-D: 11), (C-F: 15)

Removed edge: D-G

Cycle? isConnected(D, G)

U: [A-C-E]

MST: [A-C, C-E]

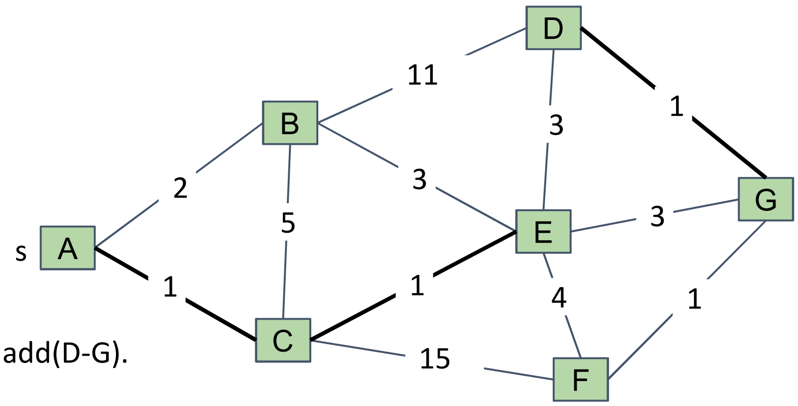


Kruskal’s Demo

Insert all edges into PQ.

Repeat: Remove smallest weight edge. Add to MST if no cycle created.

PQ: ~~(A-C: 1), (C-E: 1), (D-G: 1),~~ (F-G: 1), (A-B: 2), (E-B: 3), (D-E: 3), (G-E: 3), (E-F: 4), (B-C: 5), (B-D: 11), (C-F: 15)



Removed edge: D-G

Cycle? No. union(D, G). add(D-G).

U: [A-C-E, D-G]

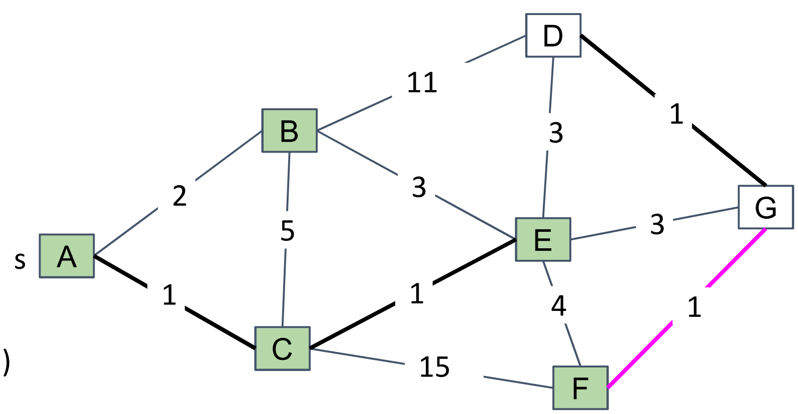
MST: [A-C, C-E, D-G]

Kruskal’s Demo

Insert all edges into PQ.

Repeat: Remove smallest weight edge. Add to MST if no cycle created.

PQ: ~~(A-C: 1), (C-E: 1), (D-G: 1), (F-G: 1),~~ (A-B: 2), (E-B: 3), (D-E: 3), (G-E: 3), (E-F: 4), (B-C: 5), (B-D: 11), (C-F: 15)



Removed edge: F-G

Cycle? isConnected(F, G)

U: [A-C-E, D-G]

MST: [A-C, C-E, D-G]

Kruskal’s Demo

Insert all edges into PQ.

Repeat: Remove smallest weight edge. Add to MST if no cycle created.

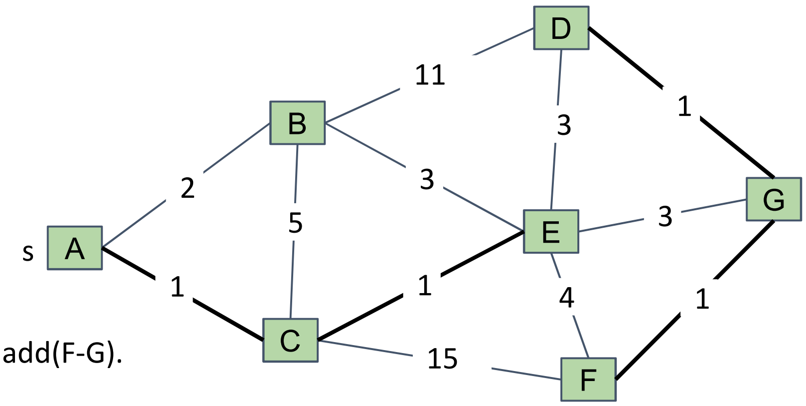
```
PQ: ~~(A-C: 1)~~, ~~(C-E: 1)~~,
~~(D-G: 1)~~, ~~(F-G: 1)~~,
(A-B: 2), (E-B: 3),
(D-E: 3), (G-E: 3),
(E-F: 4), (B-C: 5),
(B-D: 11), (C-F: 15)
```

Removed edge: F-G

Cycle? No. union(F, G). add(F-G).

`U: [A-C-E, D-G-F]`

`MST: [A-C, C-E, D-G, F-G]`



Kruskal’s Demo

Insert all edges into PQ.

Repeat: Remove smallest weight edge. Add to MST if no cycle created.

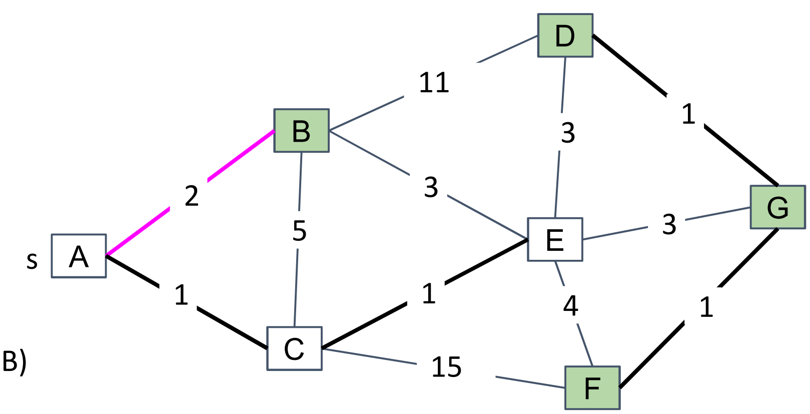
```
PQ: ~~(A-C: 1)~~, ~~(C-E: 1)~~,
~~(D-G: 1)~~, ~~(F-G: 1)~~,
~~(A-B: 2)~~, (E-B: 3),
(D-E: 3), (G-E: 3),
(E-F: 4), (B-C: 5),
(B-D: 11), (C-F: 15)
```

Removed edge: A-B

Cycle? isConnected(A, B)

`U: [A-C-E, D-G-F]`

`MST: [A-C, C-E, D-G, F-G]`

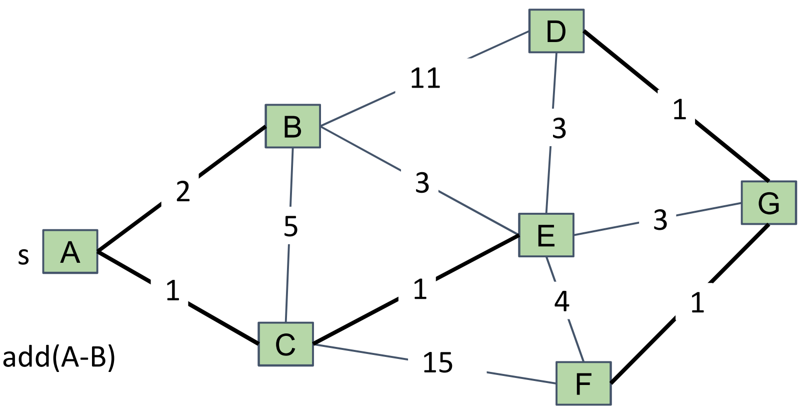


Kruskal’s Demo

Insert all edges into PQ.

Repeat: Remove smallest weight edge. Add to MST if no cycle created.

```
PQ: ~~(A-C: 1), (C-E: 1),~~
~~(D-G: 1), (F-G: 1),~~
~~(A-B: 2),~~ (E-B: 3),
(D-E: 3), (G-E: 3),
(E-F: 4), (B-C: 5),
(B-D: 11), (C-F: 15)
```



Removed edge: A-B

Cycle? No. union(A, B). add(A-B)

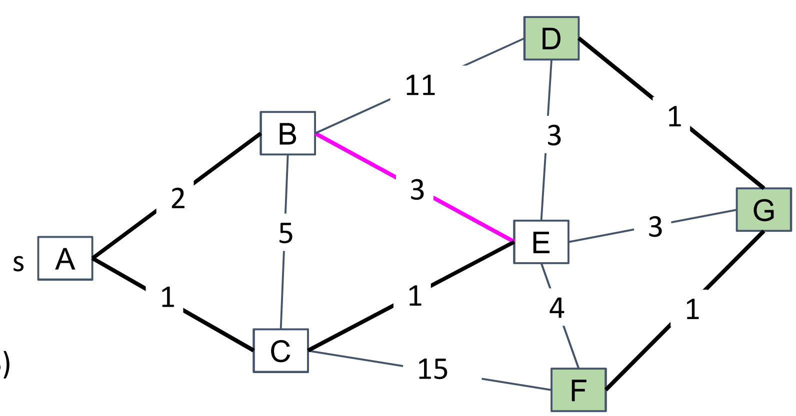
```
U: [A-C-E-B, D-G-F]
MST: [A-C, C-E, D-G, F-G, A-B]
```

Kruskal’s Demo

Insert all edges into PQ.

Repeat: Remove smallest weight edge. Add to MST if no cycle created.

```
PQ: ~~(A-C: 1), (C-E: 1),~~
~~(D-G: 1), (F-G: 1),~~
~~(A-B: 2), (E-B: 3),~~
(D-E: 3), (G-E: 3),
(E-F: 4), (B-C: 5),
(B-D: 11), (C-F: 15)
```



Removed edge: E-B

Cycle? isConnected(E, B)

```
U: [A-C-E-B, D-G-F]
MST: [A-C, C-E, D-G, F-G, A-B]
```

Kruskal's Demo

Insert all edges into PQ.

Repeat: Remove smallest weight edge. Add to MST if no cycle created.

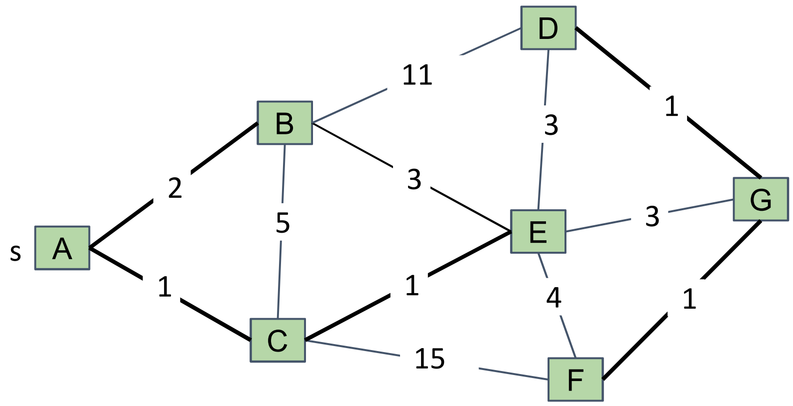
`PQ:` ~~`(A-C: 1)`~~, ~~`(C-E: 1)`~~,
~~`(D-G: 1)`~~, ~~`(F-G: 1)`~~,
~~`(A-B: 2)`~~, ~~`(E-B: 3)`~~,
`(D-E: 3), (G-E: 3),`
`(E-F: 4), (B-C: 5),`
`(B-D: 11), (C-F: 15)`

Removed edge: E-B

Cycle? Yes. Do nothing.

`U: [A-C-E-B, D-G-F]`

`MST: [A-C, C-E, D-G, F-G, A-B]`



Kruskal's Demo

Insert all edges into PQ.

Repeat: Remove smallest weight edge. Add to MST if no cycle created.

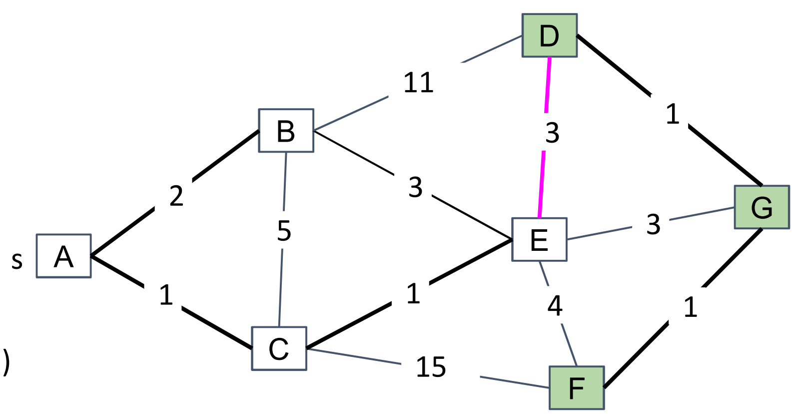
`PQ:` ~~`(A-C: 1)`~~, ~~`(C-E: 1)`~~,
~~`(D-G: 1)`~~, ~~`(F-G: 1)`~~,
~~`(A-B: 2)`~~, ~~`(E-B: 3)`~~,
~~`(D-E: 3)`~~, `(G-E: 3),`
`(E-F: 4), (B-C: 5),`
`(B-D: 11), (C-F: 15)`

Removed edge: D-E

Cycle? isConnected(D, E)

`U: [A-C-E-B, D-G-F]`

`MST: [A-C, C-E, D-G, F-G, A-B]`

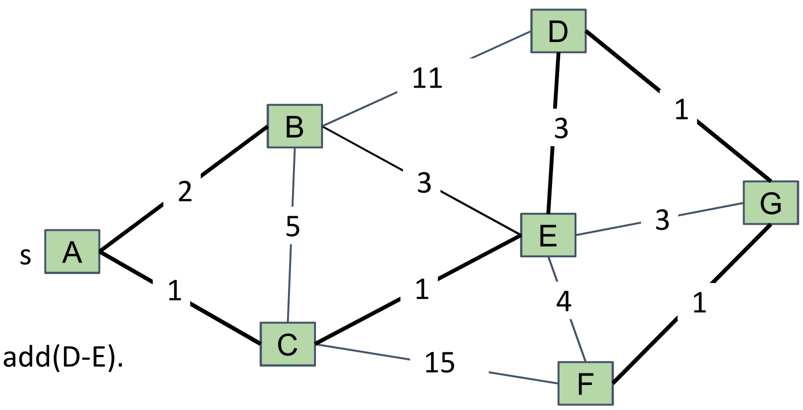


Kruskal's Demo

Insert all edges into PQ.

Repeat: Remove smallest weight edge. Add to MST if no cycle created.

PQ: ~~(A-C: 1)~~, ~~(C-E: 1)~~,
~~(D-G: 1)~~, ~~(F-G: 1)~~,
~~(A-B: 2)~~, ~~(E-B: 3)~~,
~~(D-E: 3)~~, (G-E: 3),
(E-F: 4), (B-C: 5),
(B-D: 11), (C-F: 15)



Removed edge: D-E

Cycle? No. union(D, E). add(D-E).

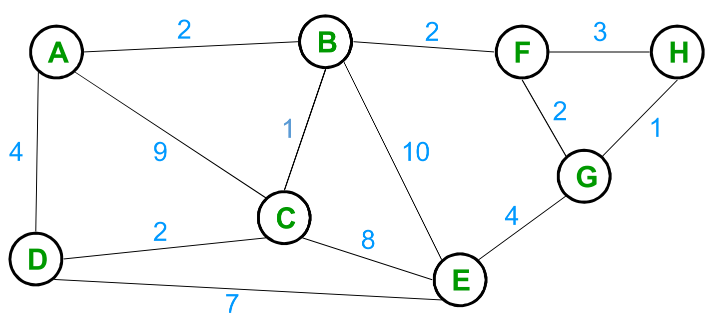
U: [A-C-E-$\boxed{\text{B-D}}$-G-F]

MST: [A-C, C-E, D-G, F-G, A-B, $\boxed{\text{D-E}}$]

V-1 edges, so done.

Quiz

- Finding the MST of this graph using Kruskal's Algorithm



Shortest Paths and MST Algorithms Summary

Problem	Algorithm	Runtime (if E > V)	Notes
Shortest Paths	Dijkstra's	O(E log V)	Fails for negative weight edges.
MST	Prim's	O(E log V)	Analogous to Dijkstra's.
MST	Kruskal's	O(E log E)	Uses Union.

Next week (Monday afternoon 03/06)

- Online Assignment 4 will be held on Monday afternoon 03/06
  - Content: Graph
  - Duration: 30 minutes
- Algorithms: Greedy Algorithm, Divide and Conquer Algorithm