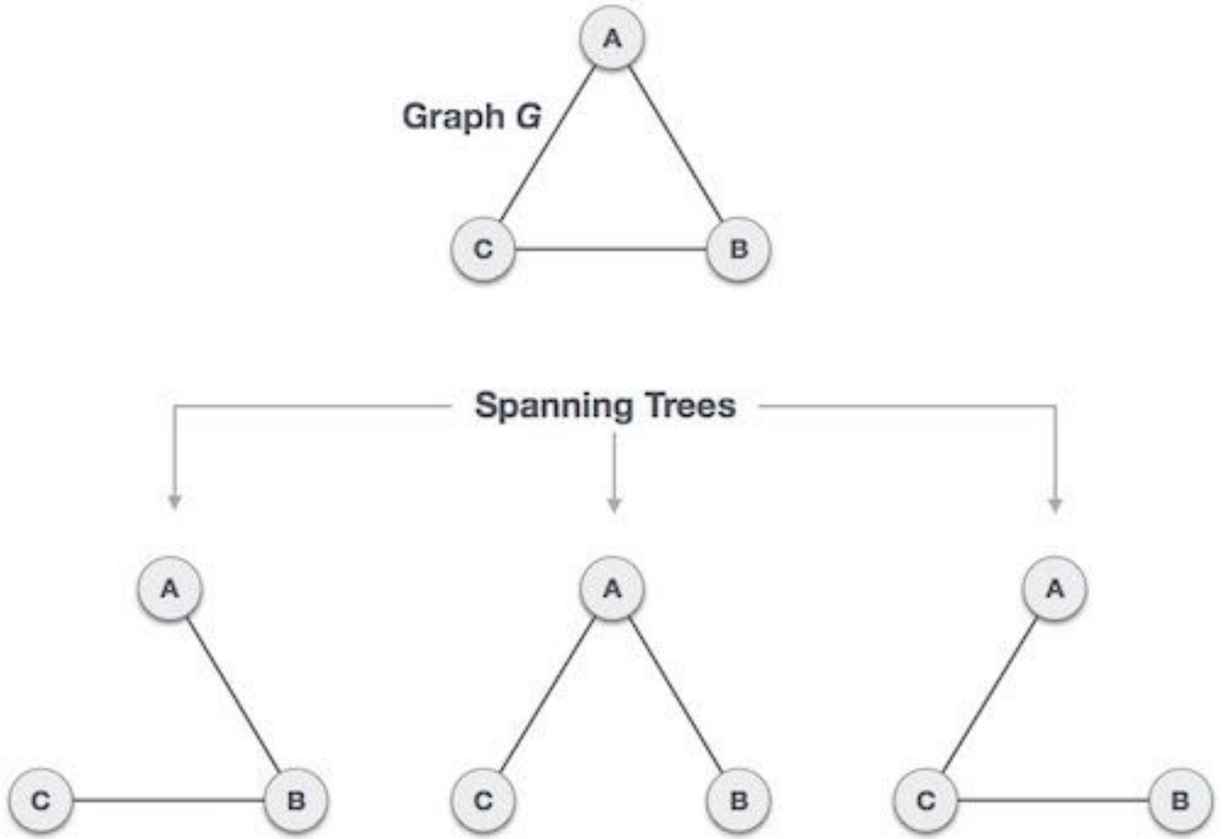
CS62 Class 22: Minimum Spanning Trees

Graphs



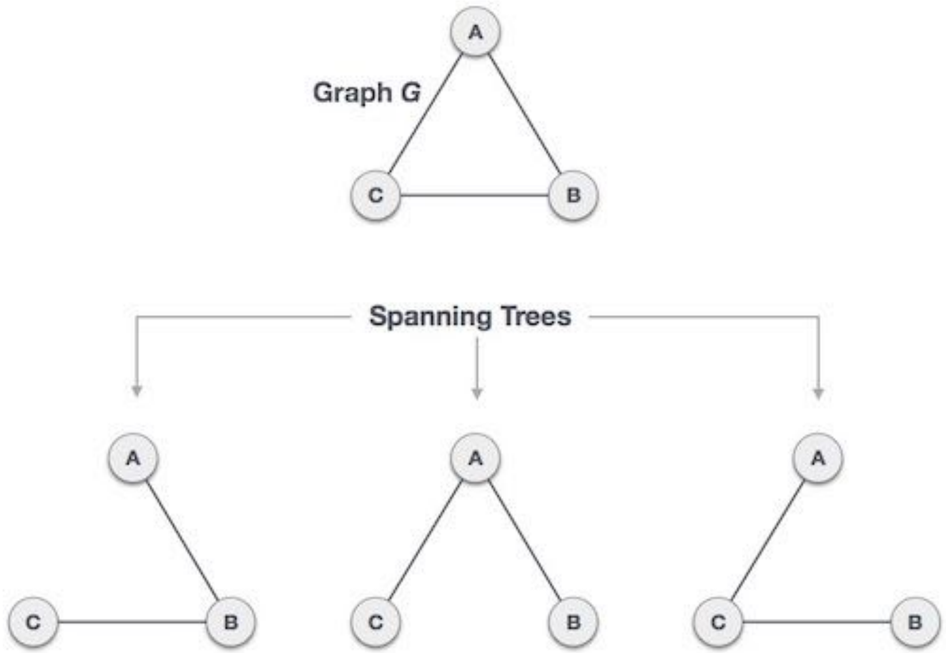
Agenda

- Minimum spanning trees
  - The cut property
- Prim’s Algorithm
- Kruskal’s Algorithm

Minimum Spanning Trees (MSTs)

Spanning Trees

- Given an edge weighted graph G (not digraph!), a spanning tree of G is a subgraph T that is:
  - A tree: connected and acyclic.
  - Spanning: includes all of the vertices of G.

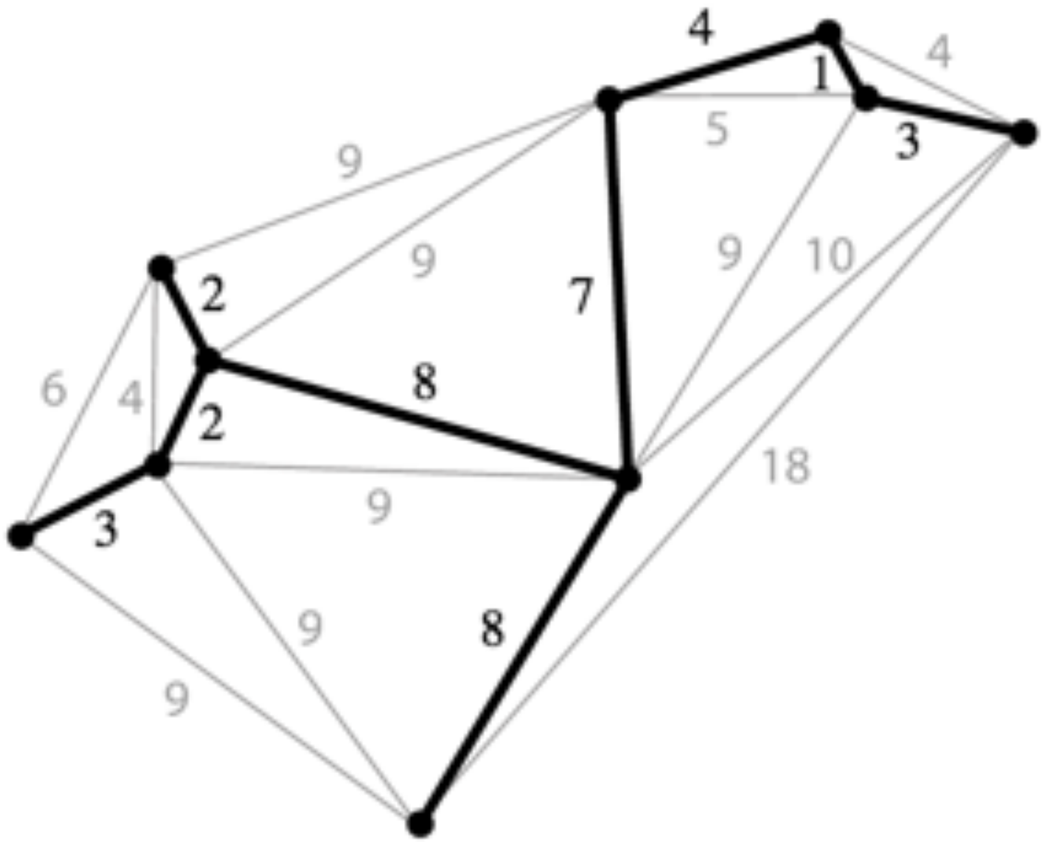


https://www.tutorialspoint.com/data_structures_algorithms/spanning_tree.htm

Minimum Spanning Trees

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

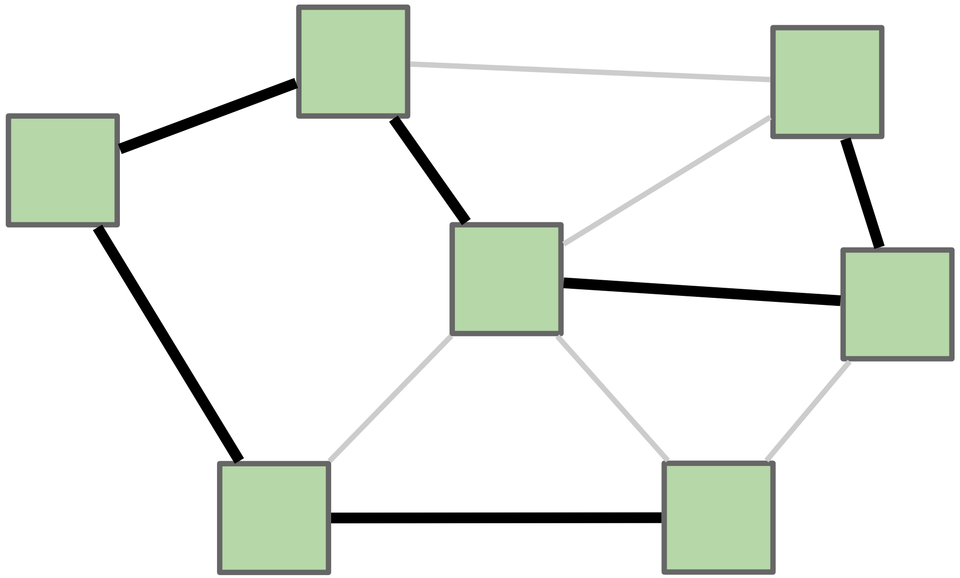
- Example: Network of power lines that connect a bunch of buildings.

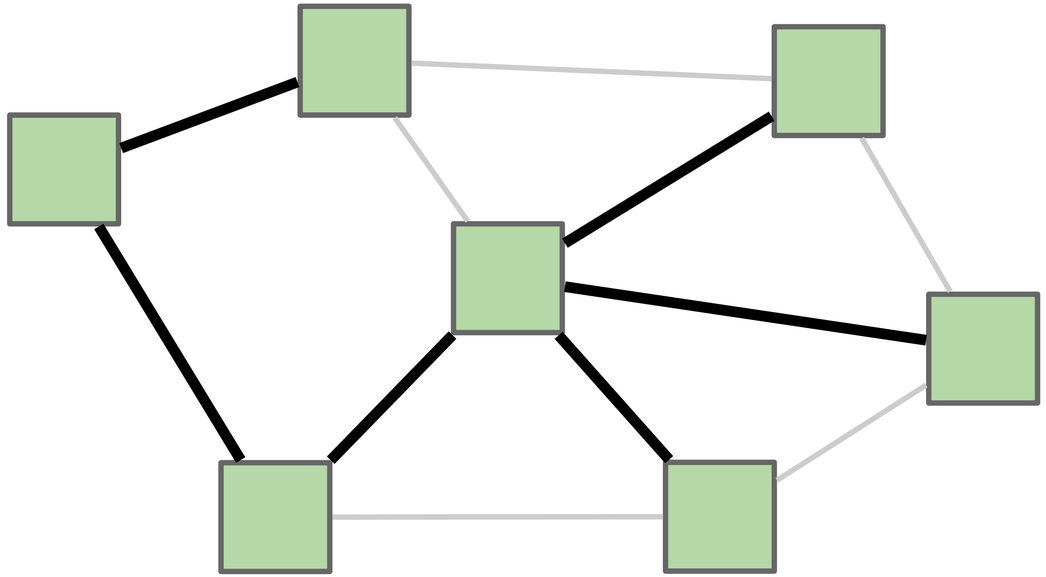


Properties

- A connected graph G can have more than one spanning tree, but only one minimum spanning tree (assuming unique weights).
- All possible spanning trees of G have the same number of vertices and edges.
  - A spanning tree has $|V| - 1$ edges.
- A spanning tree by definition cannot have any cycle.
  - Adding one edge to the spanning tree would create a cycle (i.e. spanning trees are maximally acyclic).
- Removing one edge from the spanning tree would make the graph disconnected (i.e. spanning trees are minimally connected).

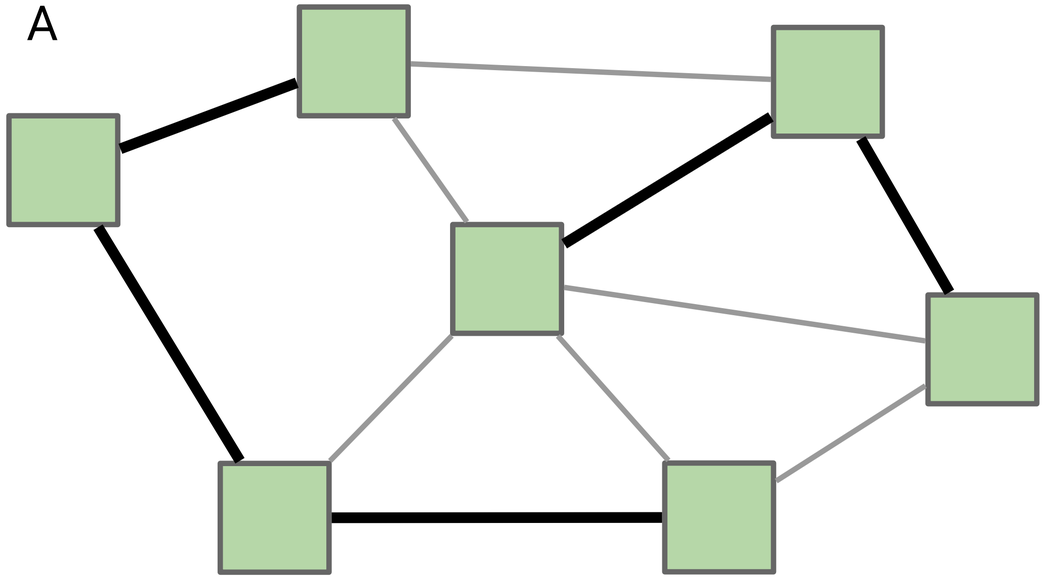
Spanning Trees



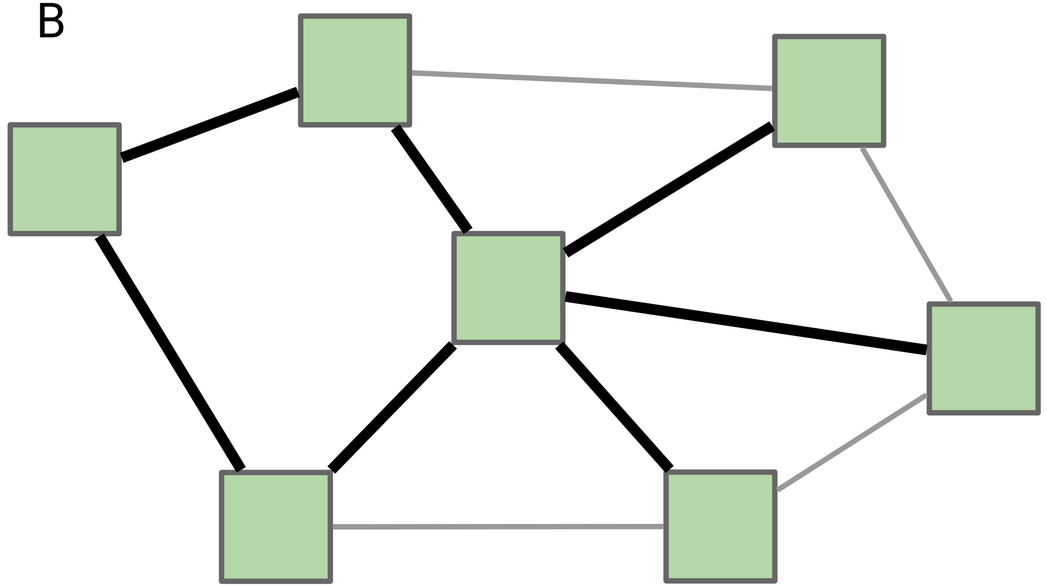


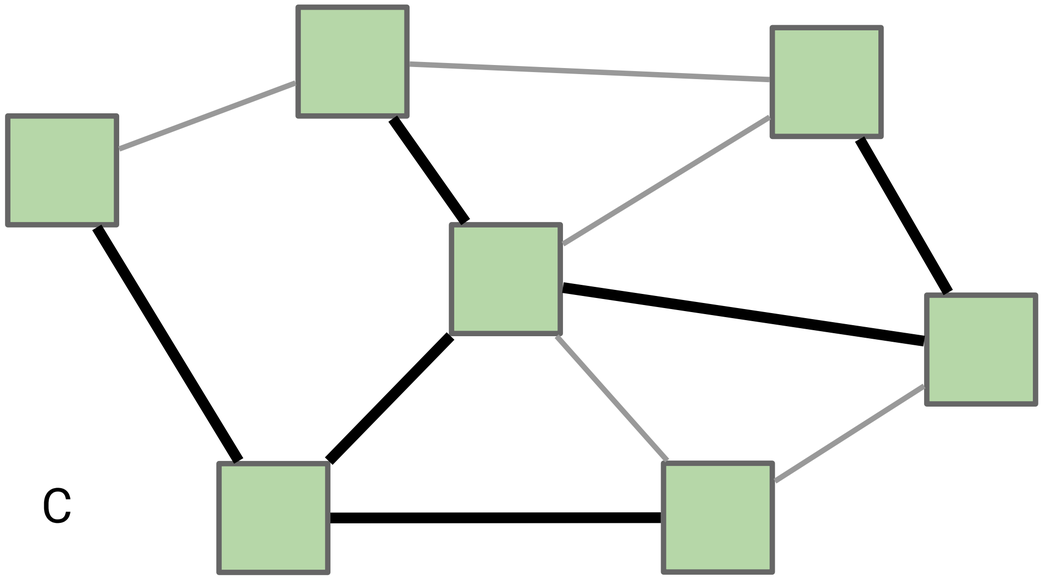
Which are Spanning Trees?

A



B





C

Answer: just C

MST Applications

Left: Old school handwriting recognition (link)

Right: Medical imaging (e.g. arrangement of nuclei in cancer cells)

For more, see: http://www.ics.uci.edu/~eppstein/gina/mst.html

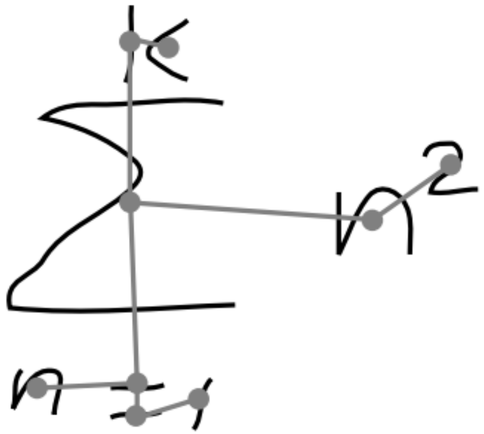
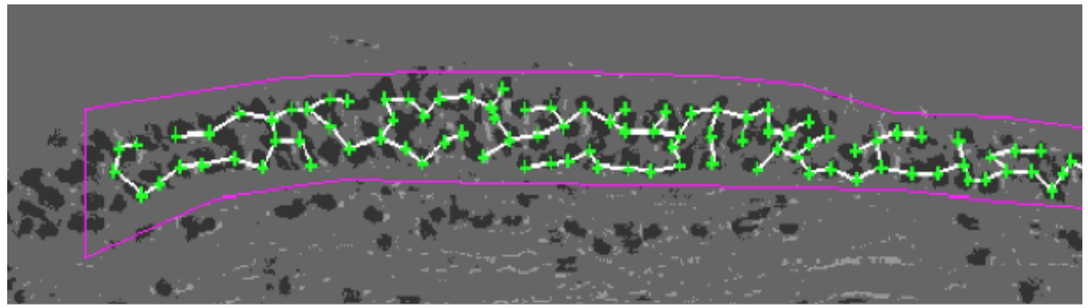
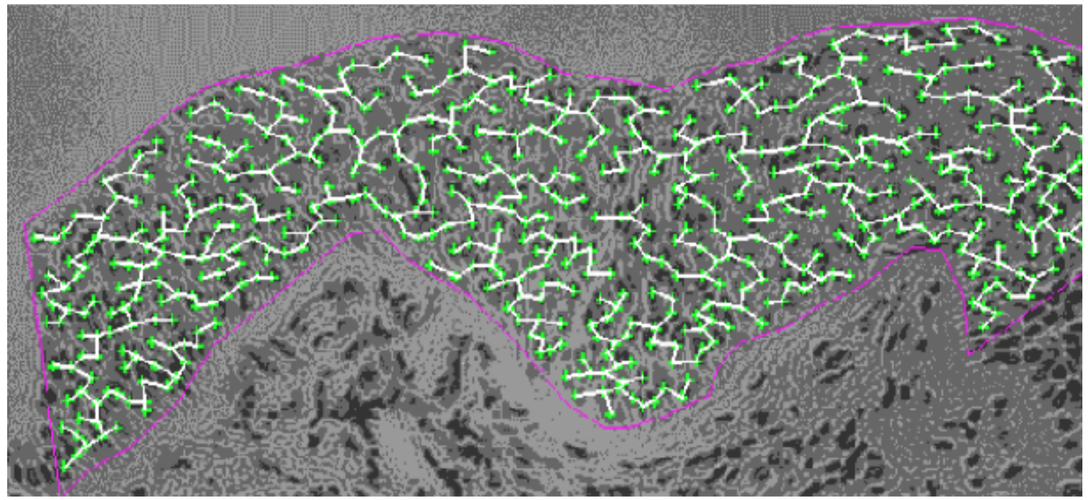


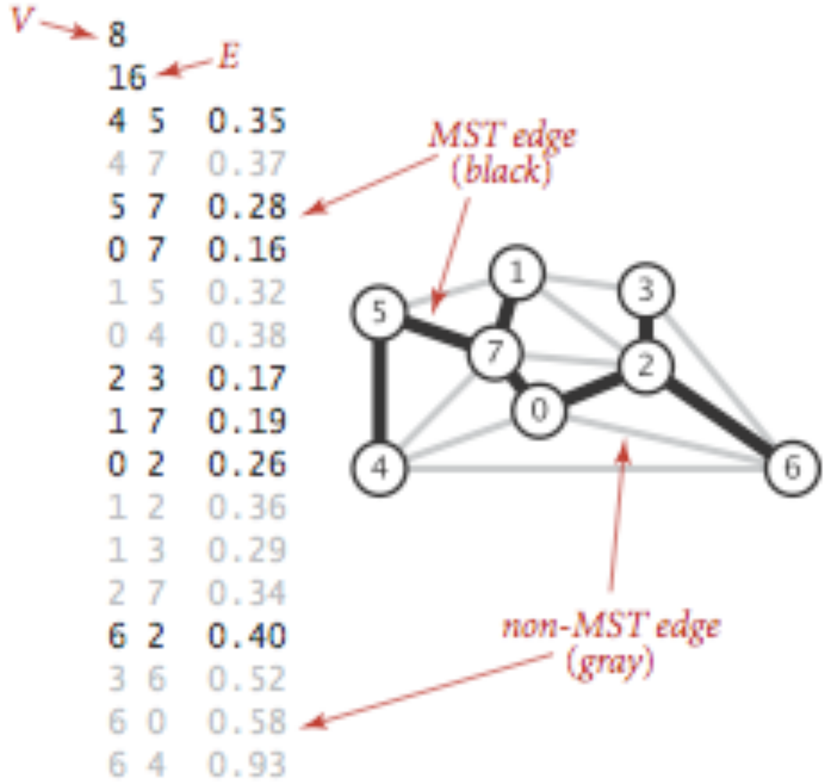
Figure 4-3: A typical minimum spanning tree





Today: 2 algorithms to find the MST

Given a connected edge-weighted **undirected** graph, find a spanning tree of minimum weight.

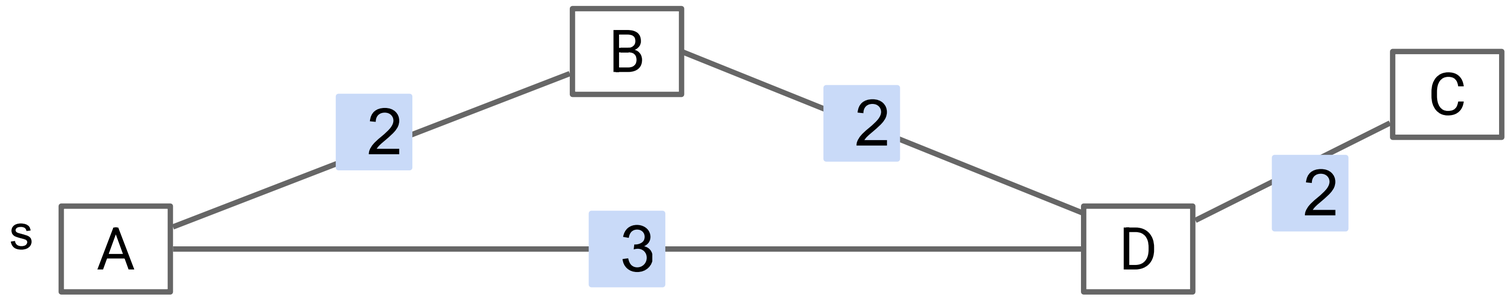


An edge-weighted graph and its MST

MST vs SPT

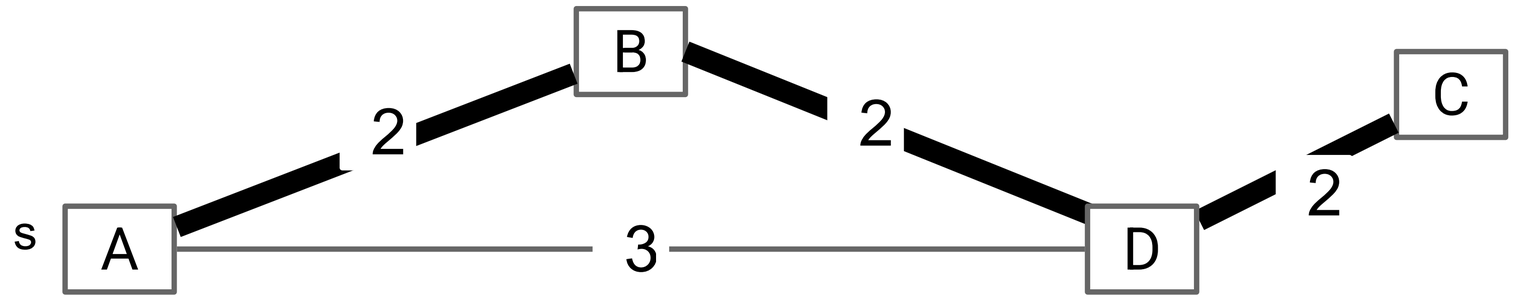
Worksheet time!

- What's the difference between a minimum spanning tree versus a shortest path tree?
- Find the MST for the graph.
- For the SPT for the graph starting at s. (Note it's an undirected graph).

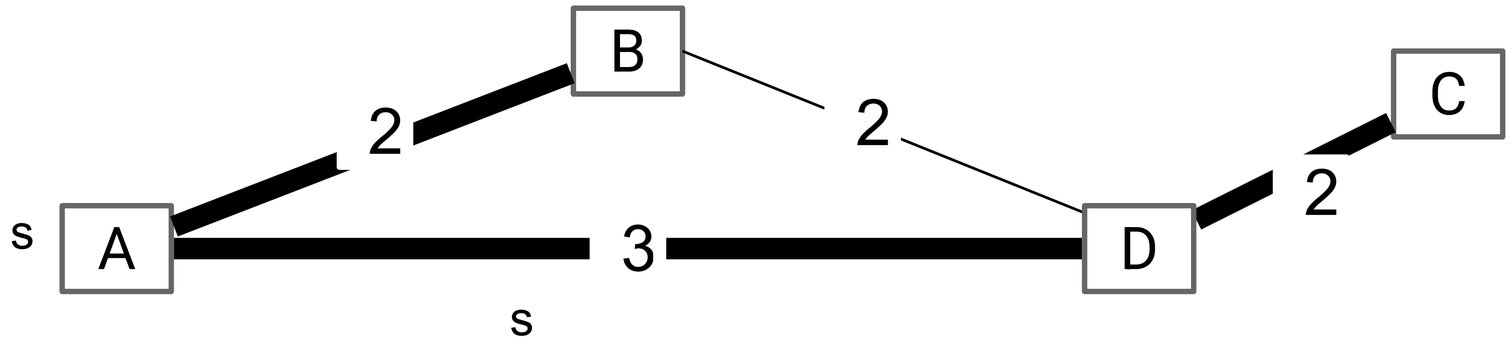


Worksheet answers

- Find the MST for the graph.



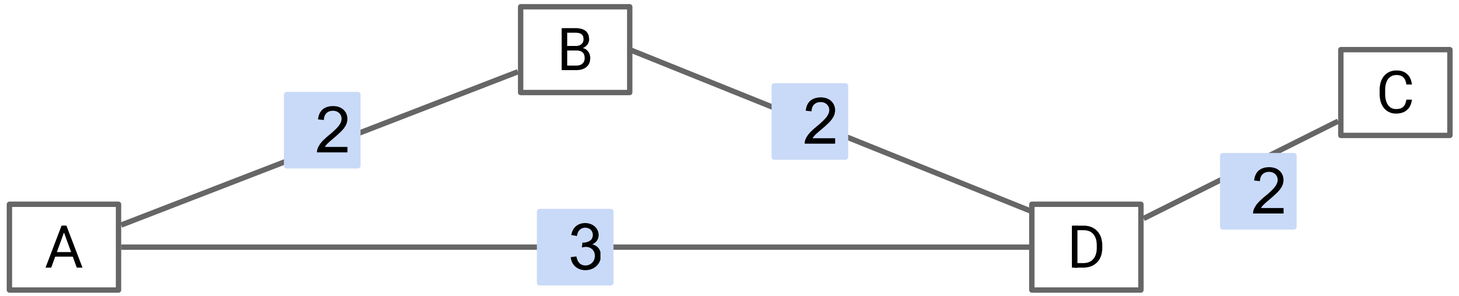
- For the SPT for the graph starting at s. (Note it's an undirected graph).



MST = SPT?

Using which node as the starting node for this SPT will result in an MST?

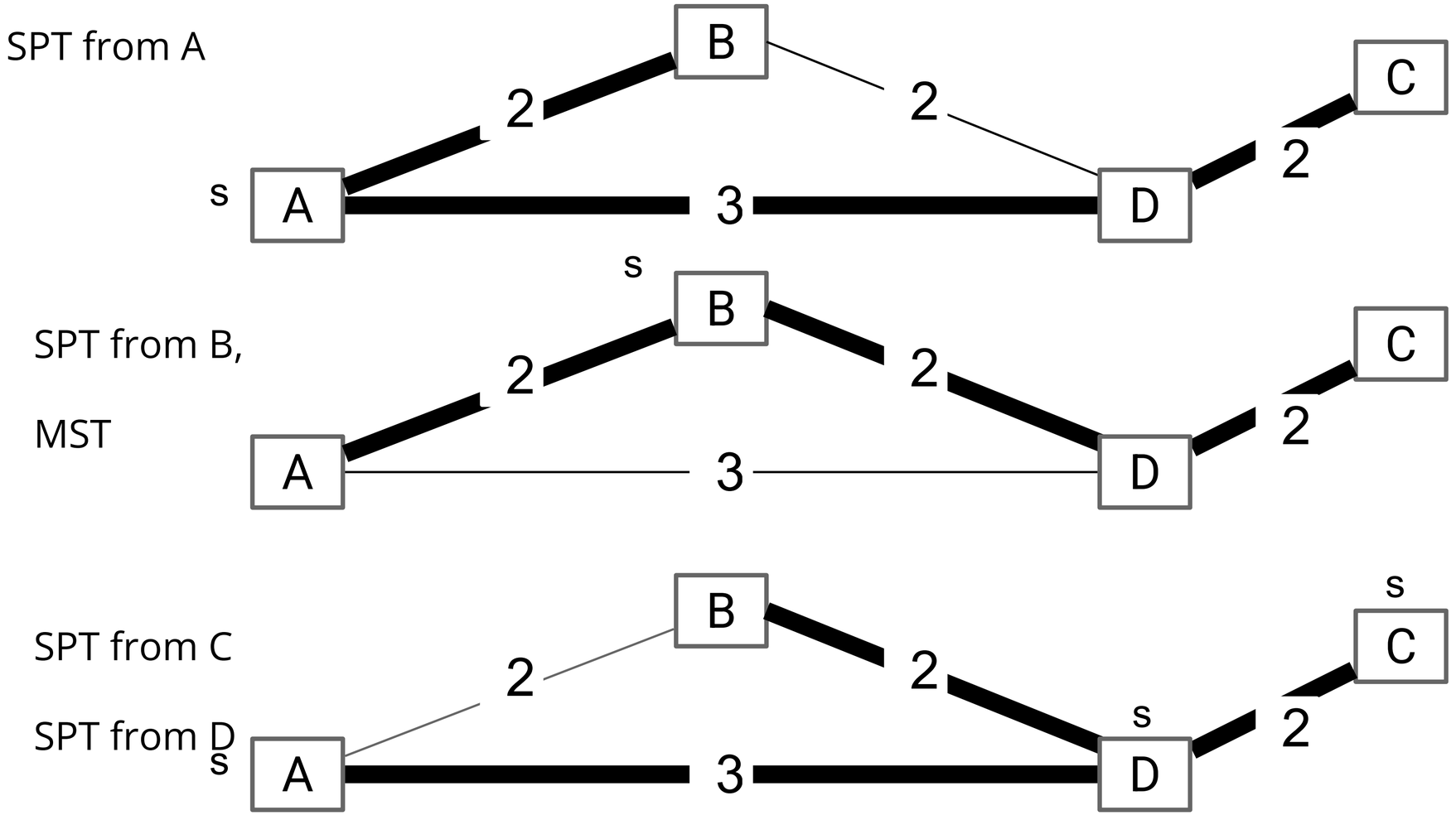
- A
- B
- C
- D
- No SPT is an MST.



MST = SPT, only sometimes

Using which node as the starting node for this SPT will result in an MST?

- A
- **B**
- C
- D
- No SPT is an MST.



MST = SPT, only sometimes

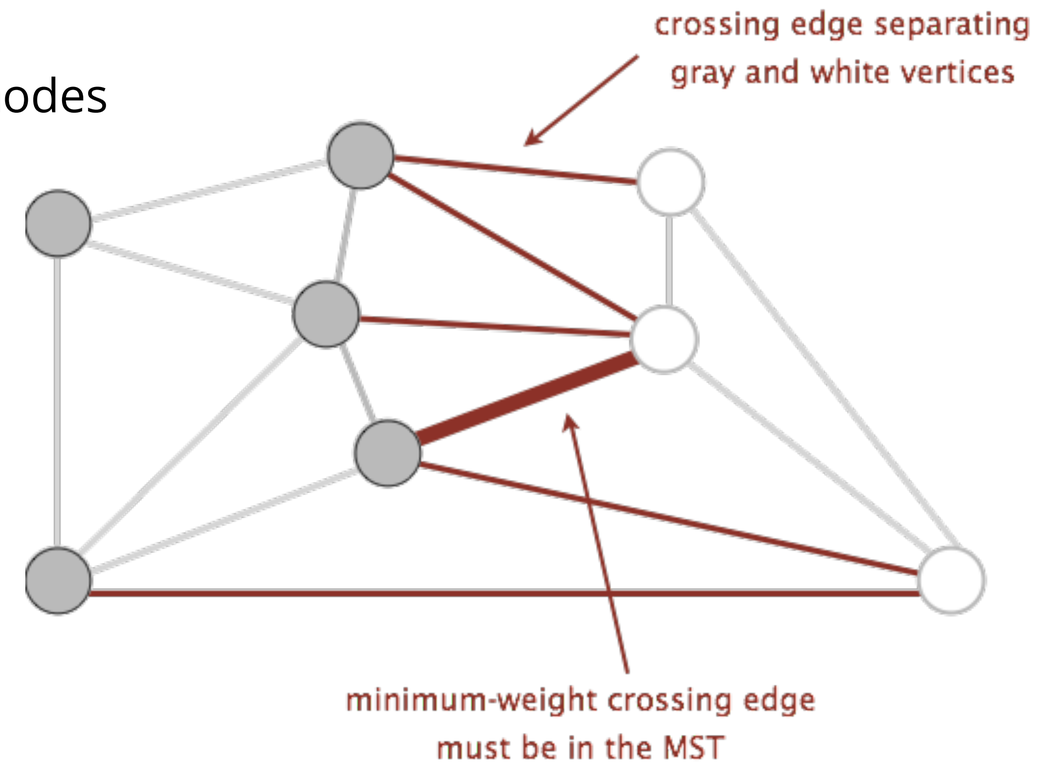
- A shortest paths tree depends on the start vertex, because it tells you how to get from a source to EVERYTHING.
- There is no source for a MST.
- Nonetheless, the MST sometimes happens to be an SPT for a specific vertex.

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: grey nodes vs white nodes

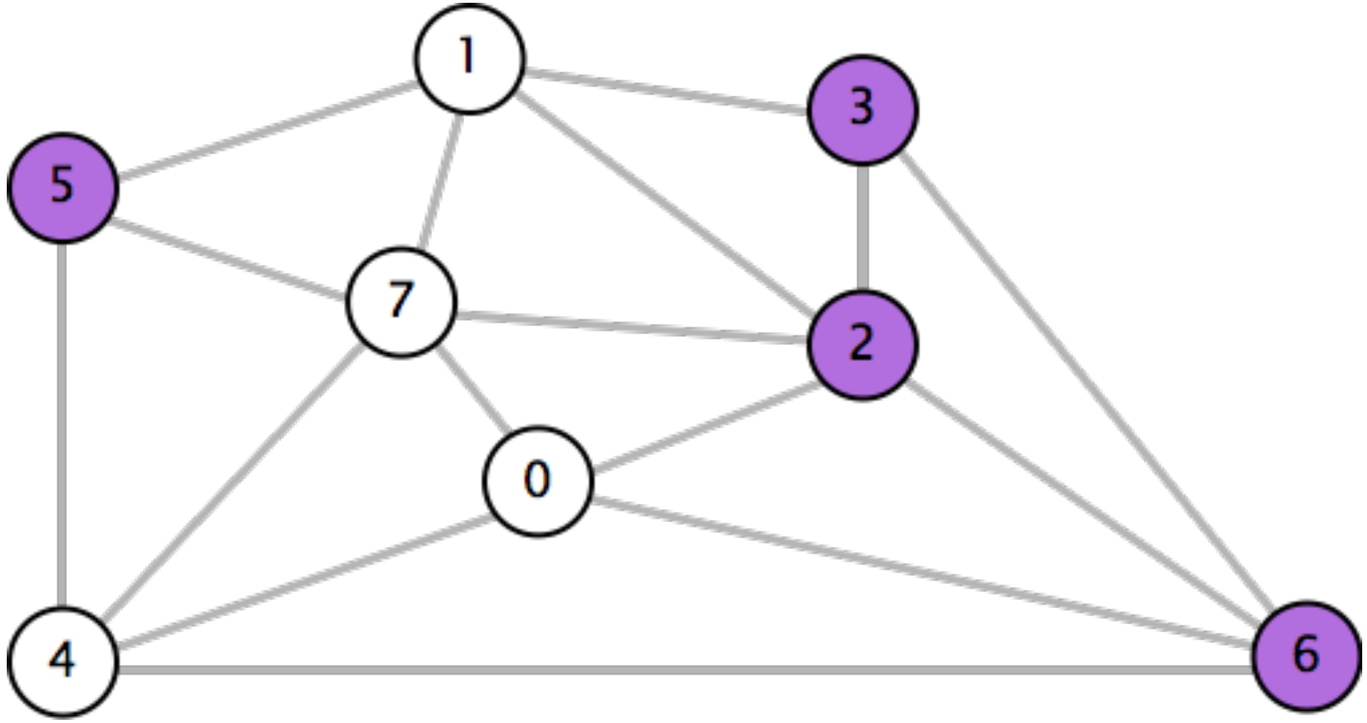


Cut property: Given any cut, minimum weight crossing edge is in the MST.

- 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}?

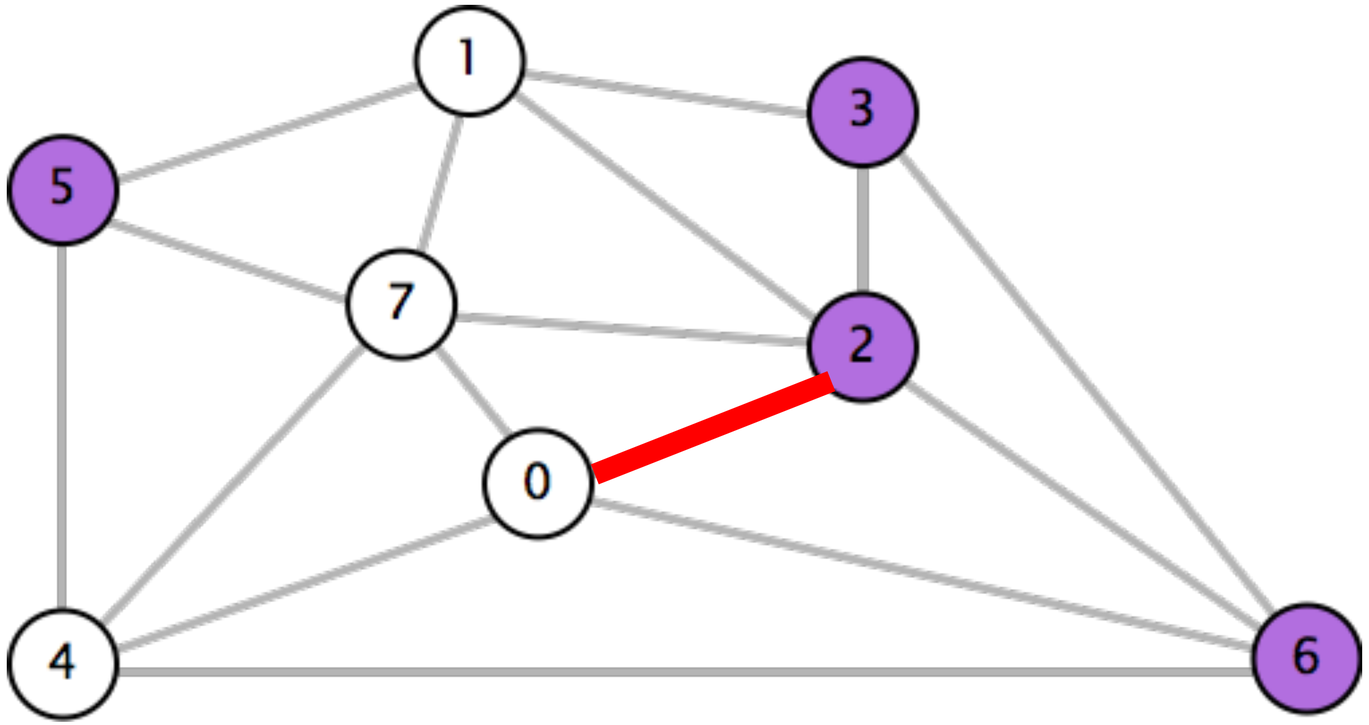



```
0-7   0.16
2-3   0.17
1-7   0.19
0-2   0.26
5-7   0.28
1-3   0.29
1-5   0.32
2-7   0.34
4-5   0.35
1-2   0.36
4-7   0.37
0-4   0.38
6-2   0.40
3-6   0.52
6-0   0.58
6-4   0.93
```

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!

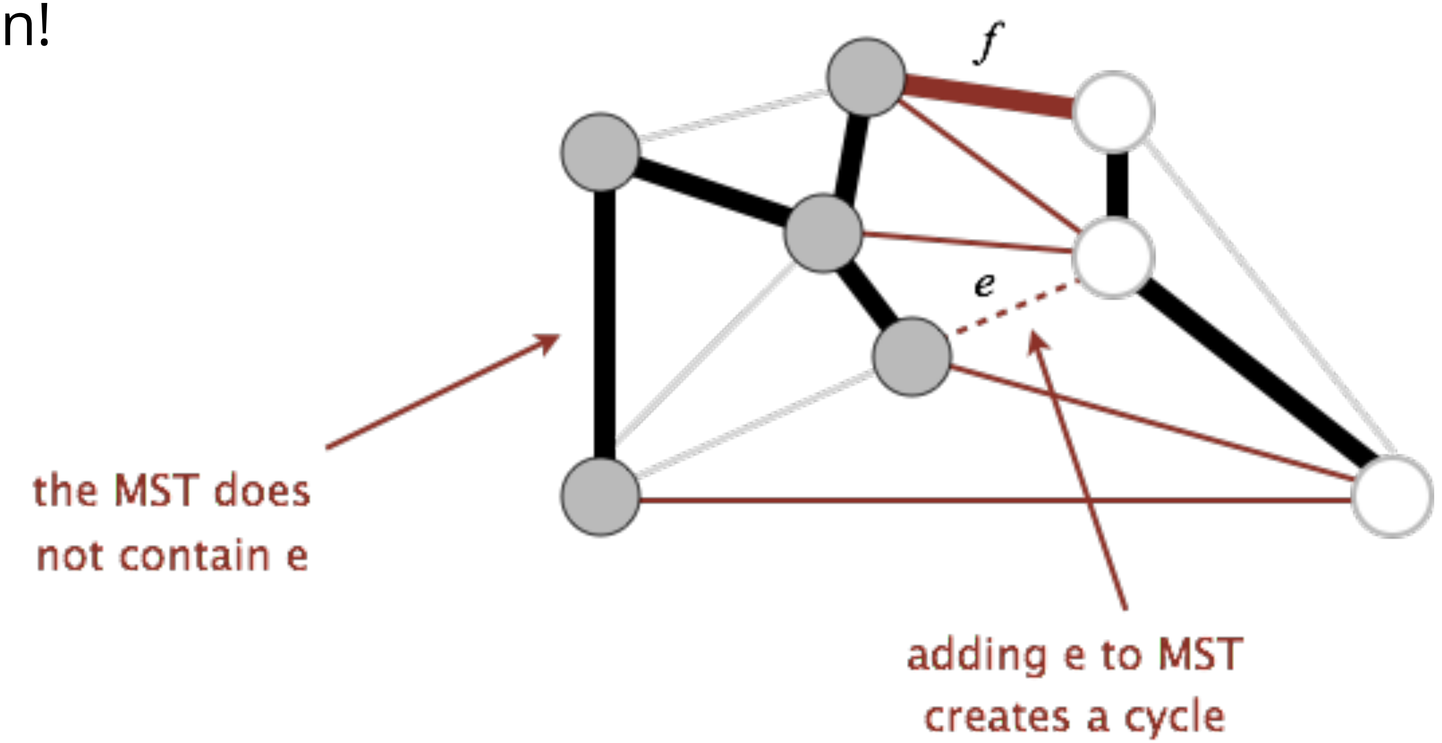



```
0-7   0.16
2-3   0.17
1-7   0.19
0-2   0.26
5-7   0.28
1-3   0.29
1-5   0.32
2-7   0.34
4-5   0.35
1-2   0.36
4-7   0.37
0-4   0.38
6-2   0.40
3-6   0.52
6-0   0.58
6-4   0.93
```

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 temporary cut that has no crossing edges in the being built MST.
- Add smallest crossing edge to the MST.
- Repeat until 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.
- Prim’s and Kruskal’s algorithms are two ways to do it.

Prim’s algorithm

Prim’s algorithm overview

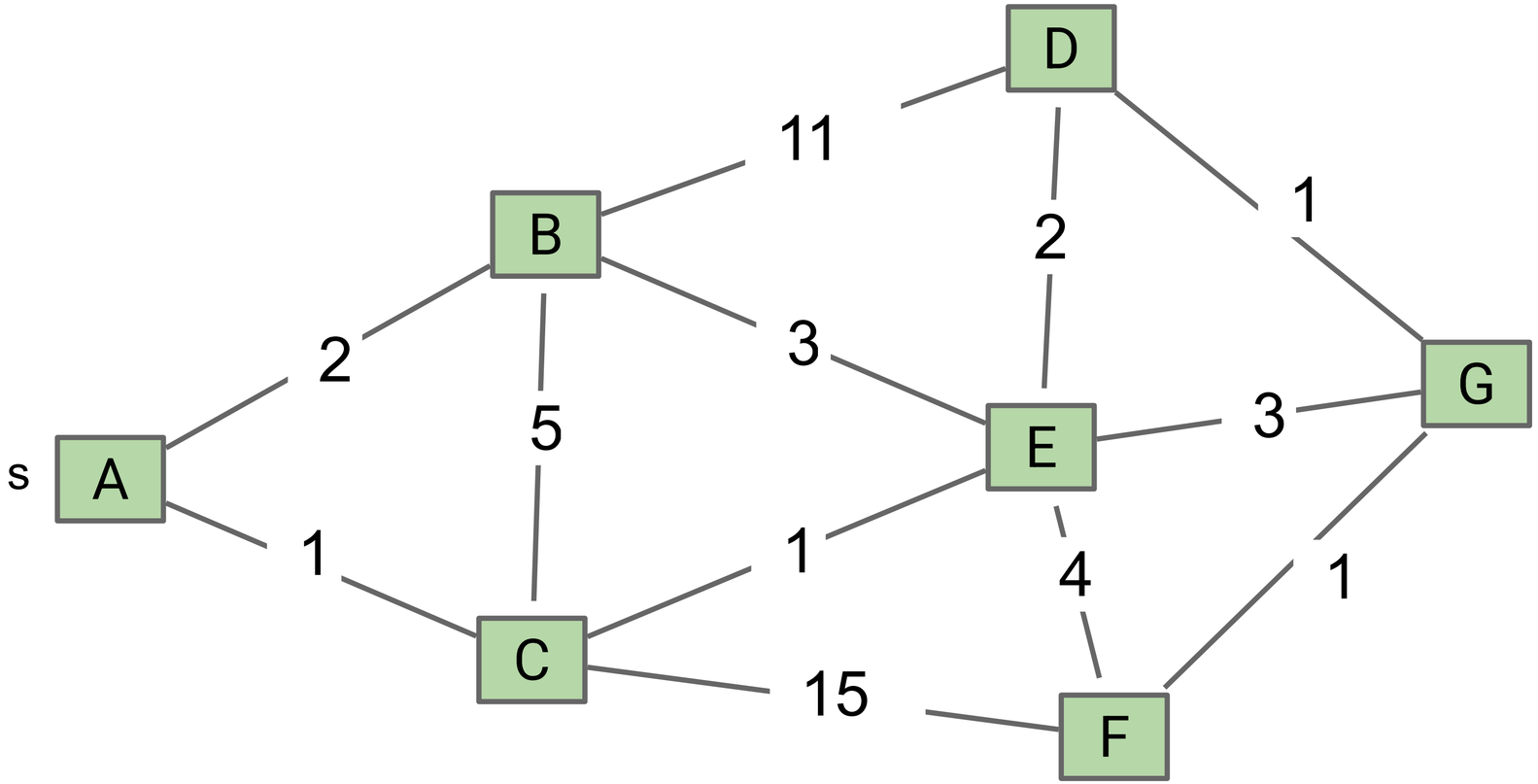
- Start with a random vertex and **greedily** grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until $|V| - 1$ edges.

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	-

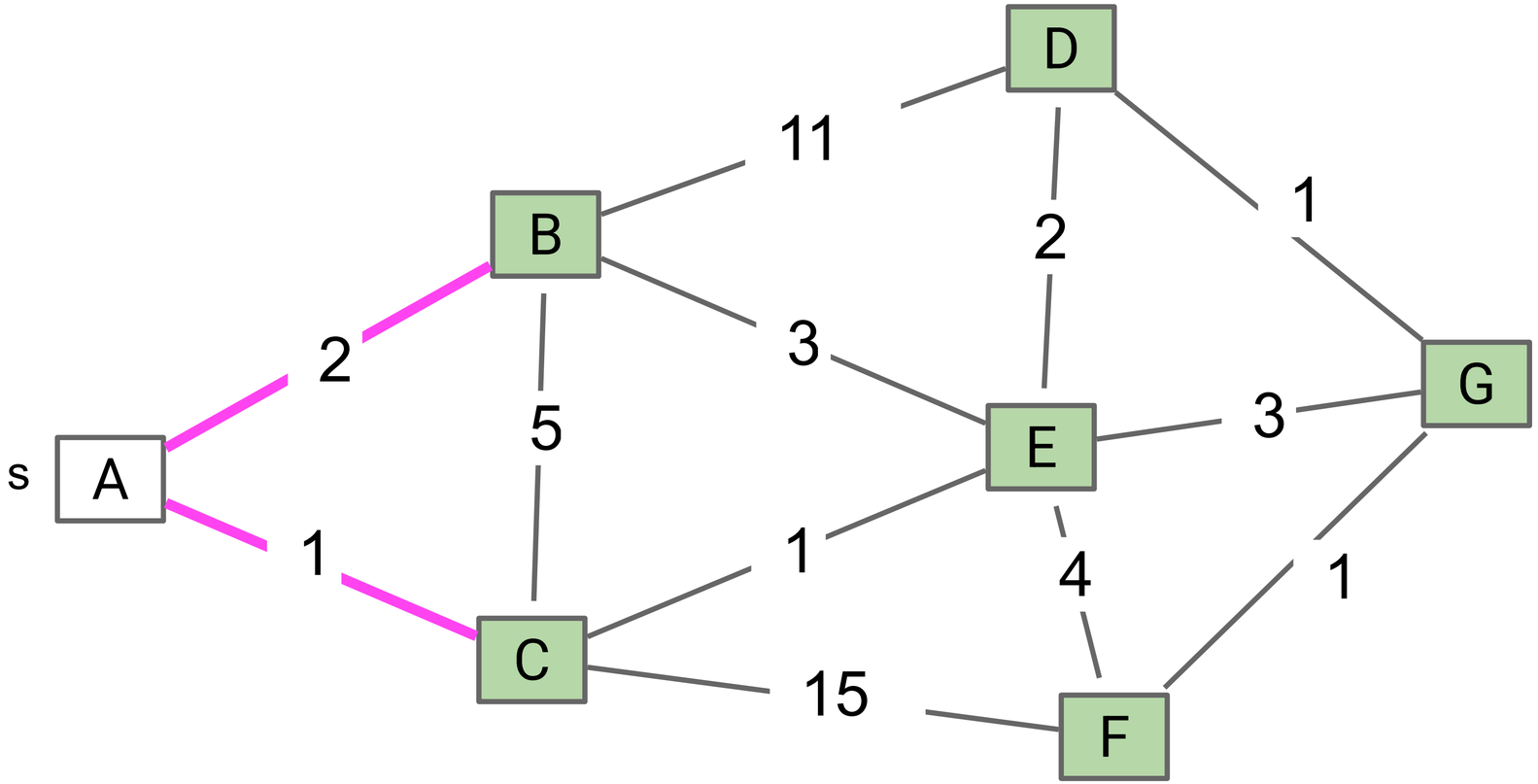


Prim's Demo (Conceptual)

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	-

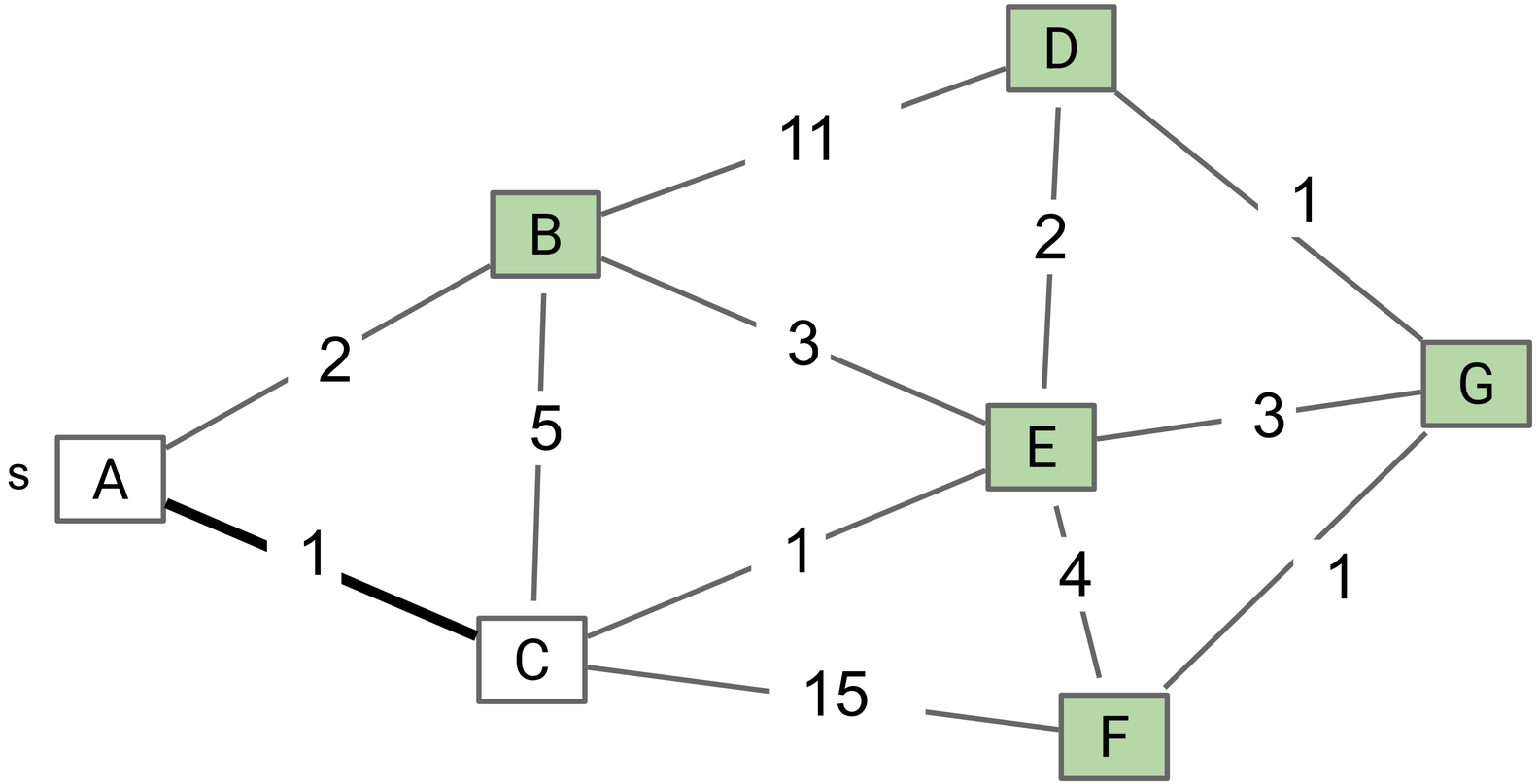


Prim's Demo (Conceptual)

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	-

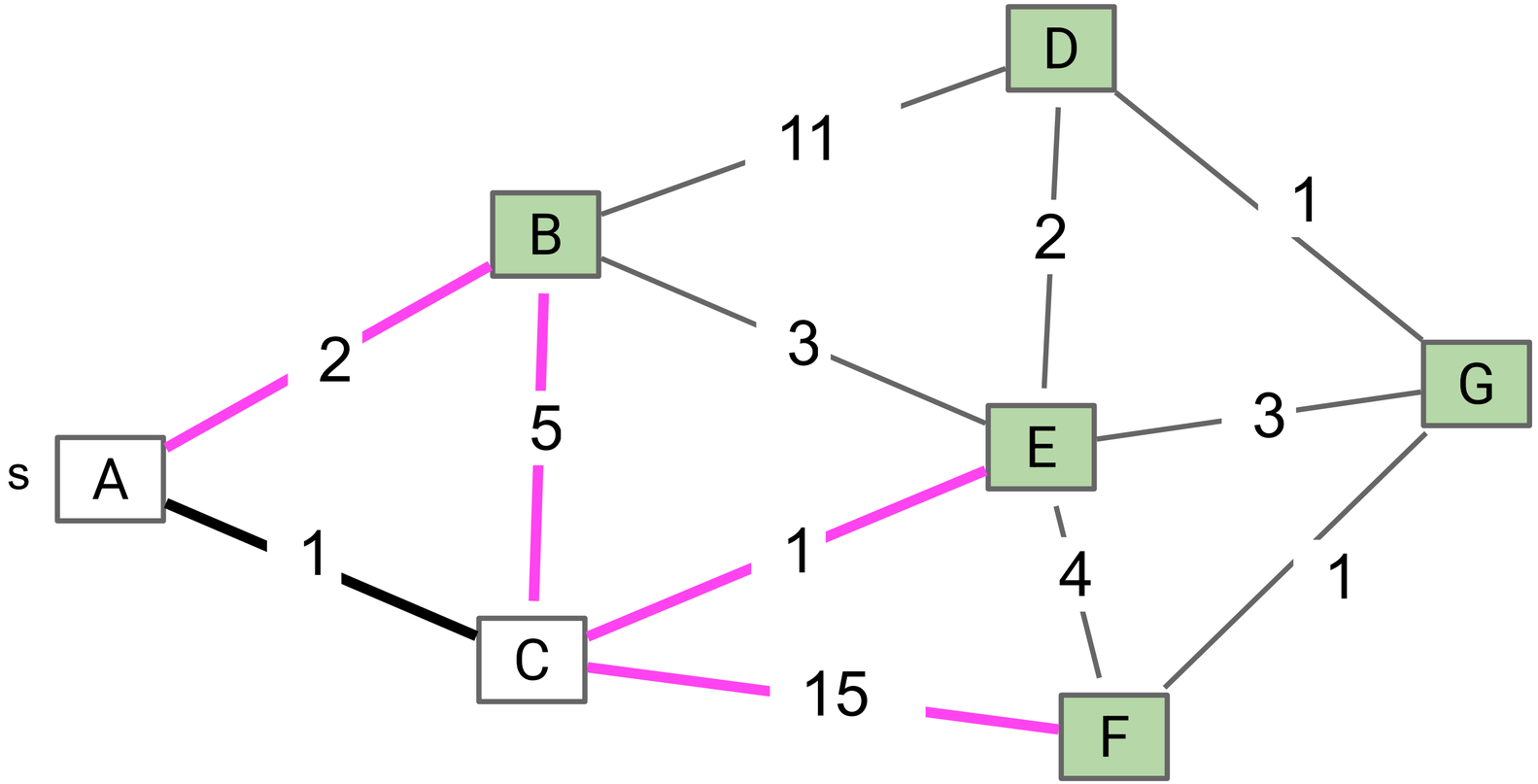


Prim's Demo (Conceptual)

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	-

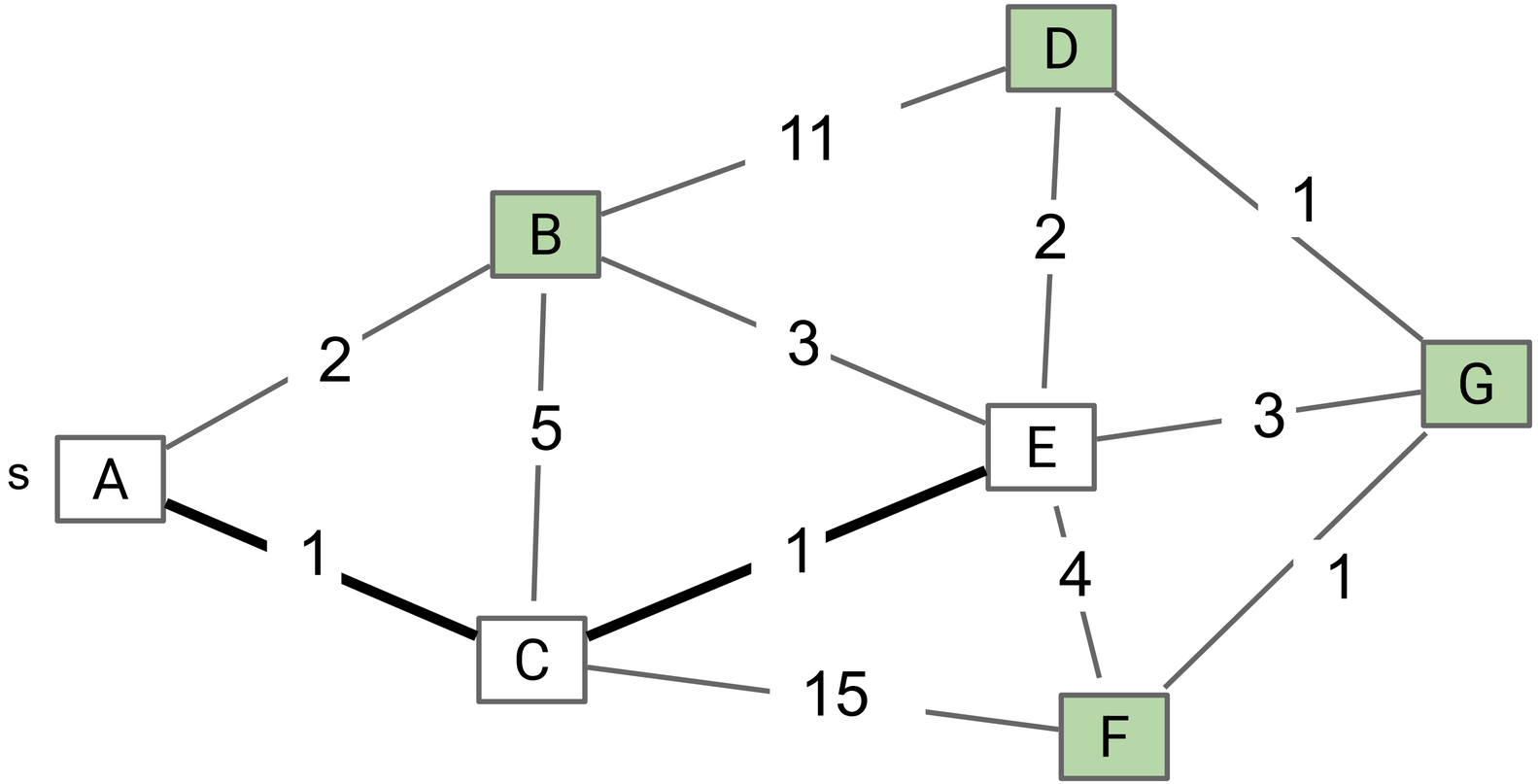


Prim's Demo (Conceptual)

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	-

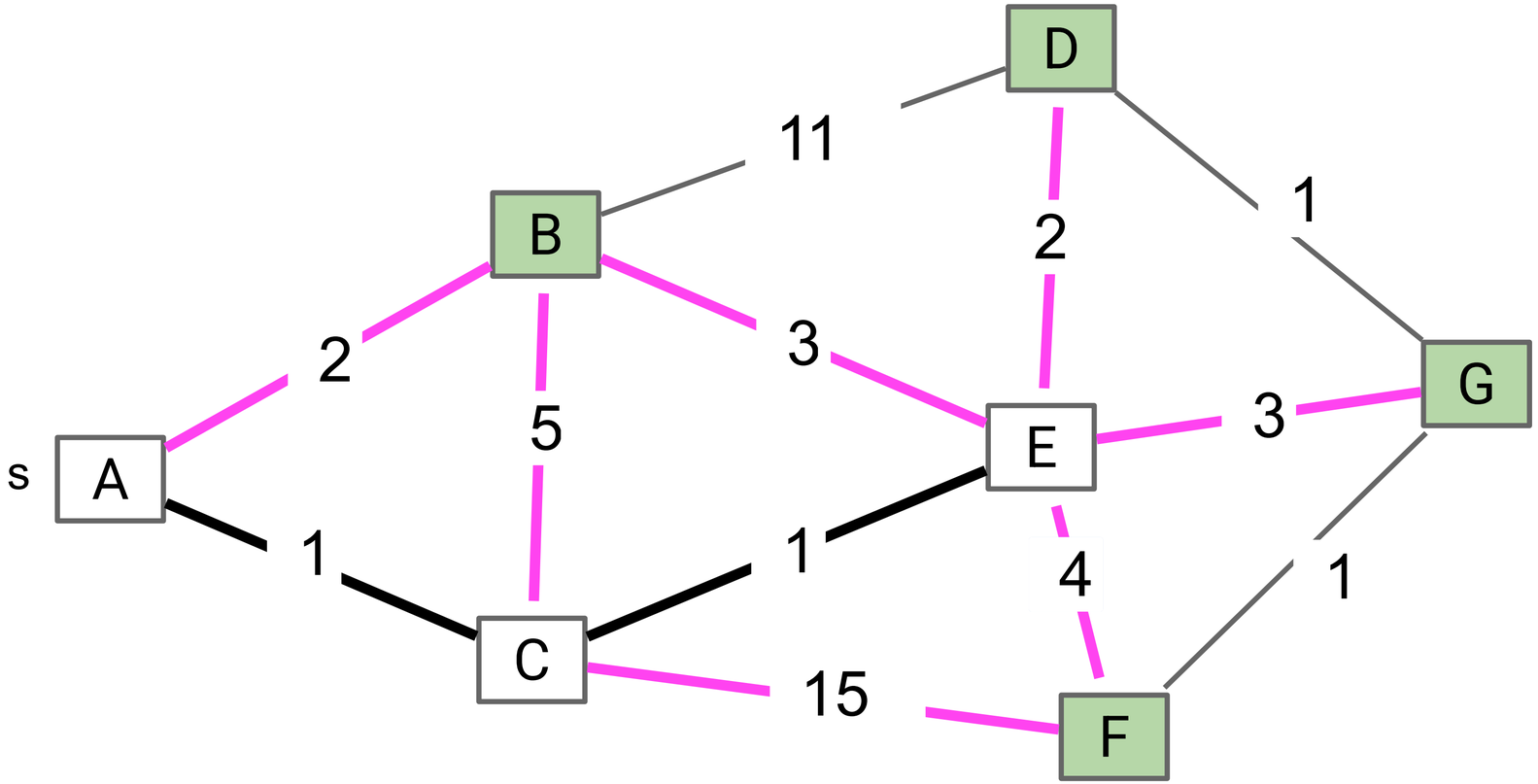


Prim's Demo (Conceptual)

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	-



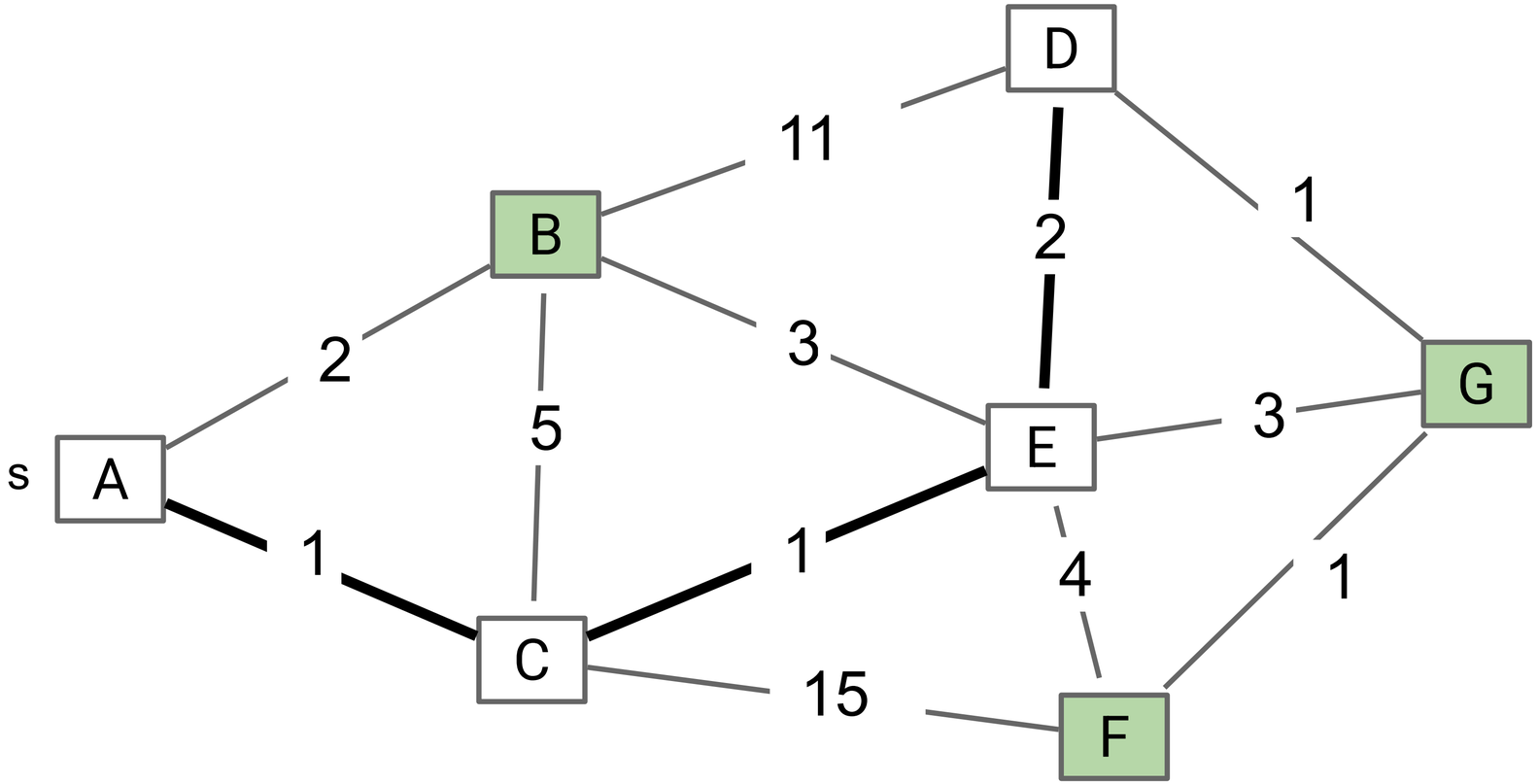
Which edge is added next?

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?

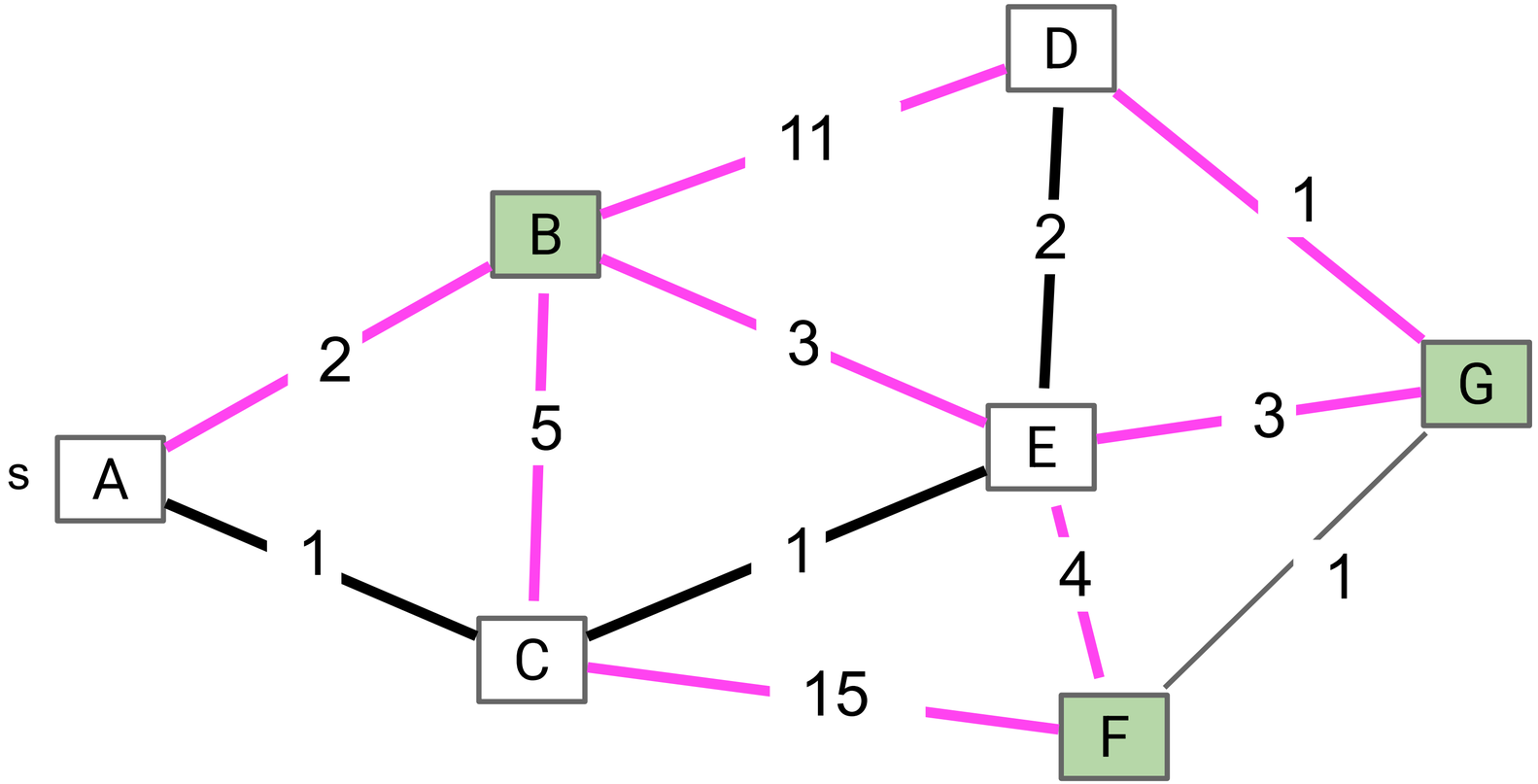
- Either A-B or D-E are guaranteed to work (see exercises for proof)!
- Note: They are not both guaranteed to be in the MST.

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	-

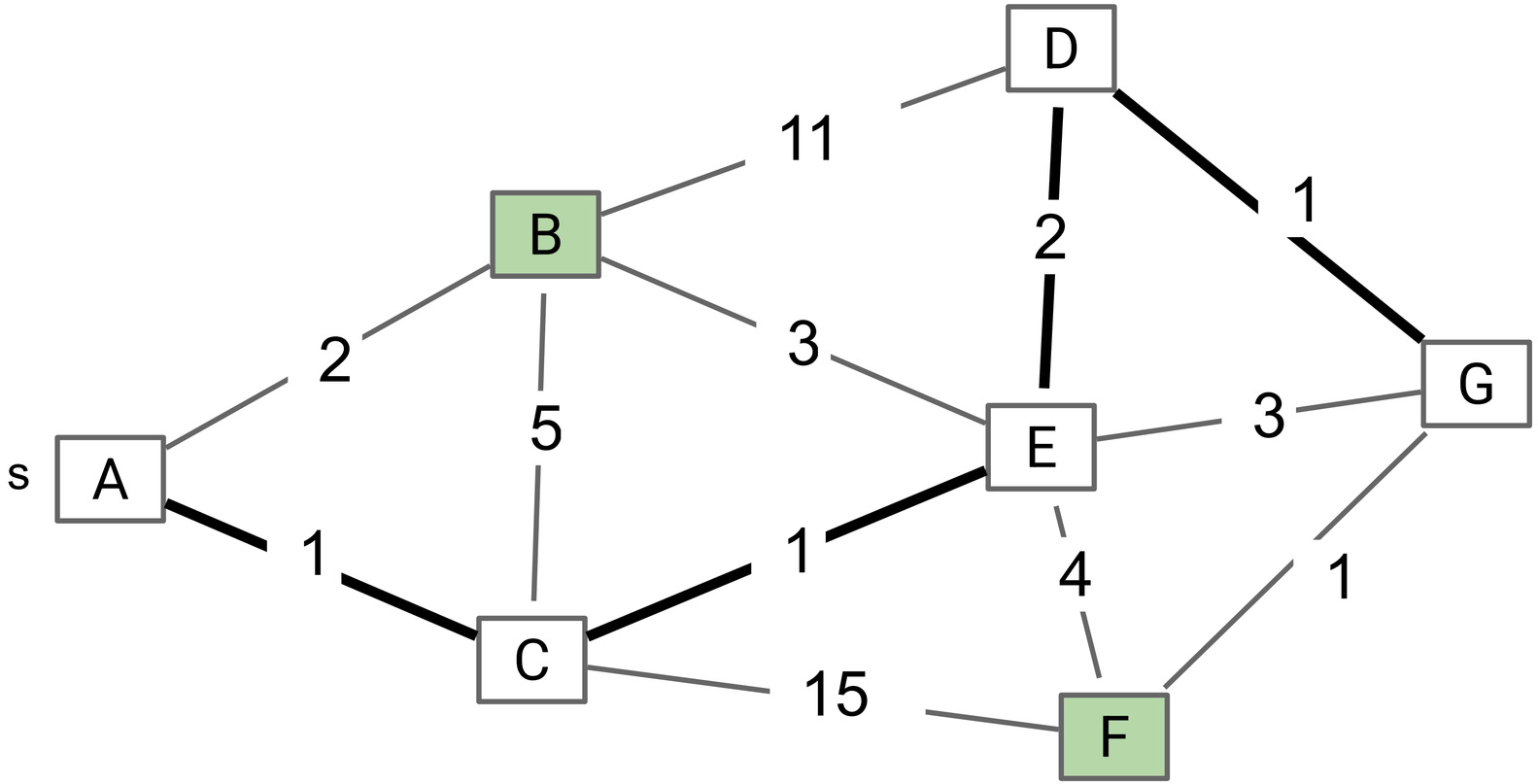


Prim's Demo (Conceptual)

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

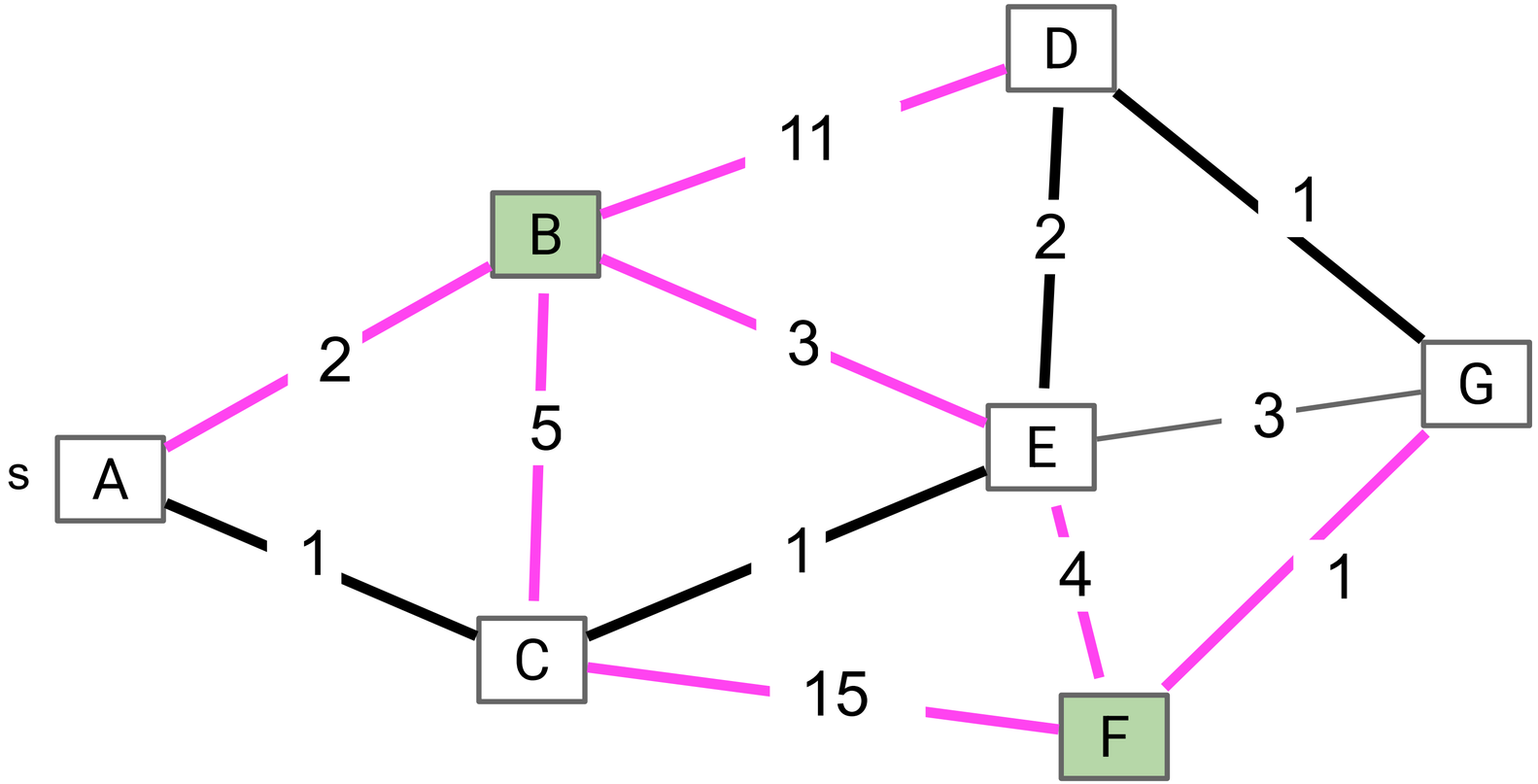


Prim's Demo (Conceptual)

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

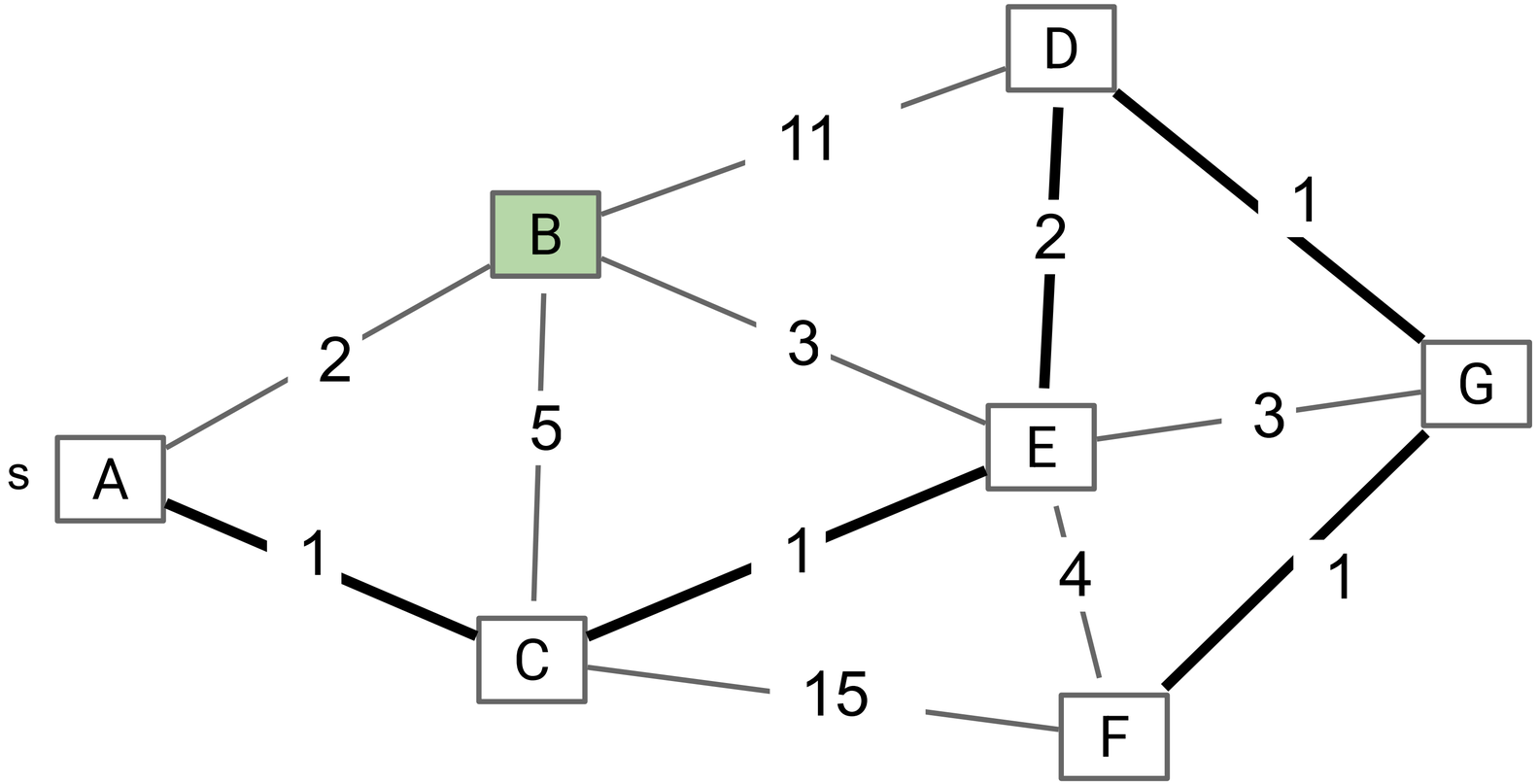


Prim's Demo (Conceptual)

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

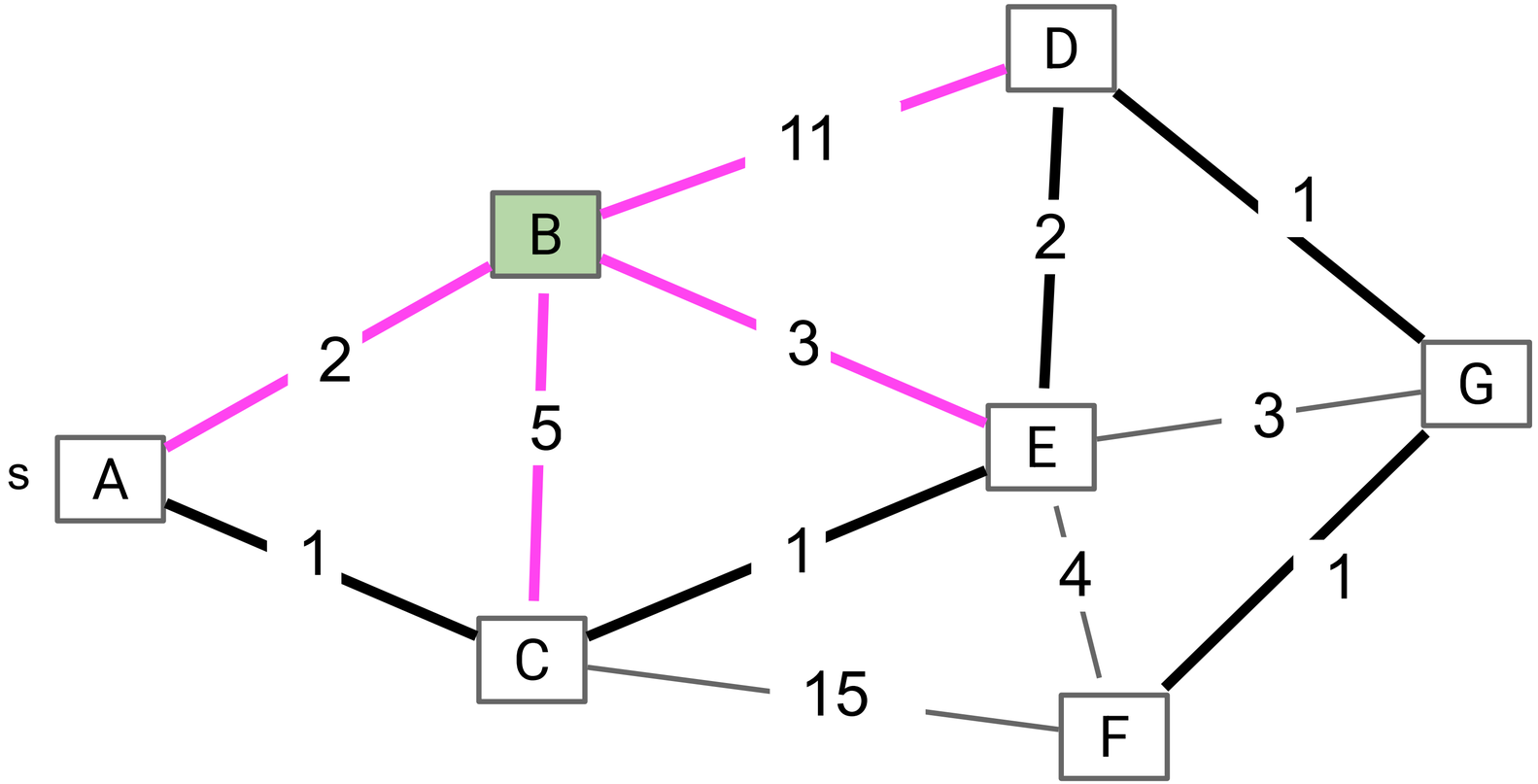


Prim's Demo (Conceptual)

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

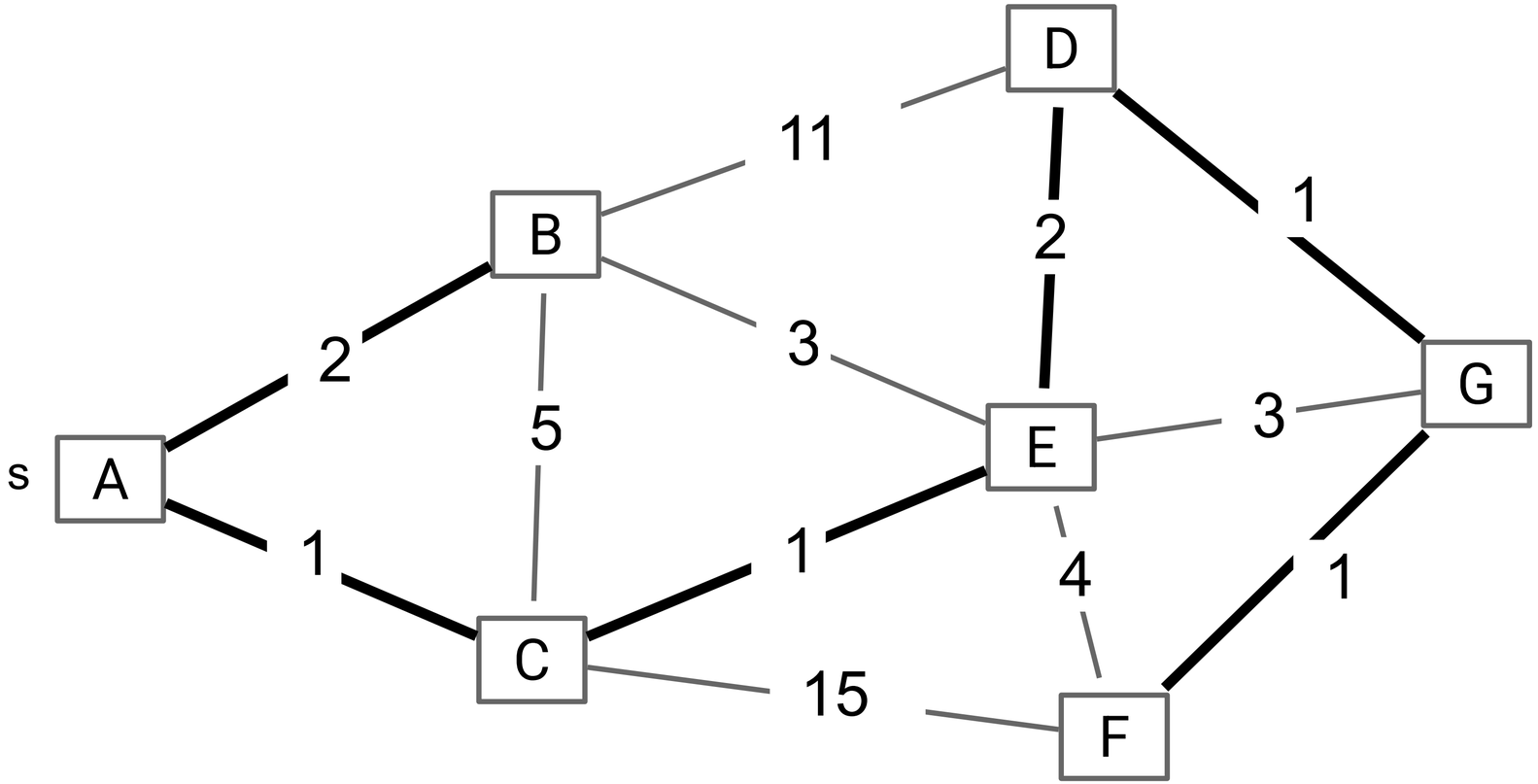


Prim's Demo (Conceptual)

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



Done!

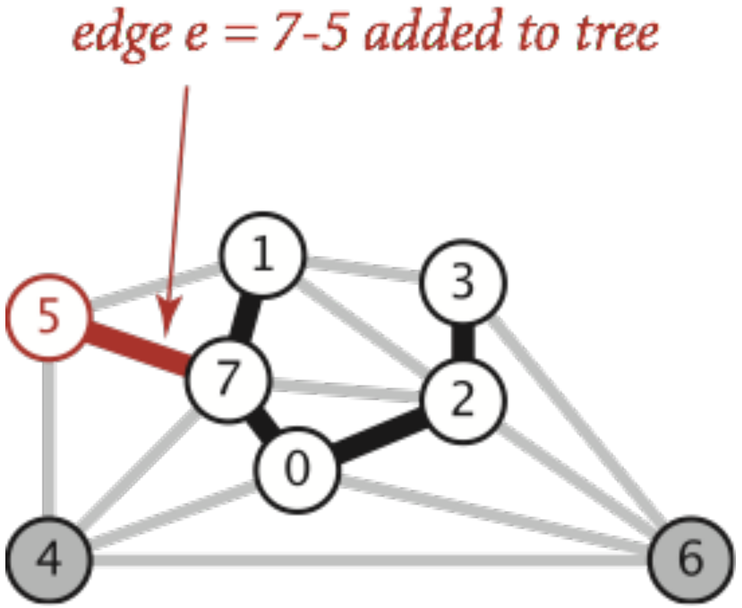
Prim’s Algorithm

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.

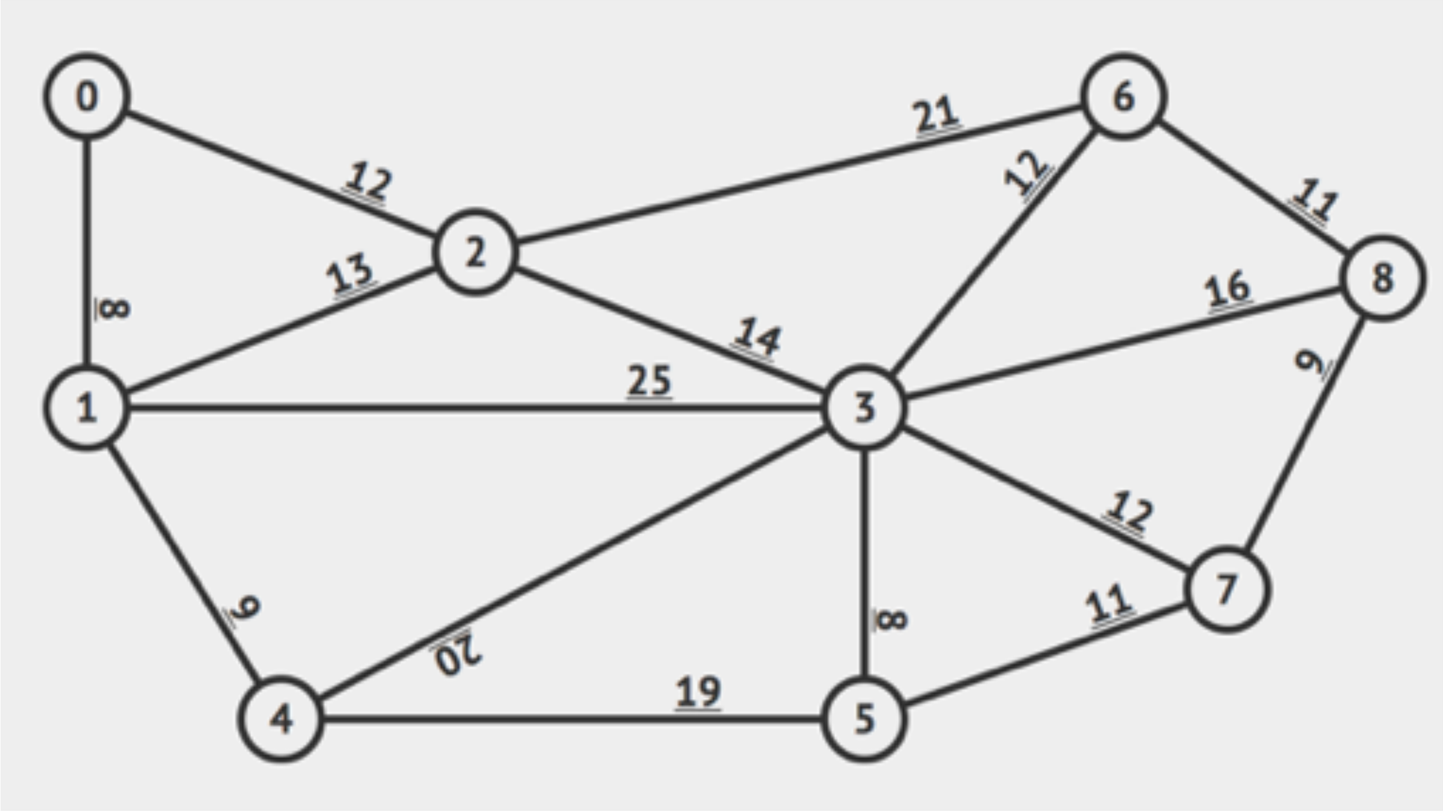
Why does Prim’s work? Special case of our generic algorithm.

- Suppose we add edge e = v->w.
- Side 1 of cut is all vertices connected to start, side 2 is all the others.
- No crossing edge is black (all connected edges on side 1).
- No crossing edge has lower weight (consider in increasing order).

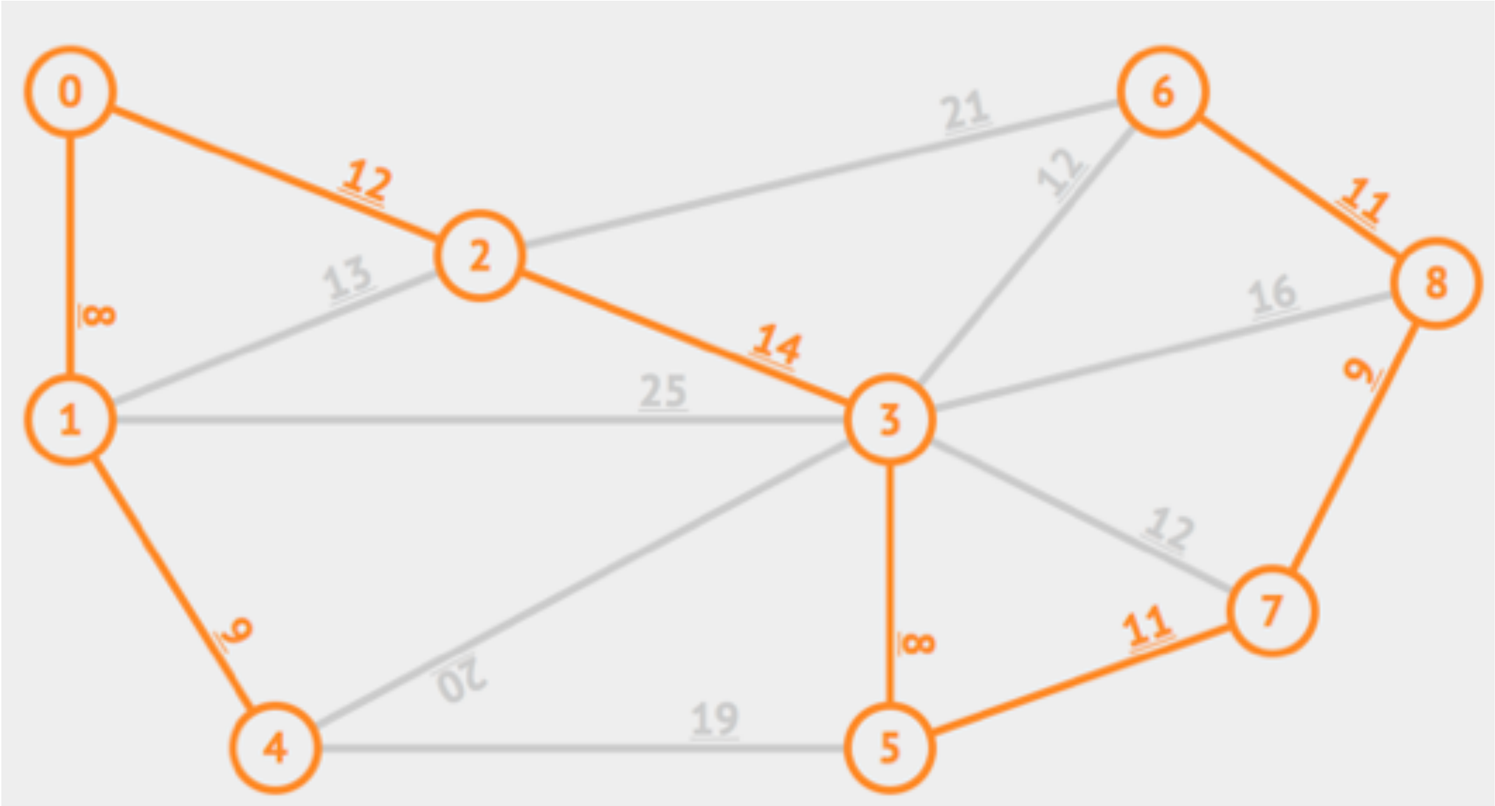


Worksheet time!

- Starting at node 0, run Prim's to find the MST.



Worksheet answer



Prim’s algorithm (optimized)

Prim's Algorithm Implementation

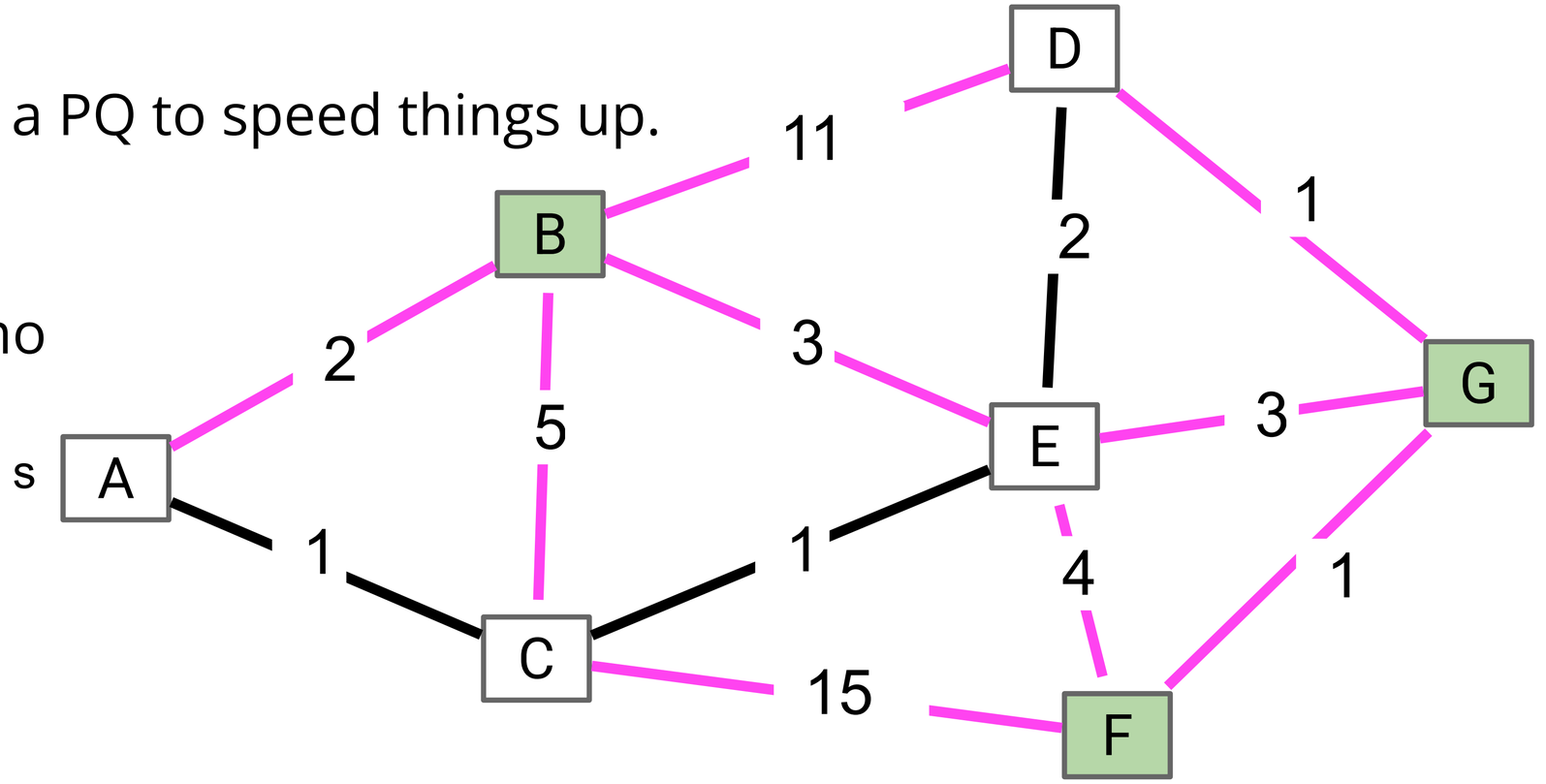
The natural implementation of the conceptual version of Prim's algorithm is highly inefficient.

- Example: Iterating over all pink 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!

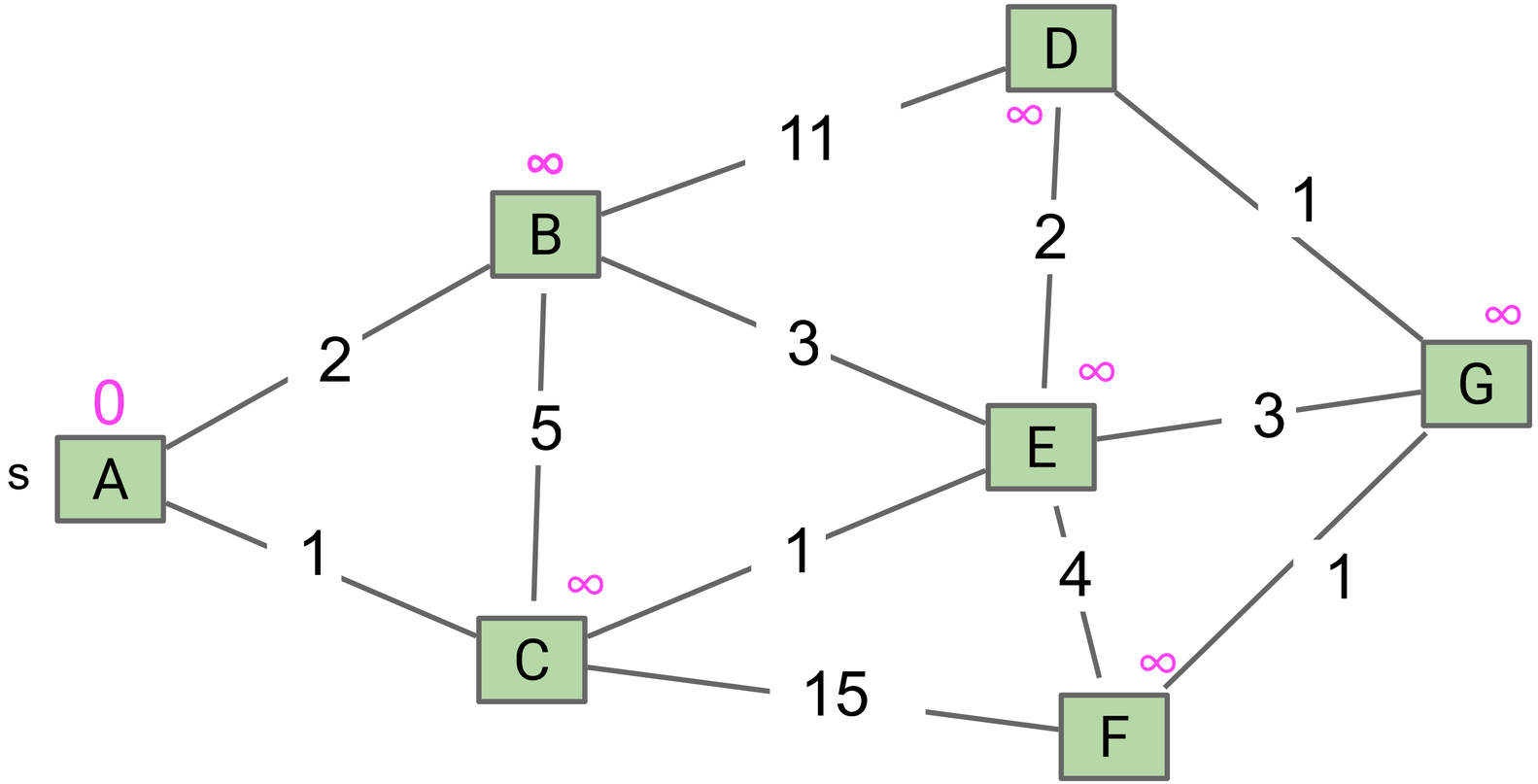


Prim's Demo

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	∞	-



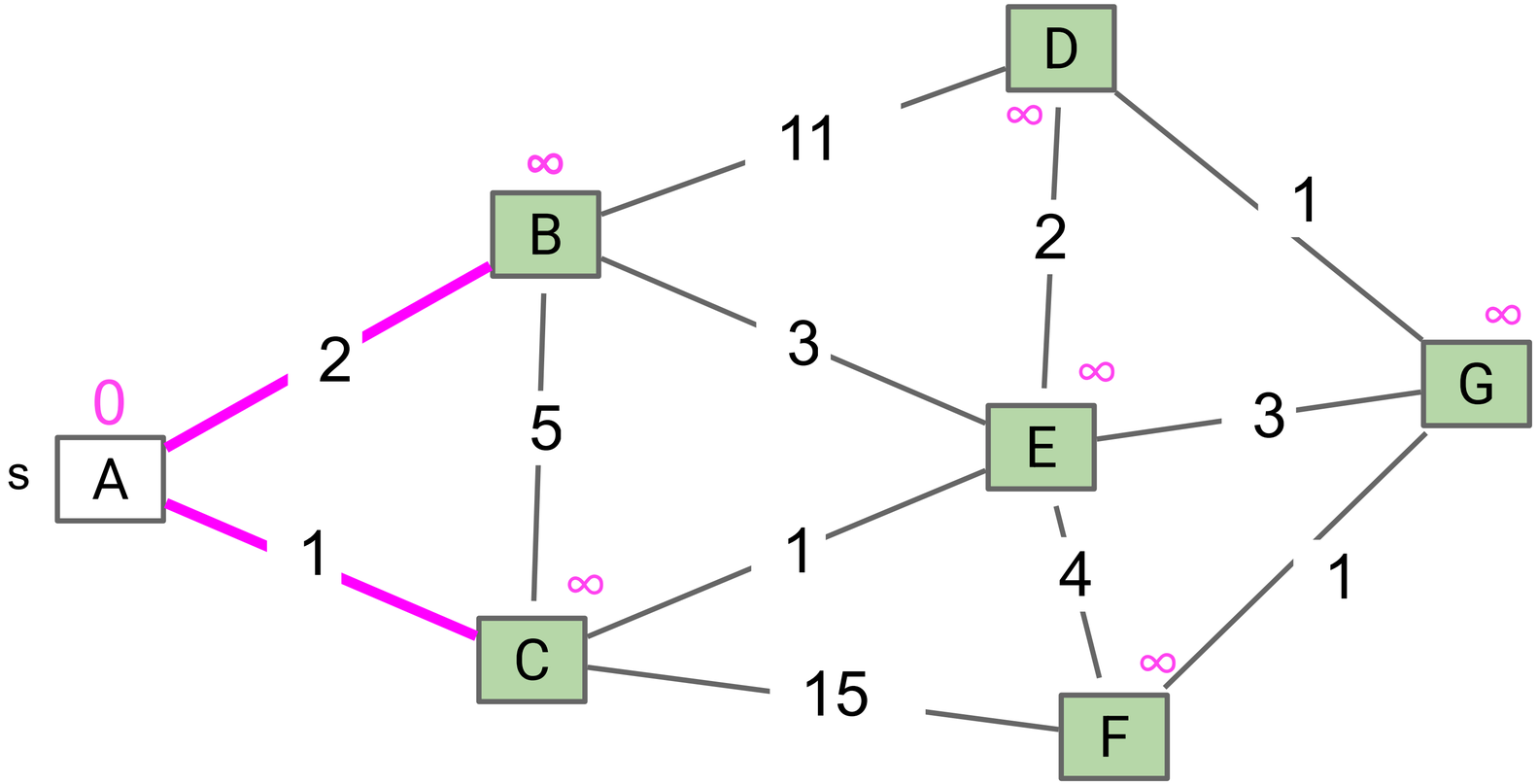
Fringe: [(A: 0), (B: ∞), (C: ∞), (D: ∞), (E: ∞), (F: ∞), (G: ∞)]

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	∞	-



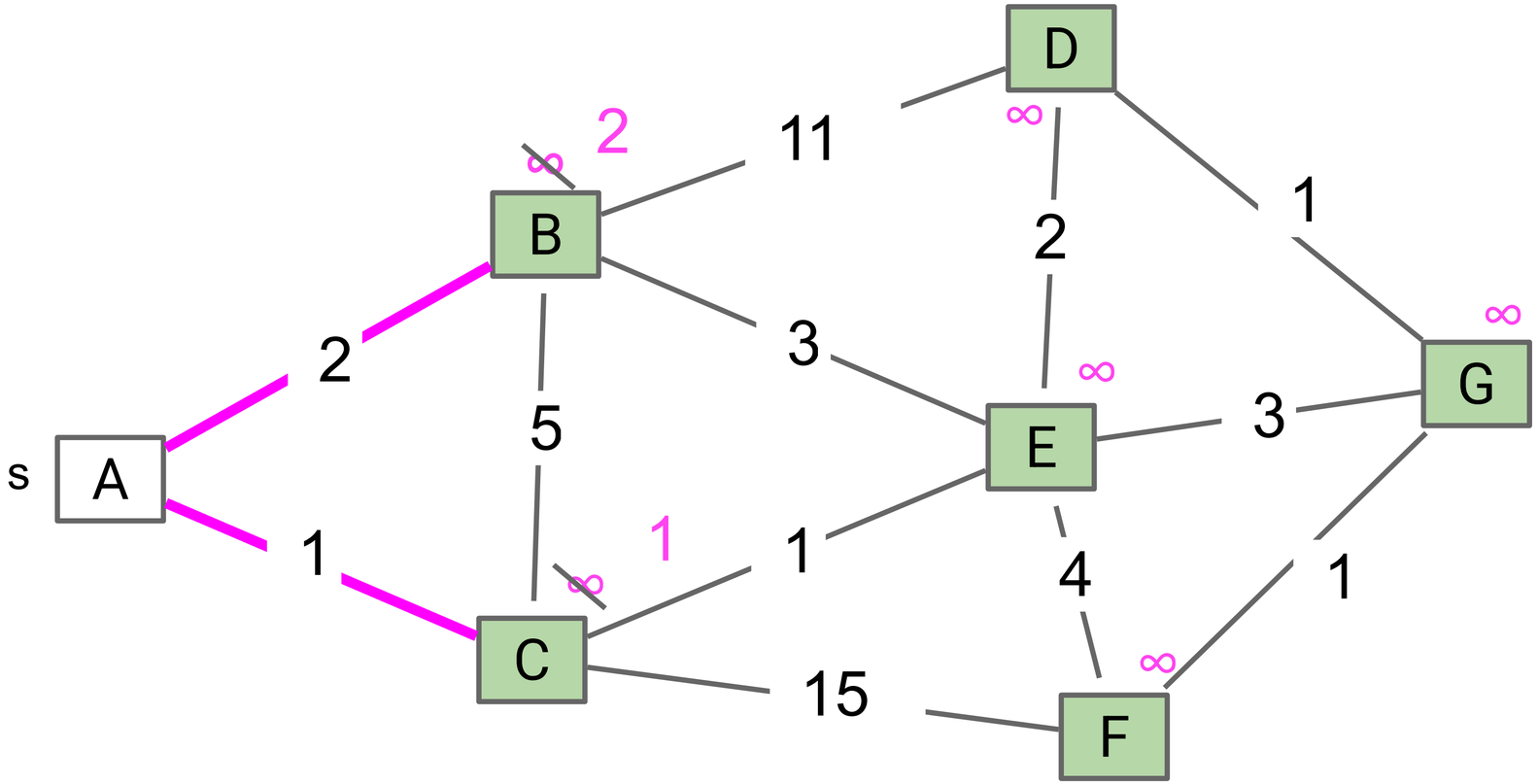
Fringe: [(B: ∞), (C: ∞), (D: ∞), (E: ∞), (F: ∞), (G: ∞)]

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	∞	–



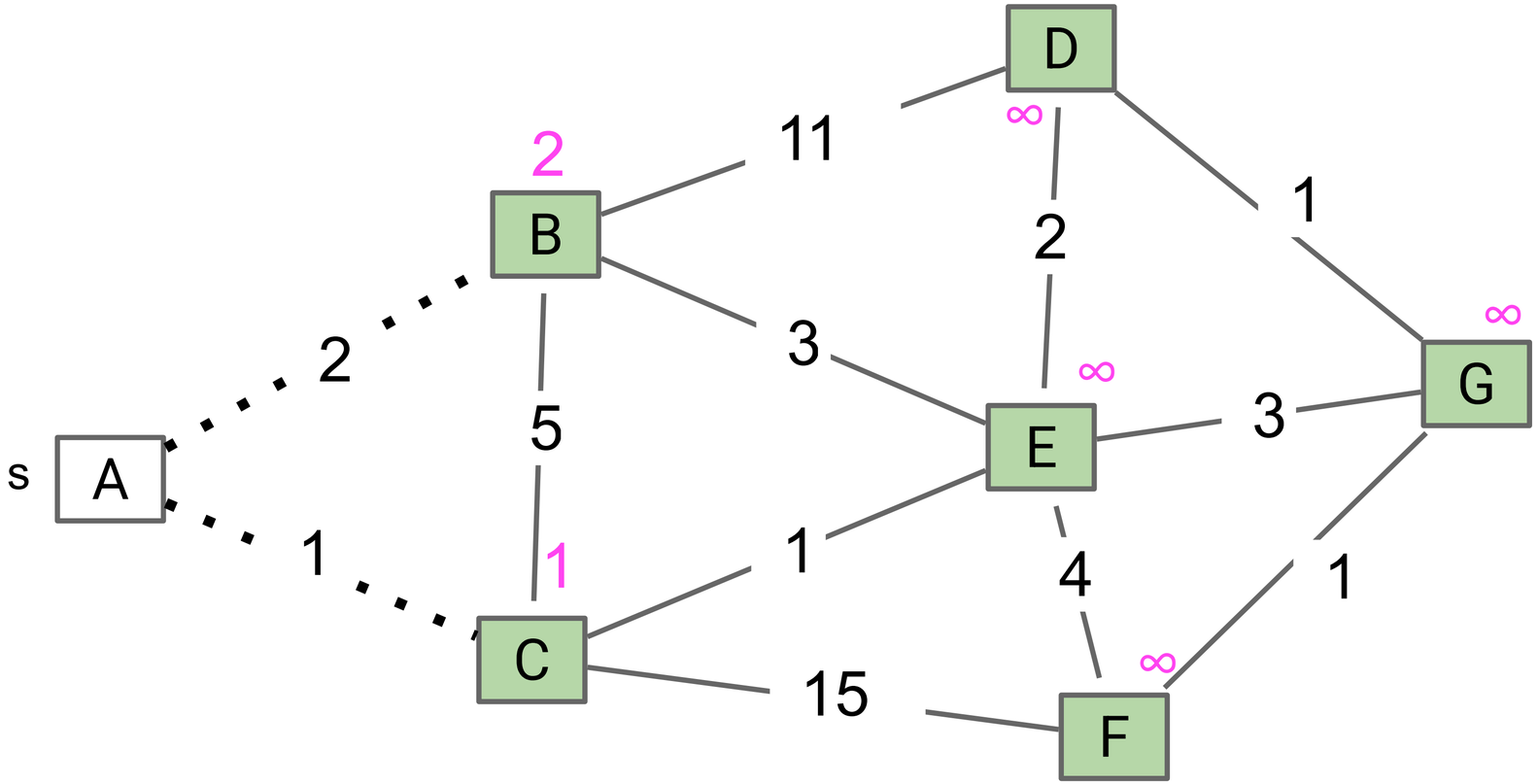
Fringe: [(C: 1), (B: 2), (D: ∞), (E: ∞), (F: ∞), (G: ∞)]

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	∞	-



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

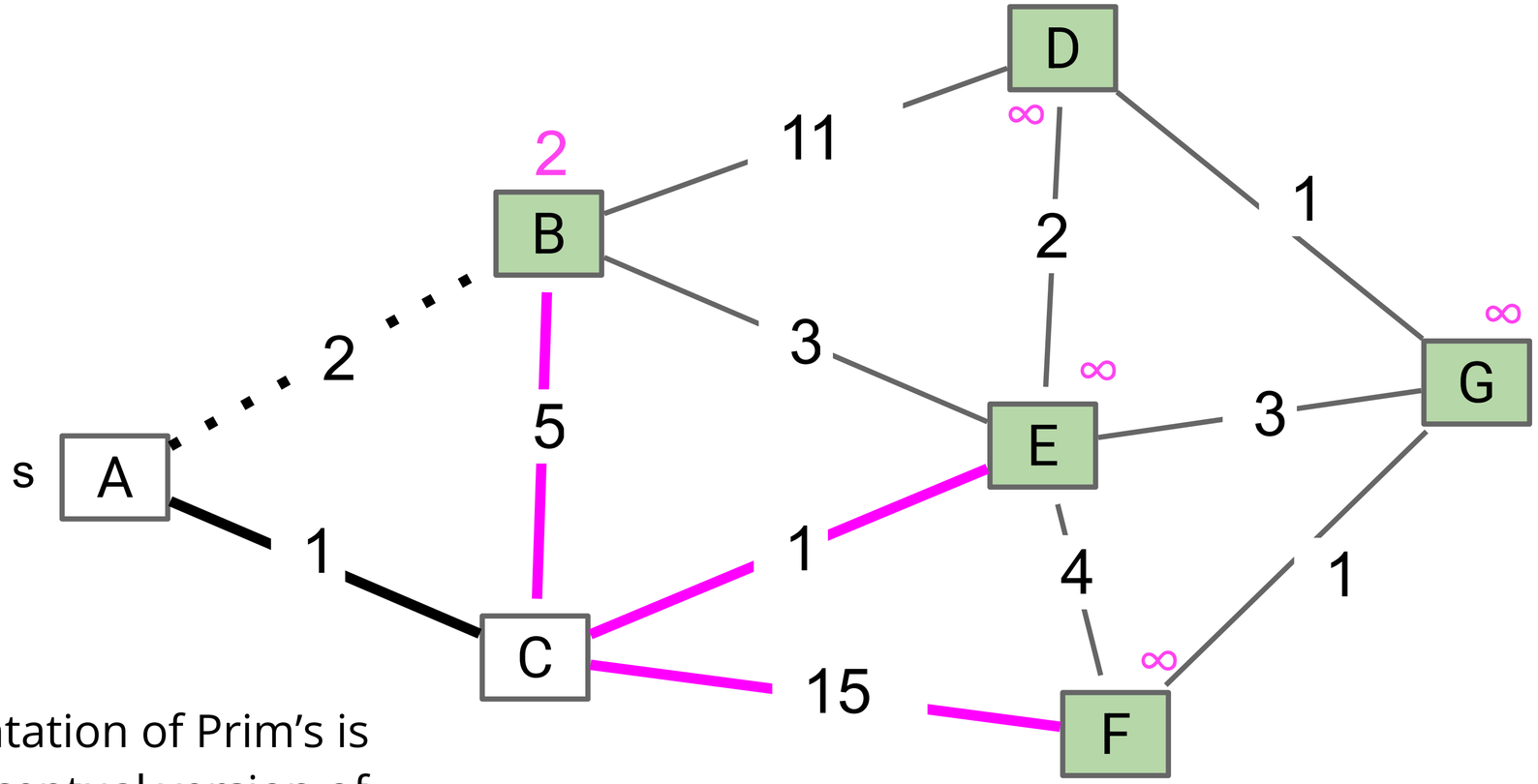
Prim's Demo

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.

Removed distTo = part of MST

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.

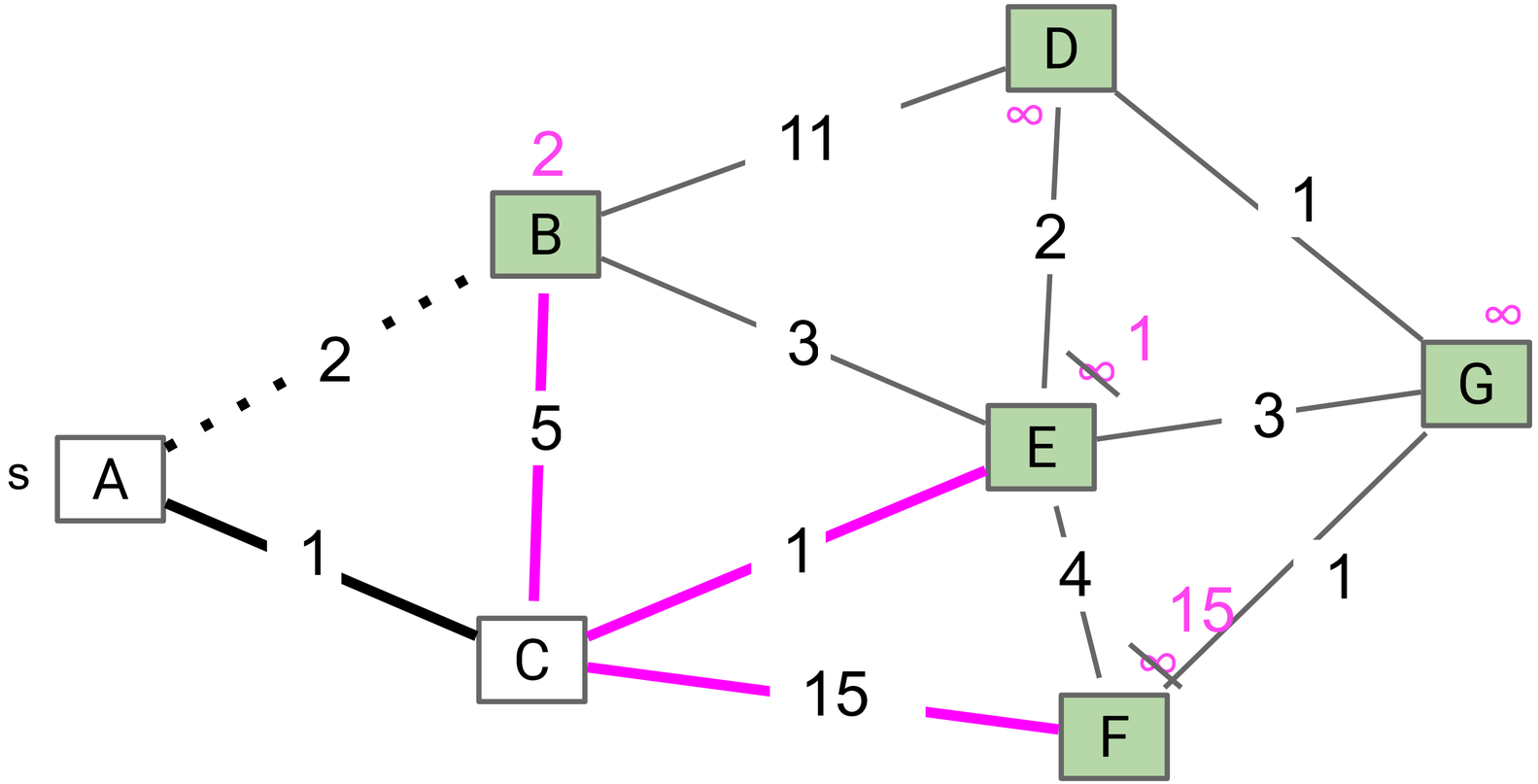
`Fringe: [(B: 2), (D: ∞), (E: ∞), (F: ∞), (G: ∞)]`

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	∞	-



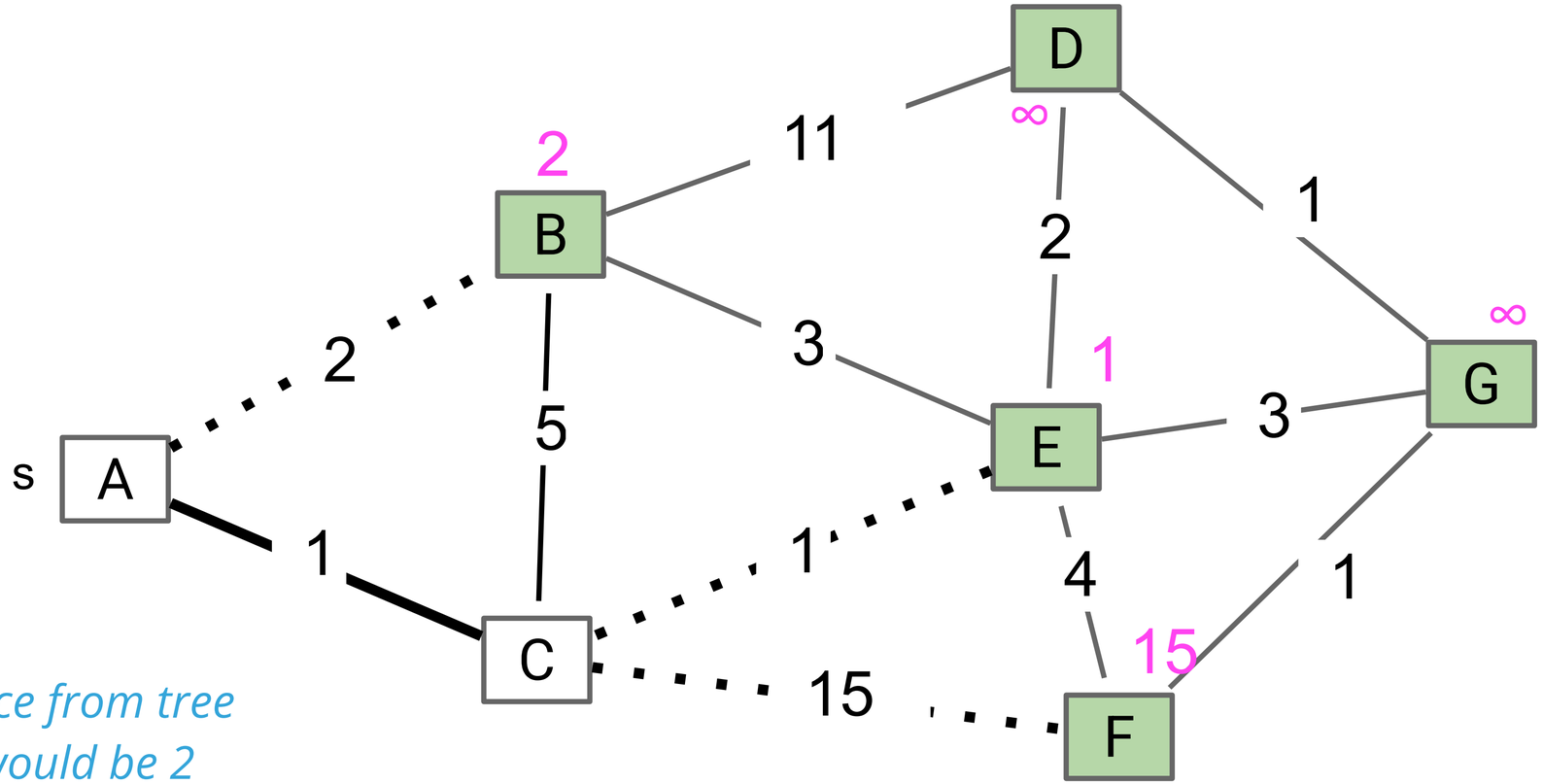
Fringe: [(E: 1), (B: 2), (F: 15), (D: ∞), (G: ∞)]

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	∞	-



Note: unlike Dijkstra's, we consider distance from tree (e.g., E is 1 away from the tree via C, but would be 2 away from the source A in Dijkstra's)

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

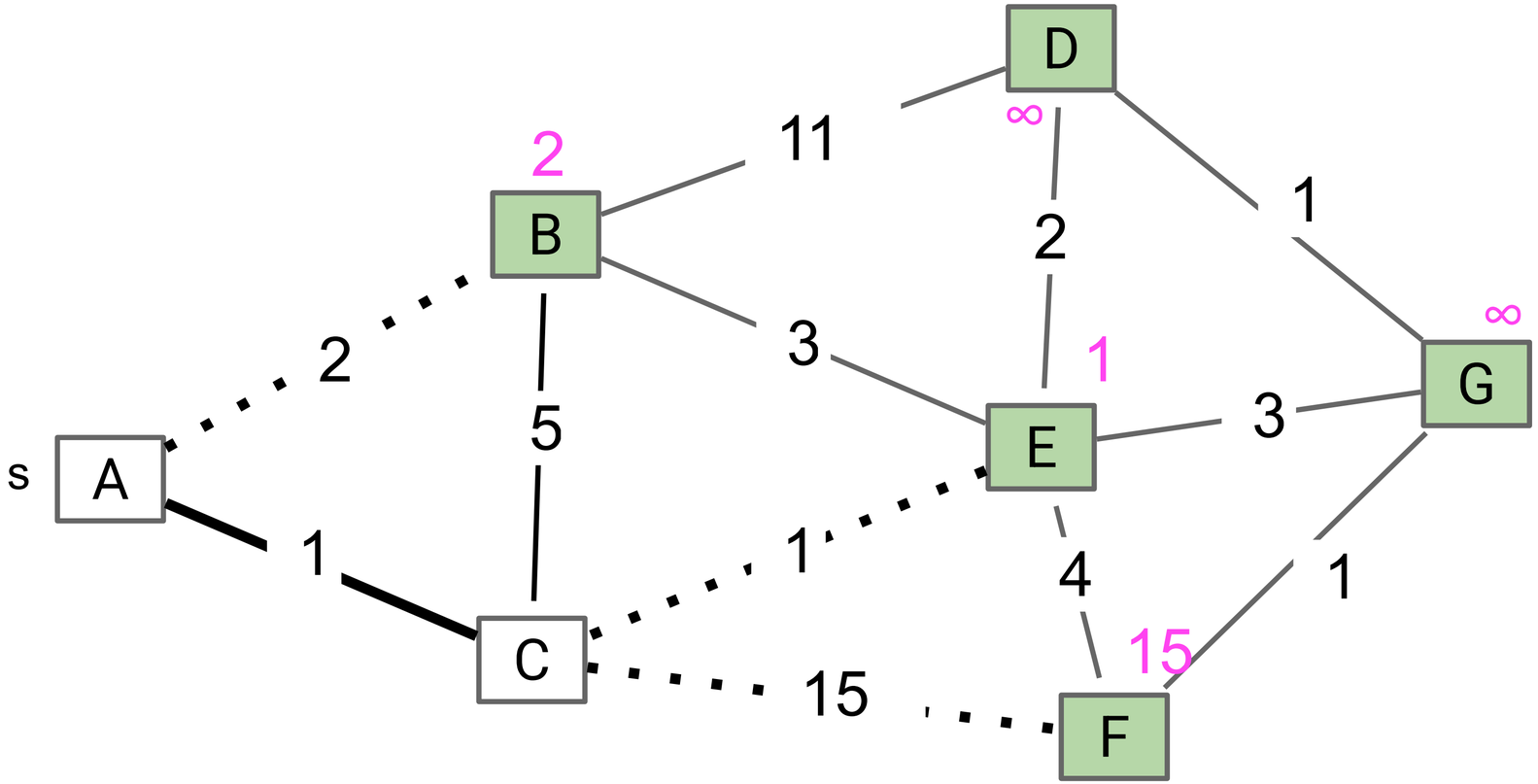
Worksheet time!

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.

- We just relaxed C's edges. Show distTo, edgeTo, and fringe after the next relaxation.

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



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

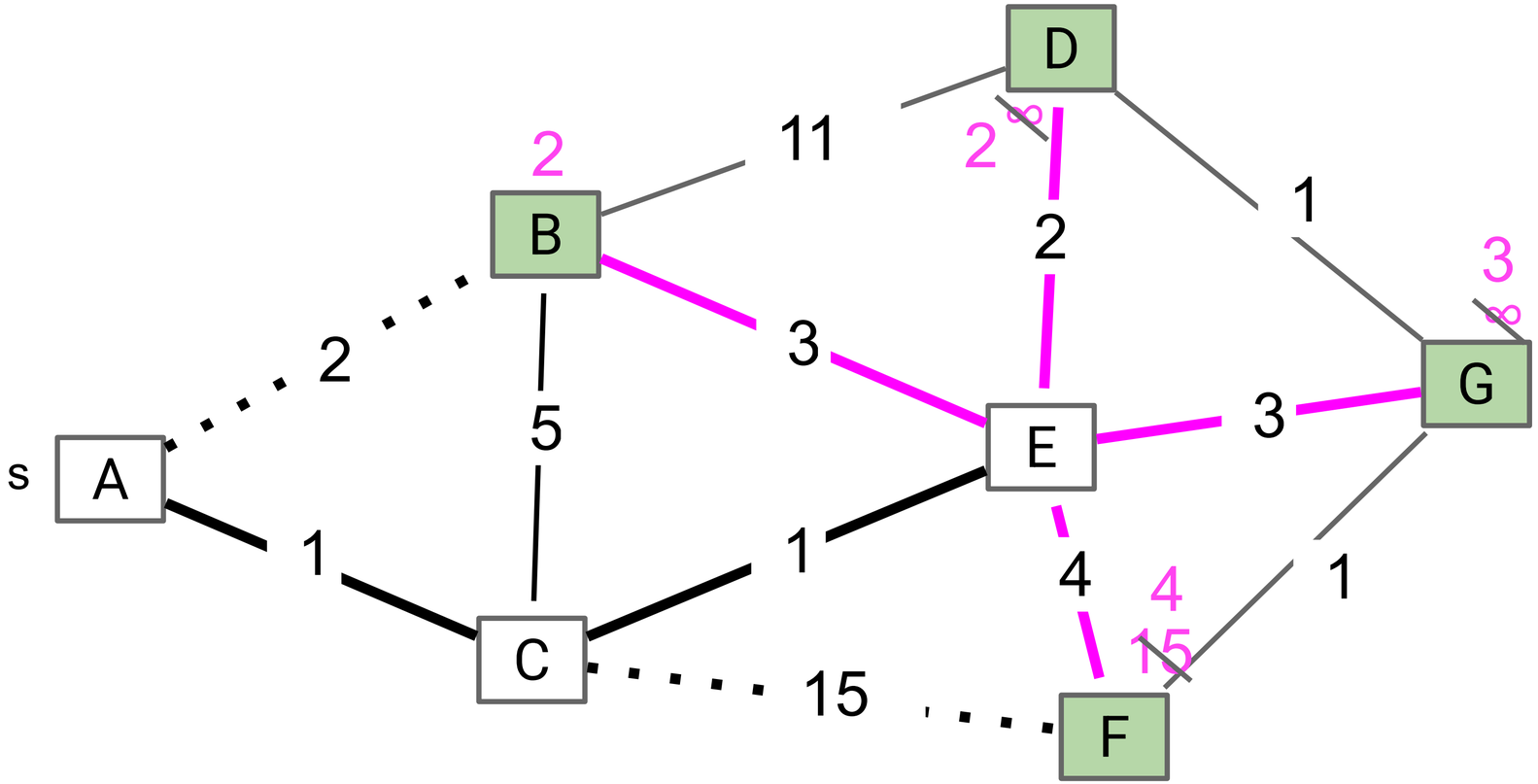
Worksheet answers

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.

- We remove the C->E edge and update D, G, and F. (B not updated)

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



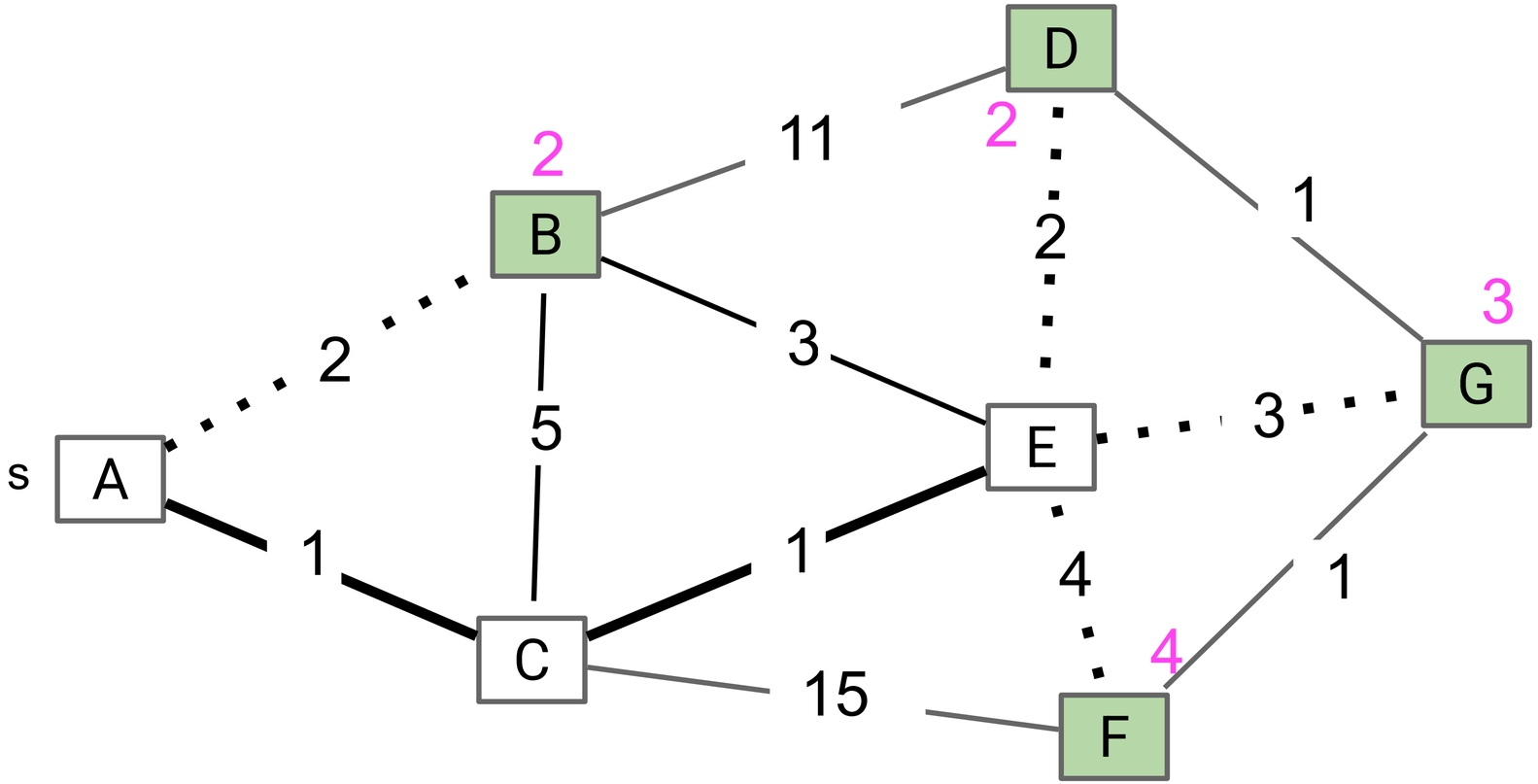
Fringe: [(B: 2), (D: 2), (G: 3), (F: 4)]

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	2	A
C		**A**
D	2	E
E		**C**
F	4	E
G	3	E



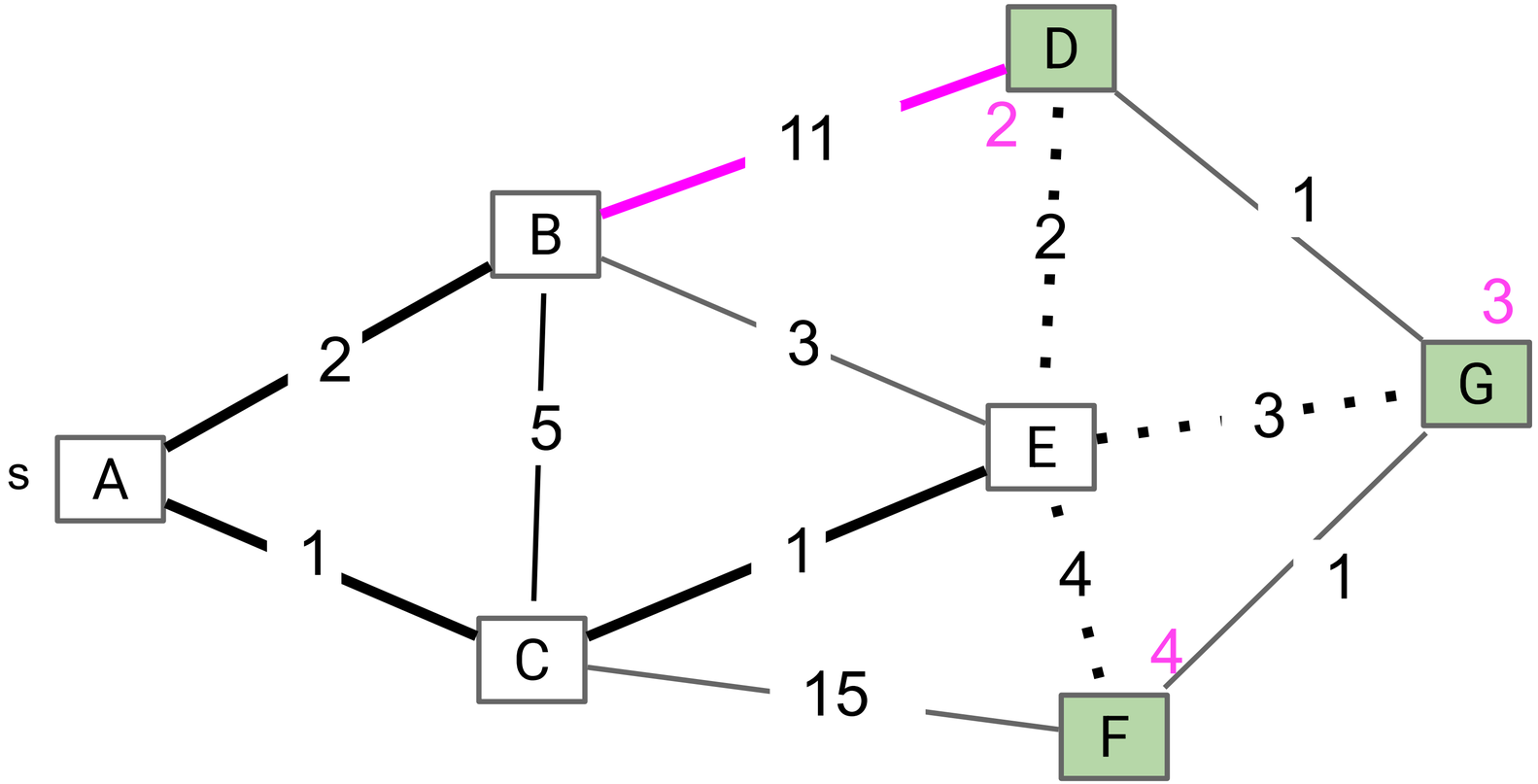
Fringe: [(B: 2), (D: 2), (G: 3), (F: 4)]

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



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

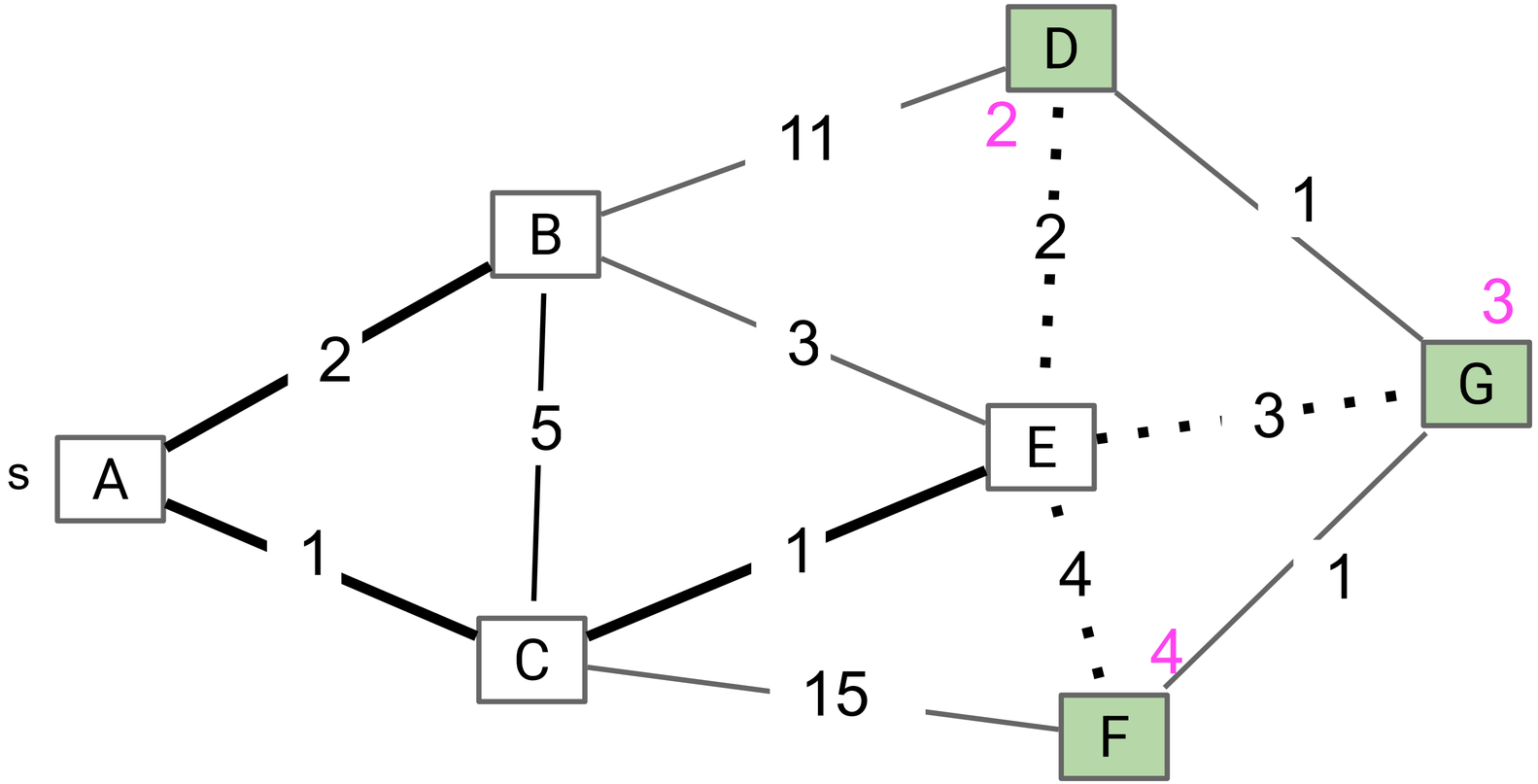
No need to consider B's other edges with weight 5 and 3 since other side is already marked

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



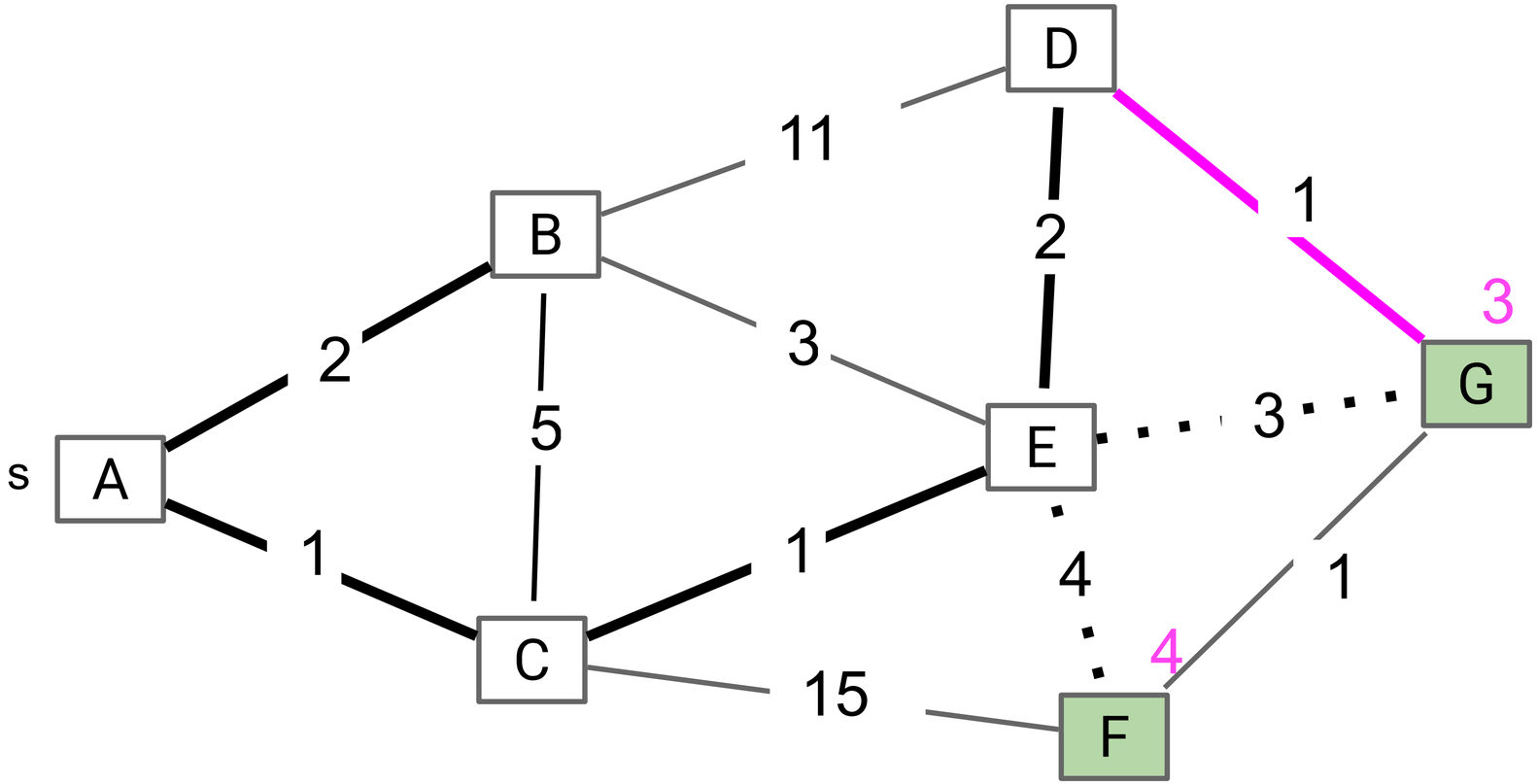
Fringe: [(D: 2), (G: 3), (F: 4)]

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



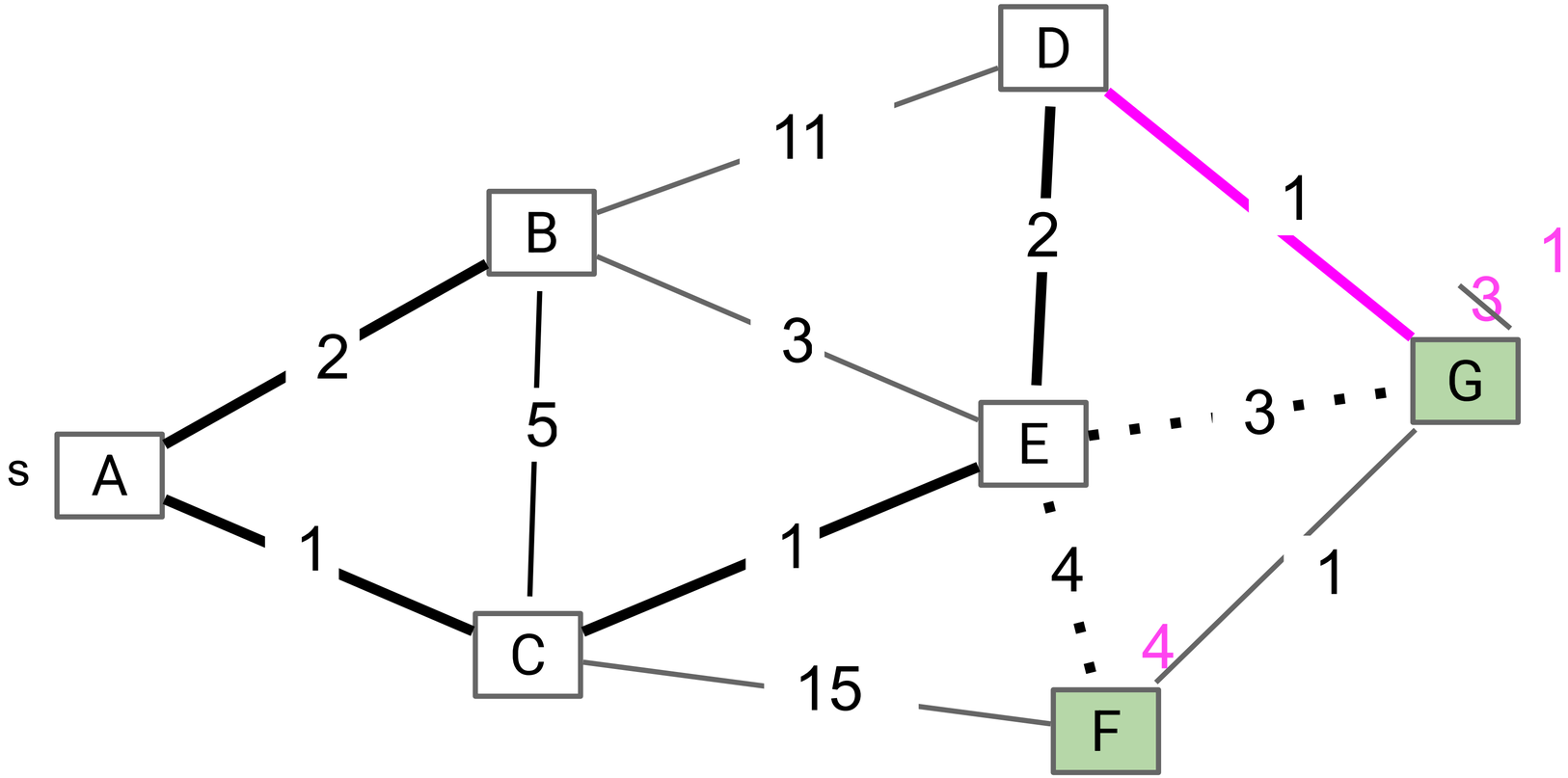
Fringe: [(G: 3), (F: 4)]

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



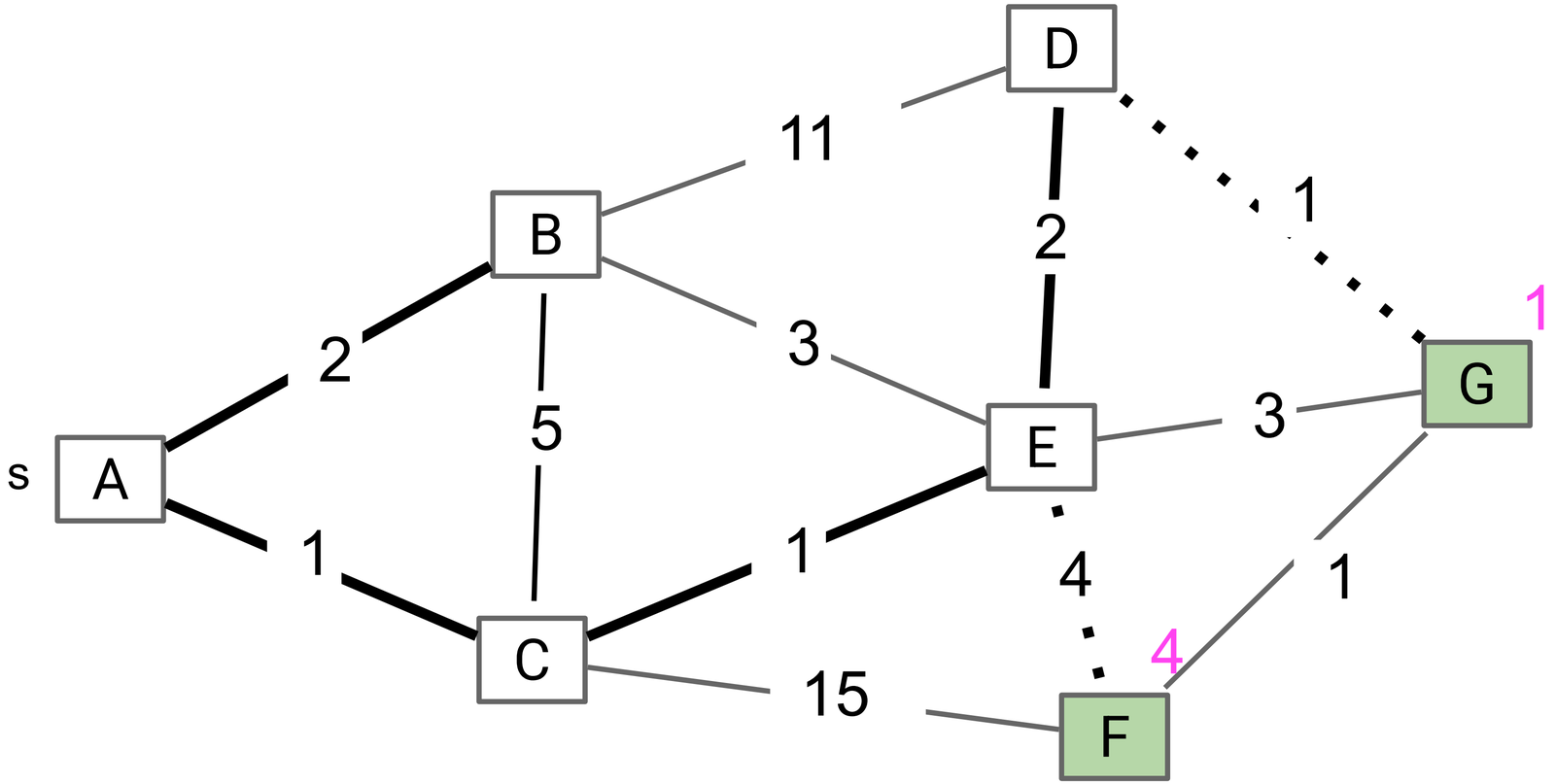
Fringe: [(G: 1), (F: 4)]

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



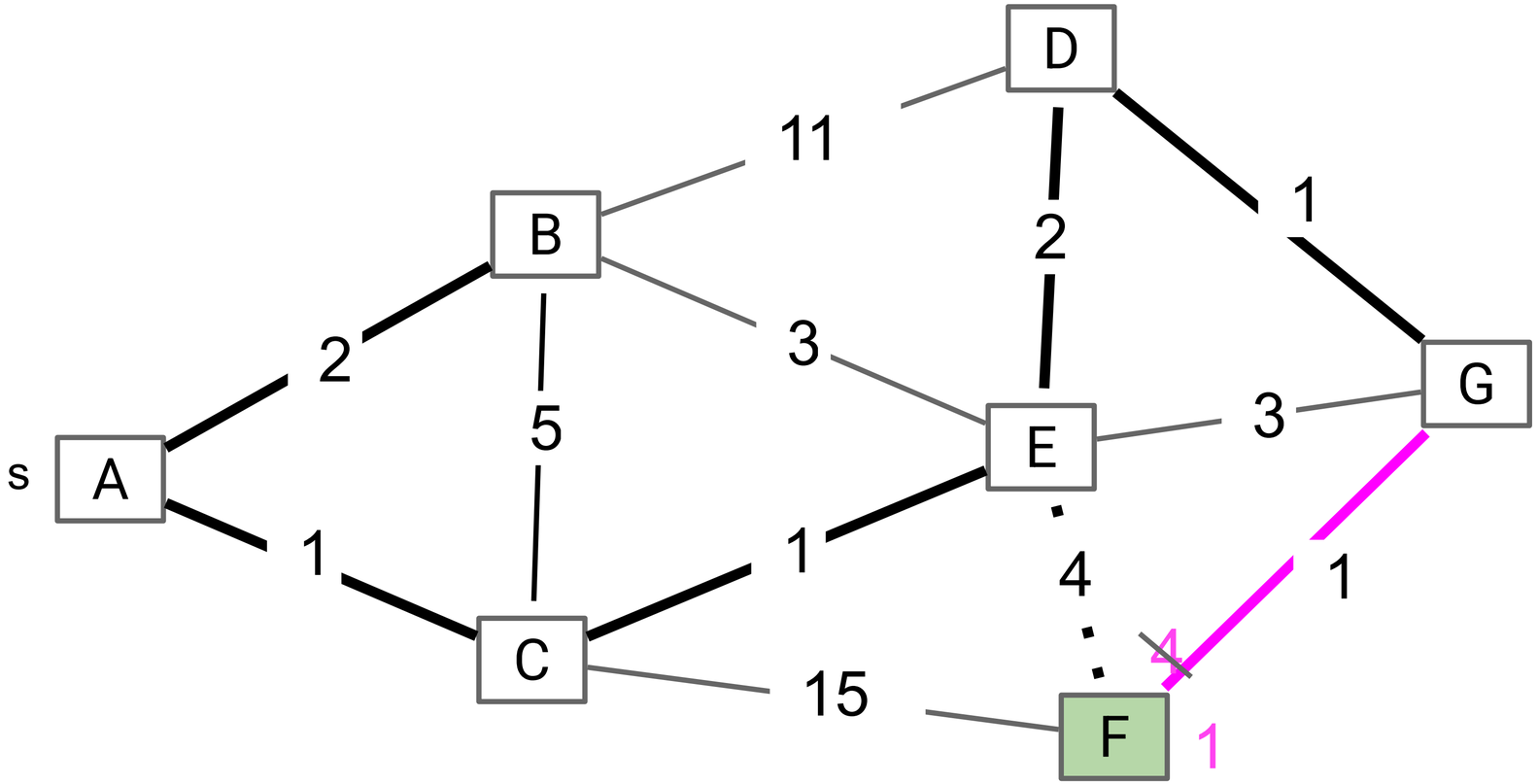
Fringe: [(G: 1), (F: 4)]

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**



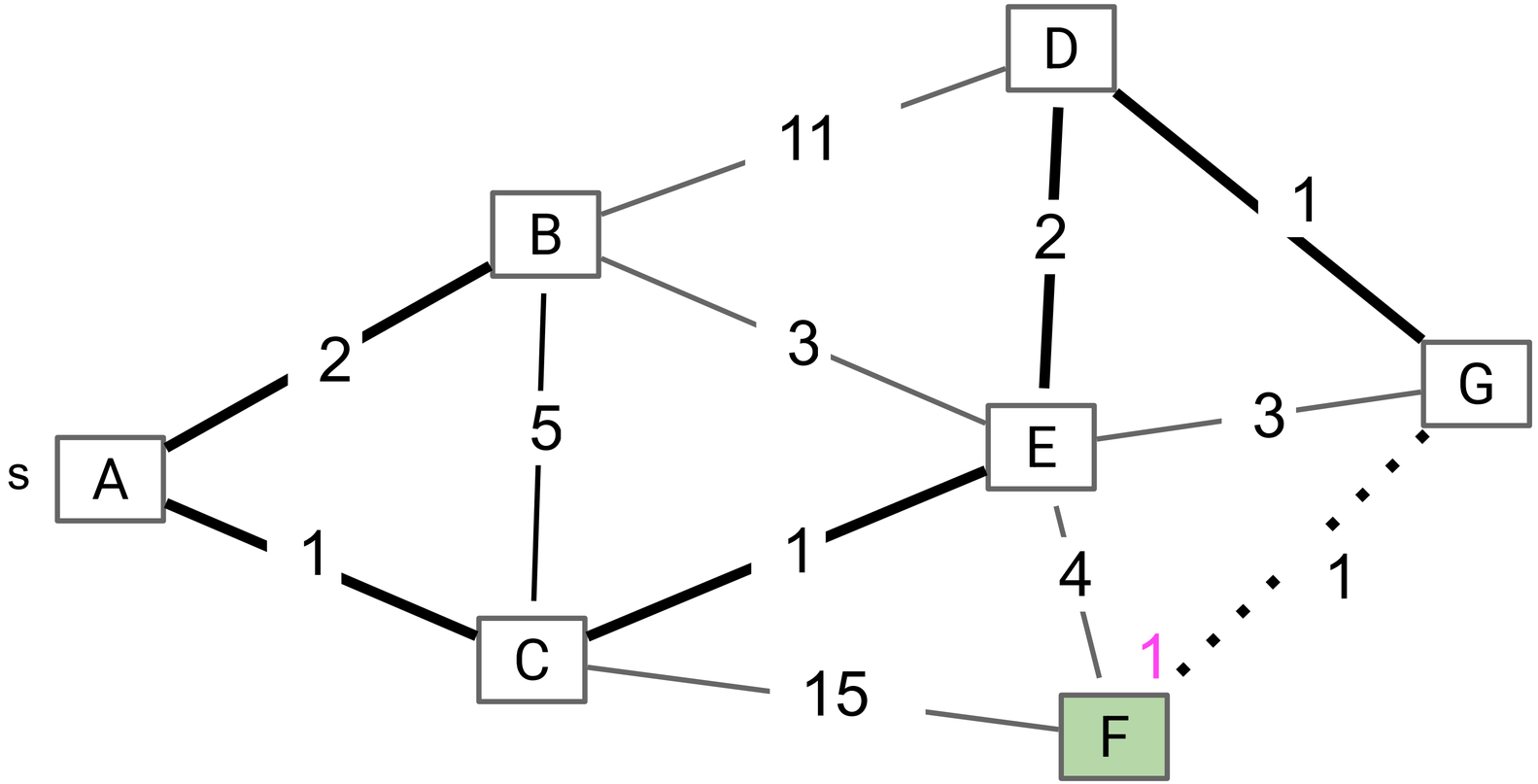
Fringe: [(F: 1)]

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**



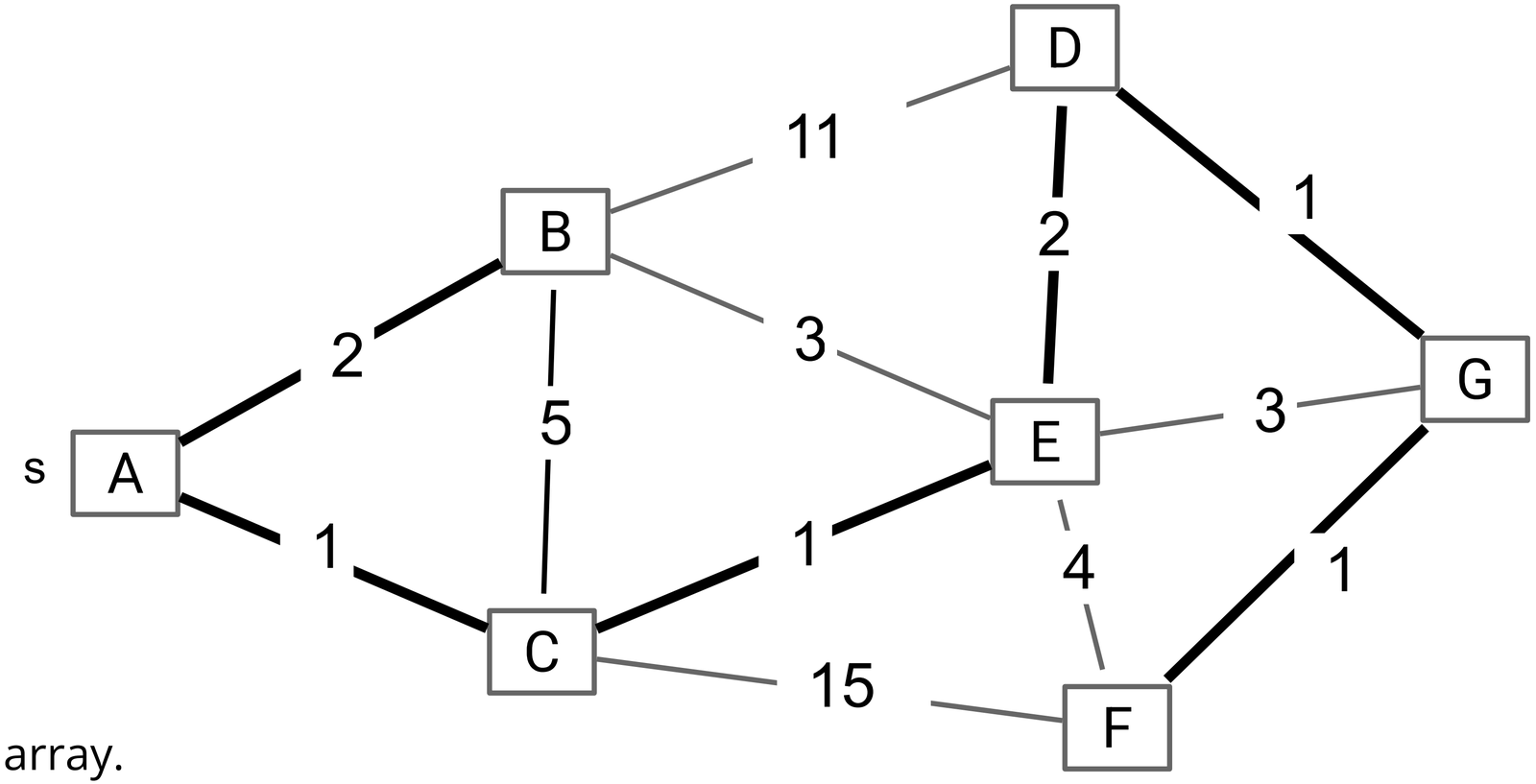
Fringe: [(F: 1)]

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**



Done! MST is represented by edgeTo array.

Fringe: []

Prim’s algorithm code & runtime

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 Implementation (Pseudocode, 1/2)

```
public class PrimMST {
  public PrimMST(EdgeWeightedGraph G) {
    edgeTo = new Edge[G.V()];
    distTo = new double[G.V()];
    marked = new boolean[G.V()];
    fringe = new SpecialPQ<Double>(G.V());

    distTo[s] = 0.0;
    setDistancesToInfinityExceptS(s);
    insertAllVertices(fringe);

    /* Get vertices in order of distance from tree. */
    while (!fringe.isEmpty()) {
      int v = fringe.delMin();
      scan(G, v);
    }
  }
  ...
```

Fringe is ordered by distTo tree. Must be a specialPQ like Dijkstra's.

Get vertex closest to tree that is unvisited.

Scan means to consider all of a vertices outgoing edges.

Prim's Implementation (Pseudocode, 2/2)

```
  while (!fringe.isEmpty()) {
    int v = fringe.delMin();
    scan(G, v);
  }
```

Important invariant, fringe must be ordered by current best known distance from tree.

```
private void scan(EdgeWeightedGraph G, int
v) {
  marked[v] = true;
  for (Edge e : G.adj(v)) {
    int w = e.other(v);
    if (marked[w]) { continue; }
    if (e.weight() < distTo[w]) {
      distTo[w] = e.weight();
      edgeTo[w] = e;
      pq.decreasePriority(w, distTo[w]);
    }
  }
}
```

Vertex is closest, so add to MST.

Already in MST, so go to next edge.

Better path to a particular vertex found, so update current best known for that vertex.

Prim's Runtime

```
  while (!fringe.isEmpty()) {
    int v = fringe.delMin();
    scan(G, v);
  }
```

```
private void scan(EdgeWeightedGraph G, int
v) {
  marked[v] = true;
  for (Edge e : G.adj(v)) {
    int w = e.other(v);
    if (marked[w]) { continue; }
    if (e.weight() < distTo[w]) {
      distTo[w] = e.weight();
      edgeTo[w] = e;
      pq.decreasePriority(w, distTo[w]);
    }
  }
}
```

Q: What is the runtime of Prim's algorithm?

- Assume all PQ operations take O(log(V)) time.
- Give your answer in Big O notation.

Prim’s Algorithm Runtime

Priority Queue operation count, assuming binary heap based PQ:

- Insertion: V operations, each costing O(log V) time.
- Delete-min: V operations, each costing O(log V) time.
- Decrease priority: E operations, each costing O(log V) time.

Overall runtime: O(V*log(V) + V*log(V) + E*log(V)).

- Assuming E > V, this is just **O(E log V)** (Same as Dijkstra's).

	# Operations	Cost per operation	Total cost
PQ add	V	O(log V)	O(V log V)
PQ delMin	V	O(log V)	O(V log V)
PQ decreasePriority	O(E)	O(log V)	O(E log V)

Kruskal’s algorithm

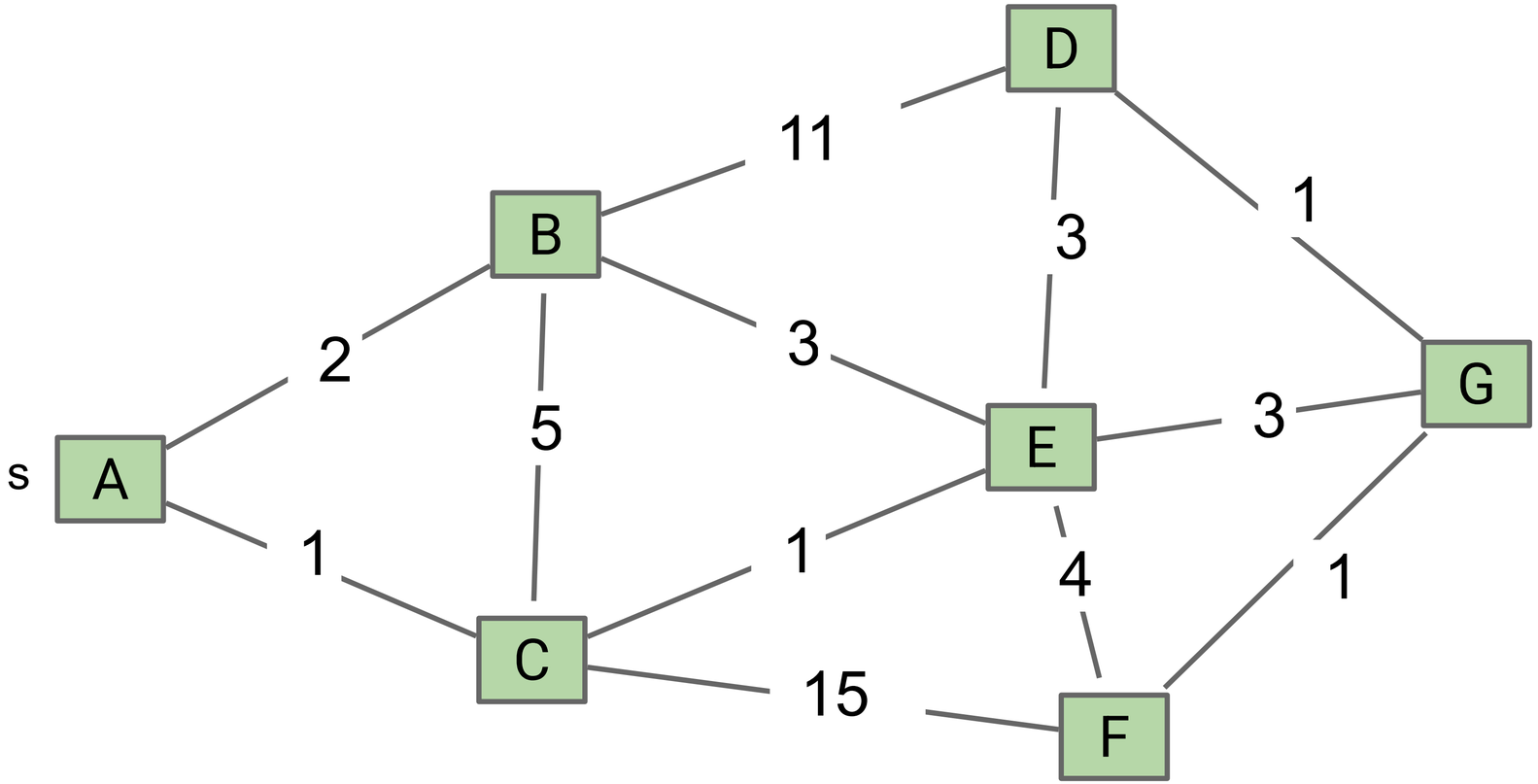
Kruskal’s algorithm overview

- Sort edges in ascending order of weight.
- Starting from the one with the smallest weight, add it to the MST unless doing so would create a cycle.
- Uses union-find, a data structure we haven't covered.

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
```



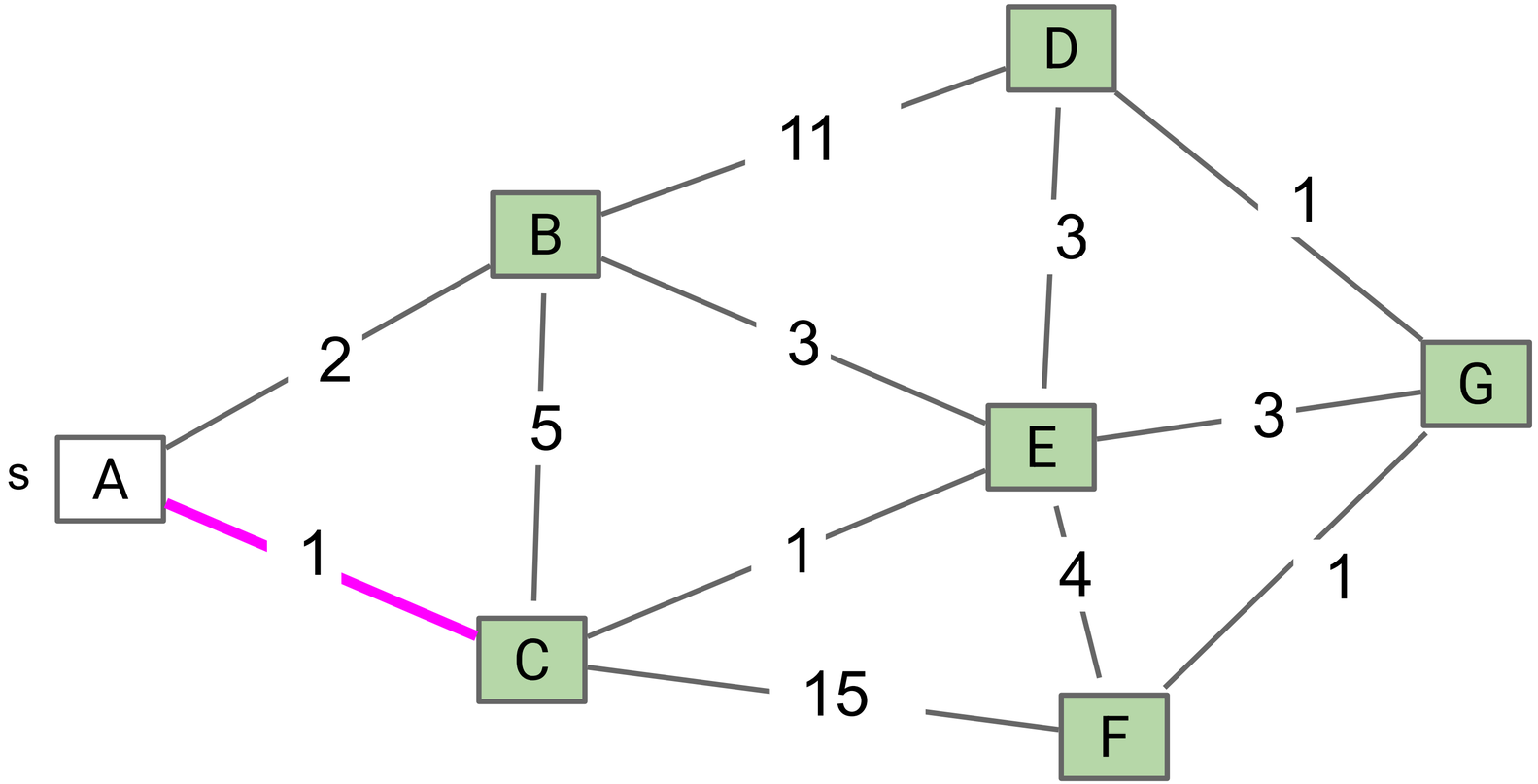
MST: []

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	Cycle?
C–F	15	

MST: [ ]



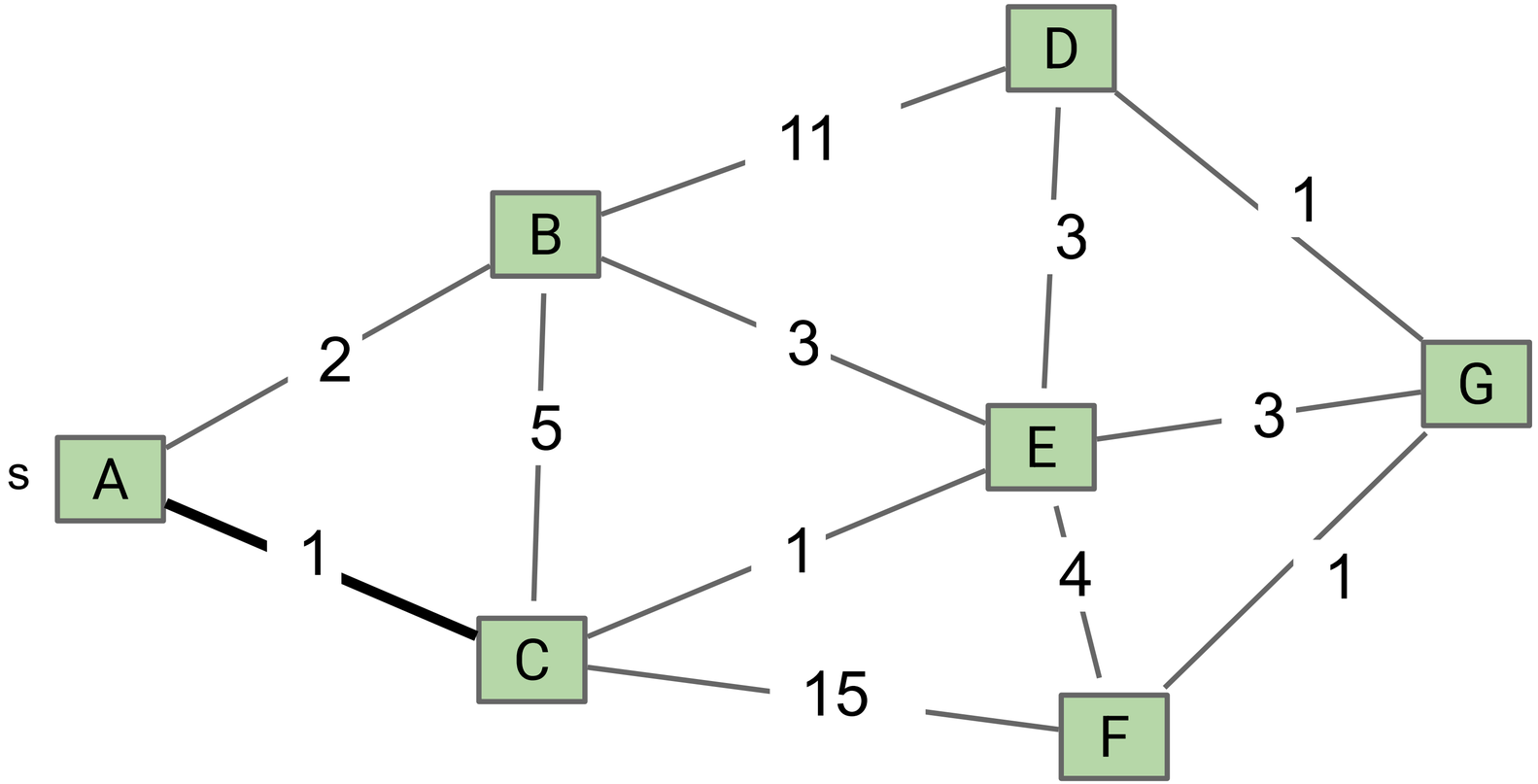
White and green colorings for vertices show cut being implicitly utilized by Kruskal's algorithm.

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.



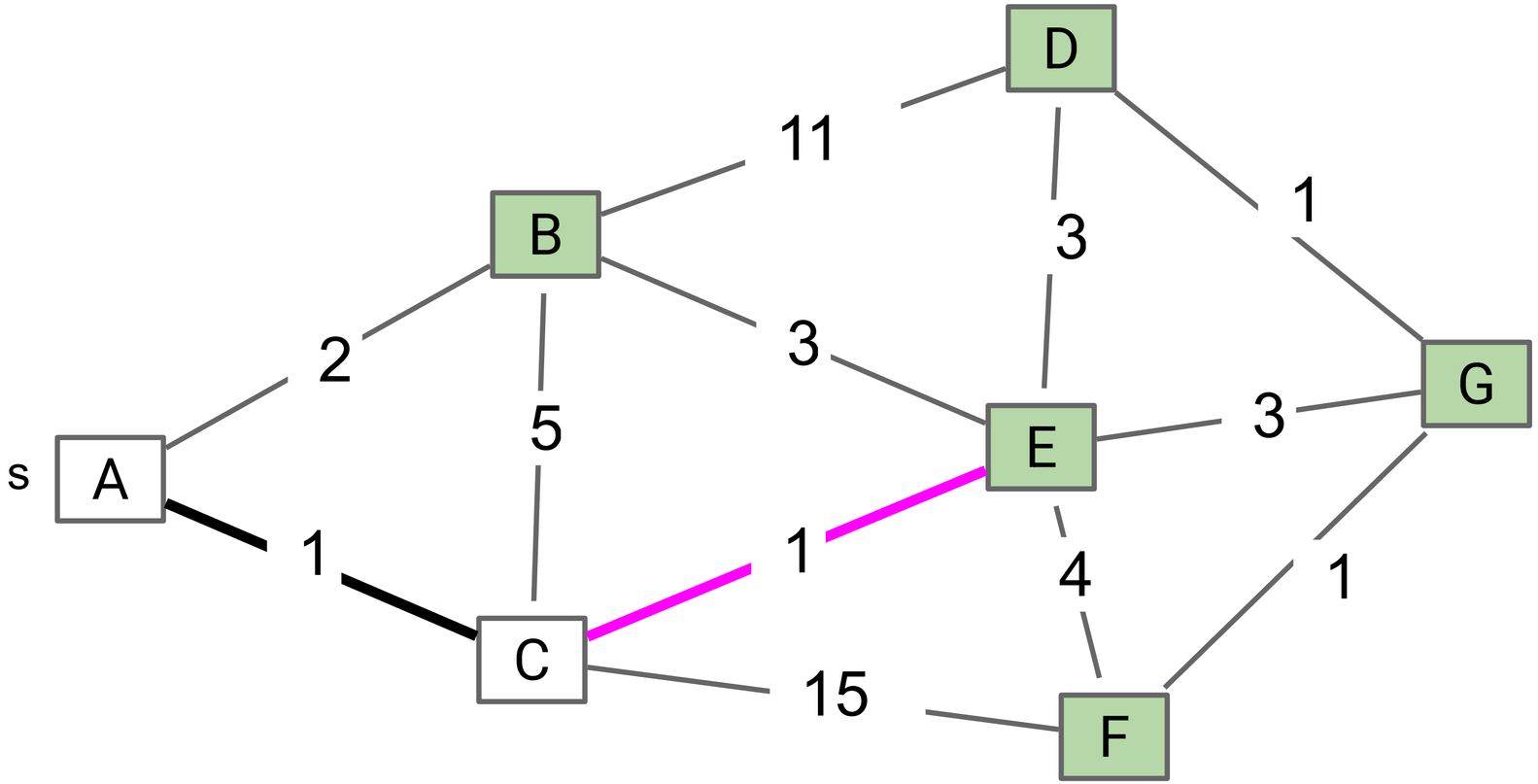
MST: [A–C]

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?



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

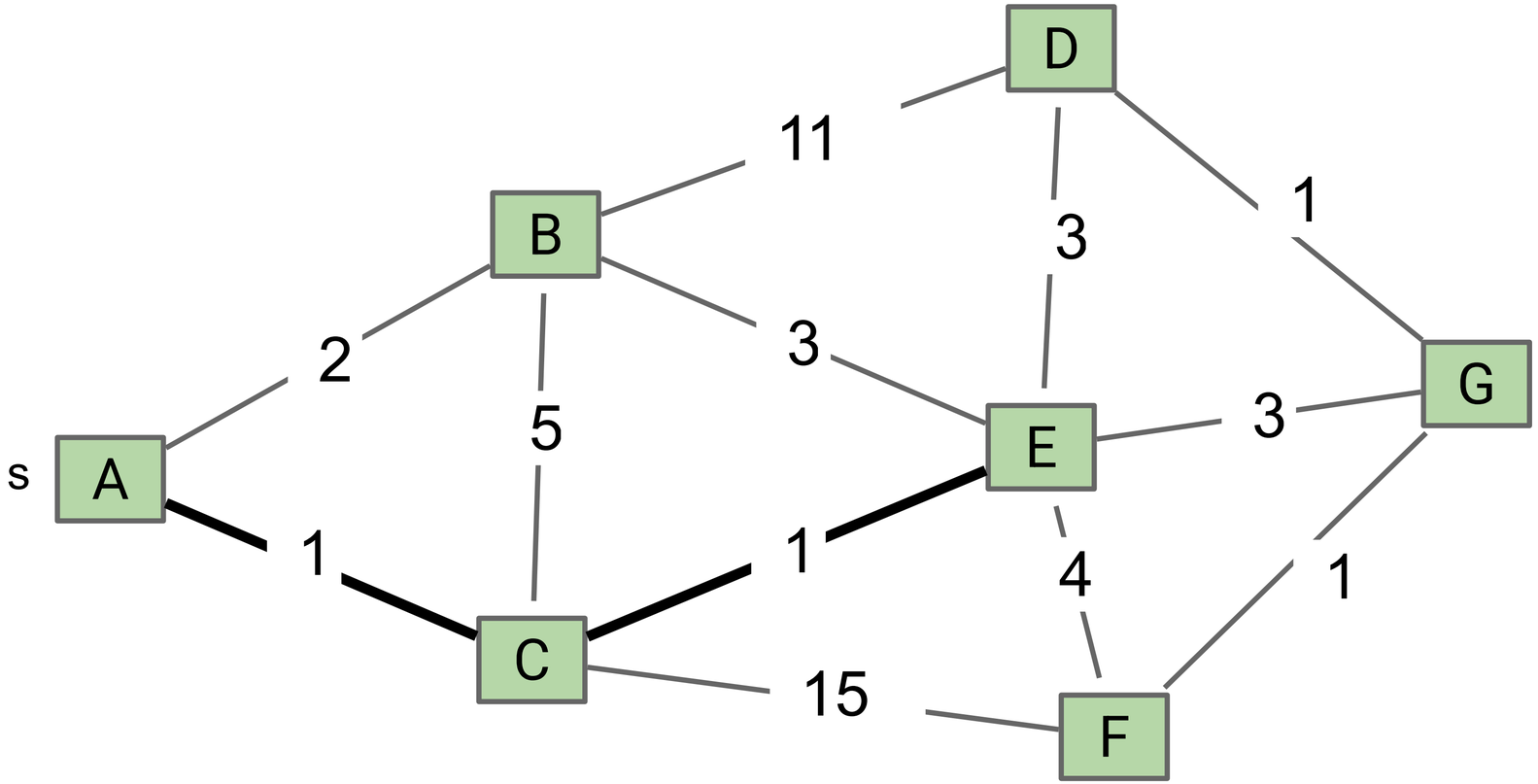
MST: [A–C]

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]

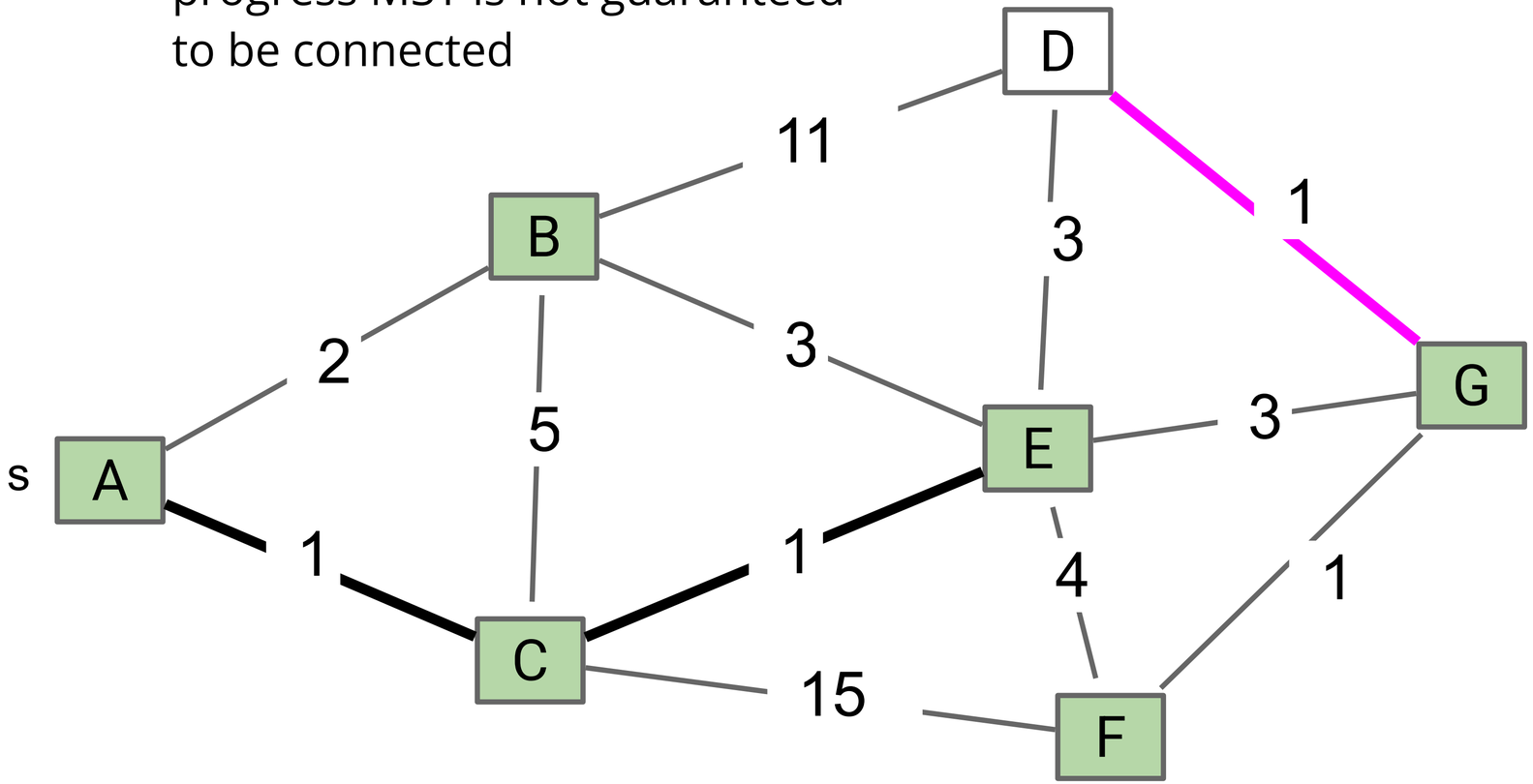
Kruskal's Demo

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?

Note that unlike Prim's, the in-progress MST is not guaranteed to be connected



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

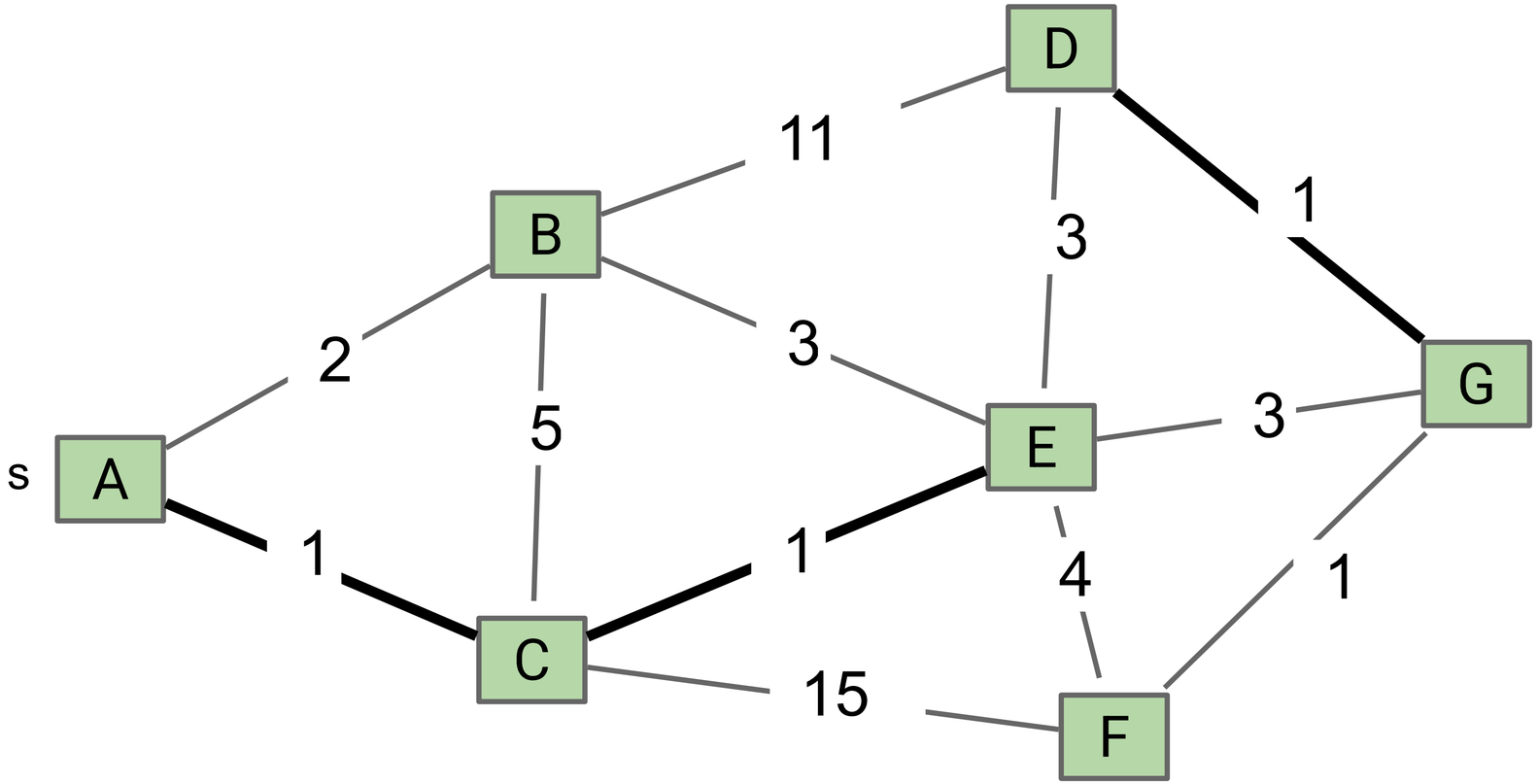
MST: [A–C, C–E]

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.

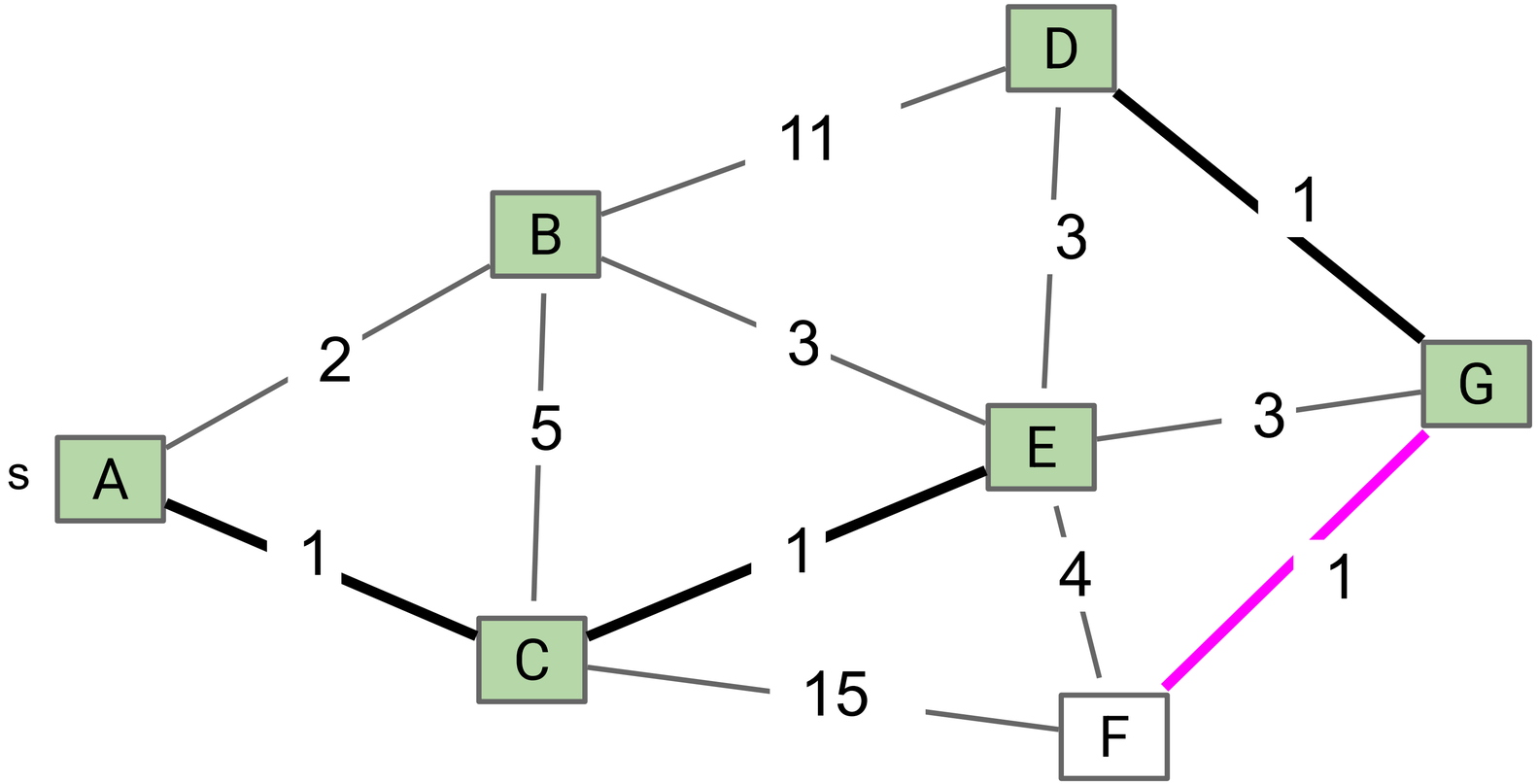


MST: [A–C, C–E, D–G]

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	Cycle?
C–F	15	



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

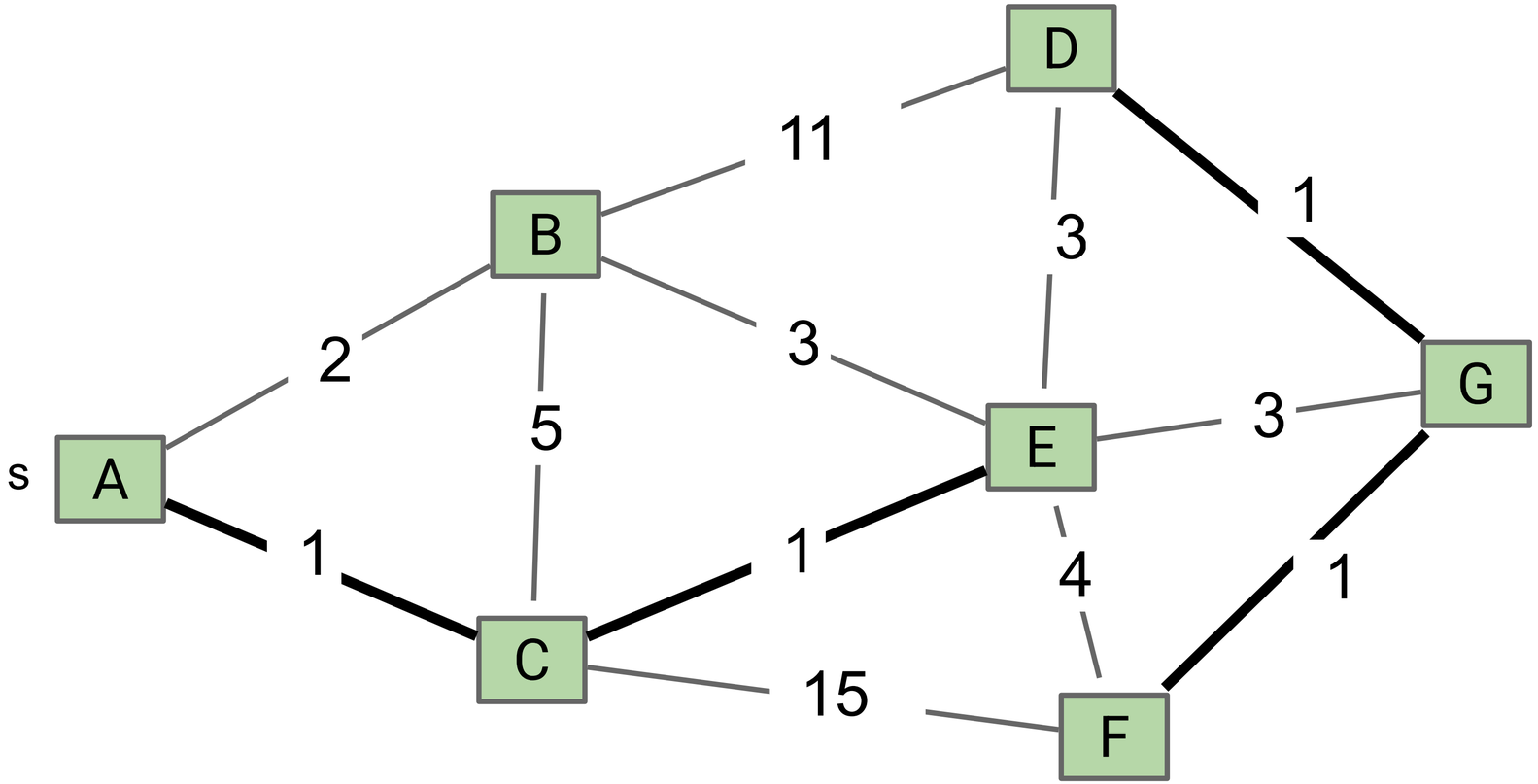
MST: [A–C, C–E, D–G]

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.

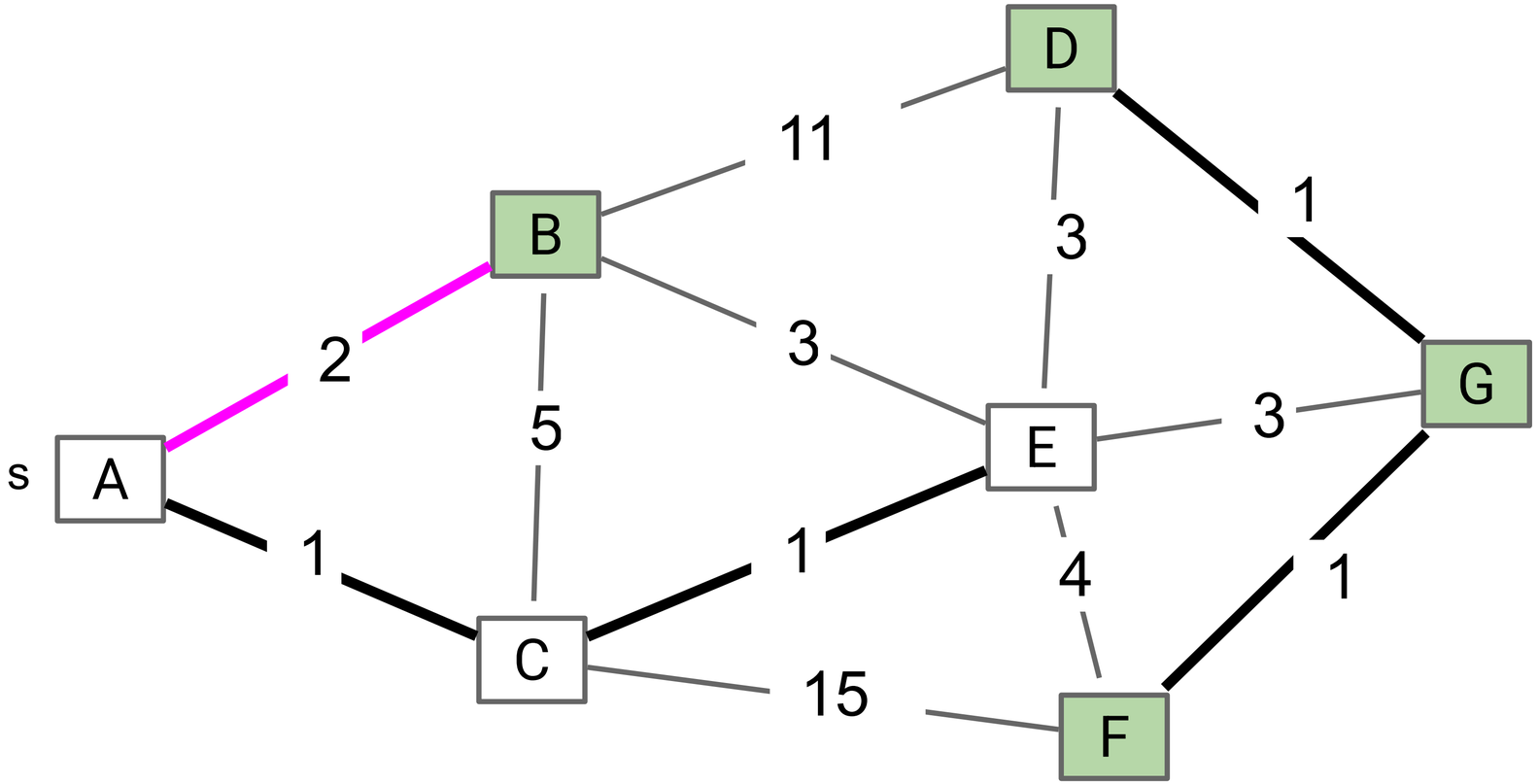


MST: [A–C, C–E, D–G, F–G]

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	Cycle?
C–F	15	



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

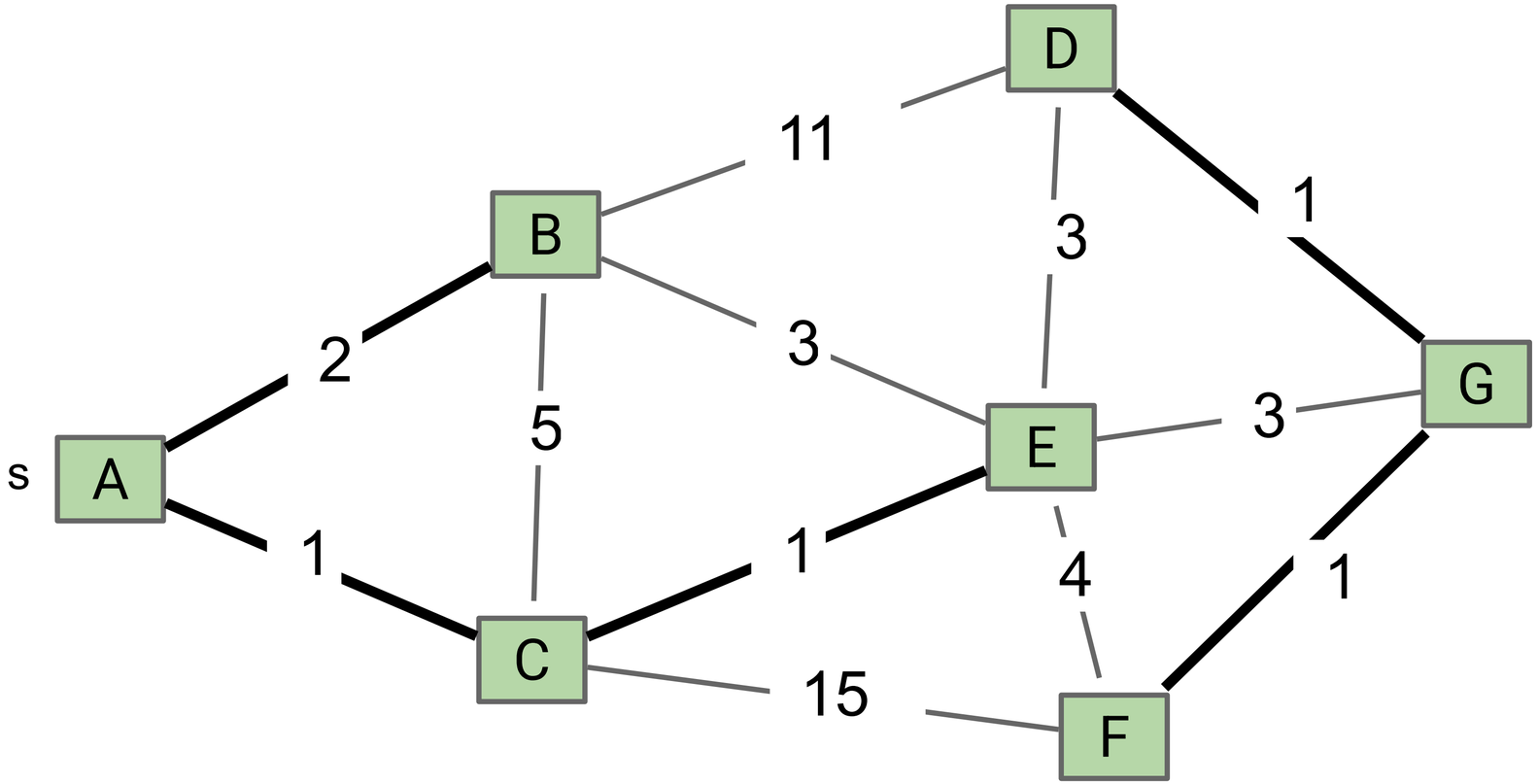
MST: [A–C, C–E, D–G, F–G]

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.

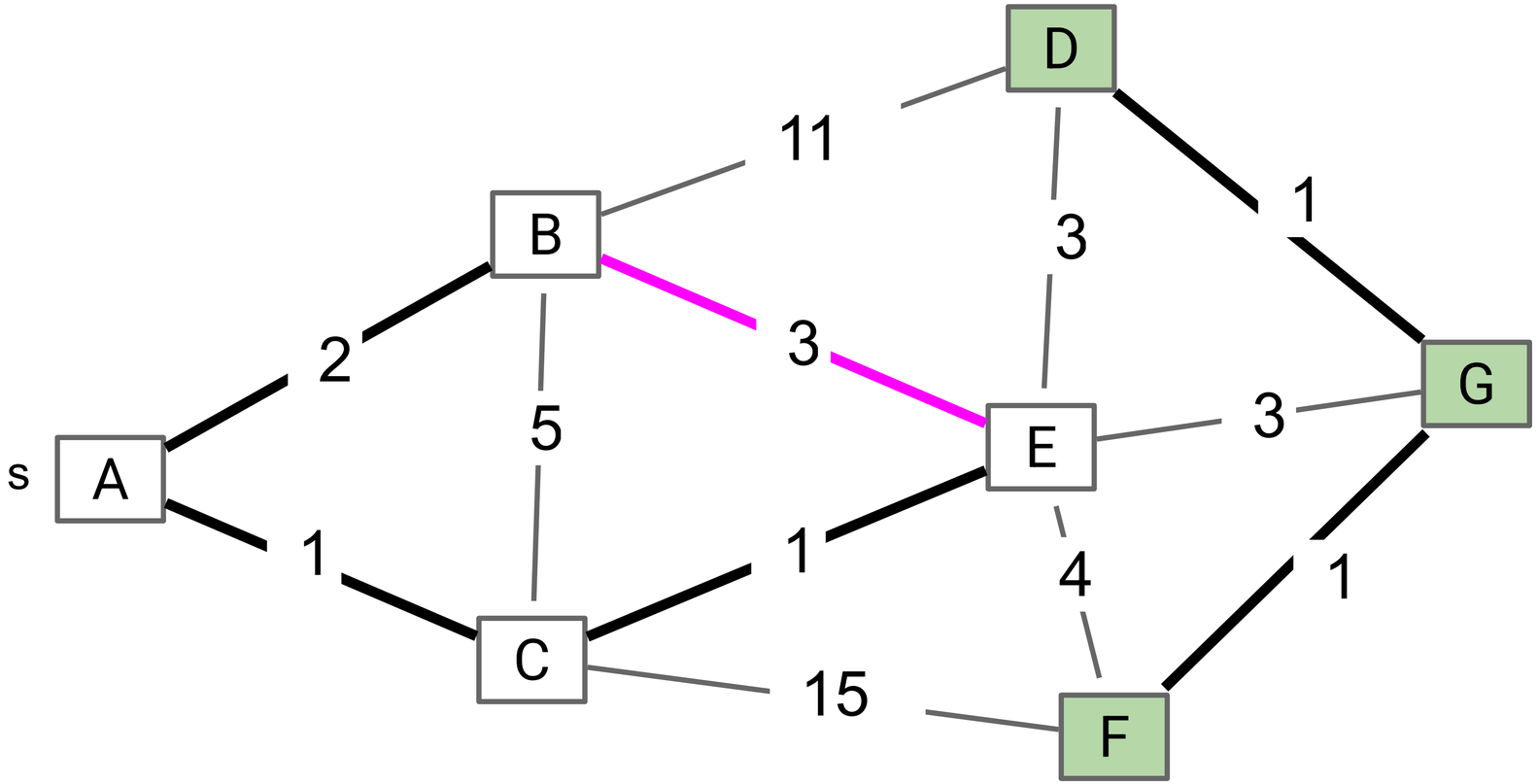


MST: [A–C, C–E, D–G, F–G, A–B]

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	Cycle?
C–F	15	



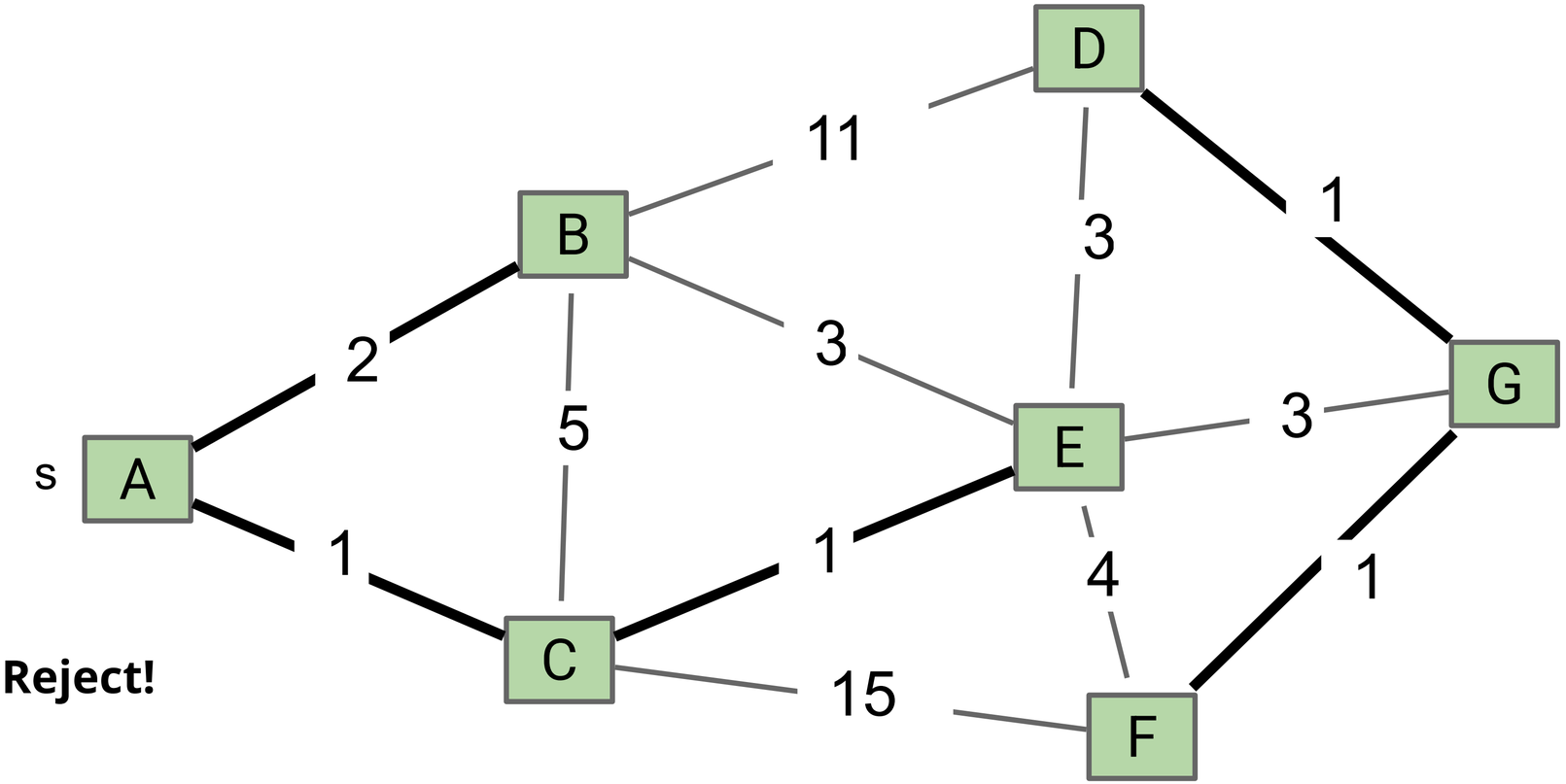
White and green colorings for vertices show cut being implicitly utilized by Kruskal's algorithm.

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!

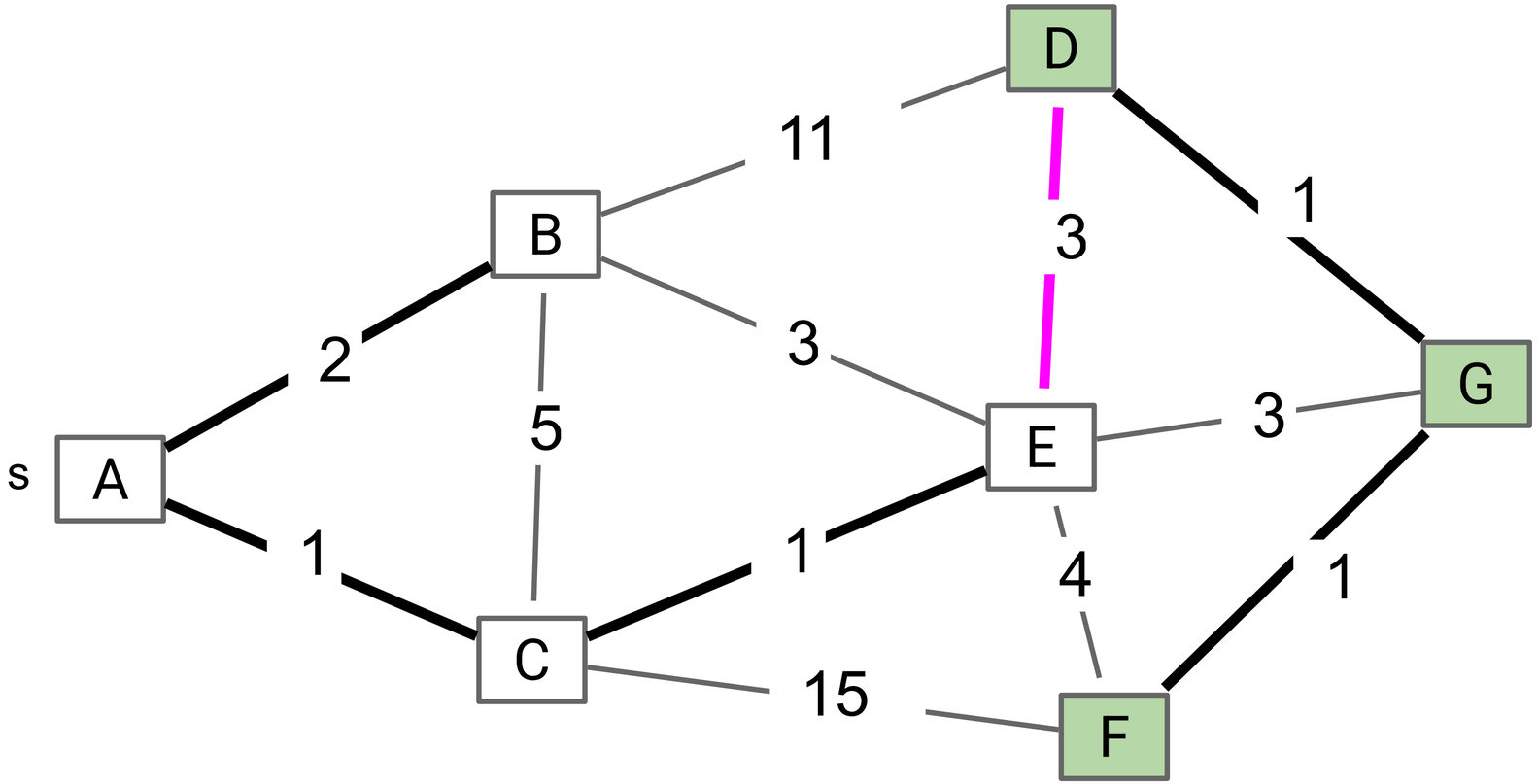
MST: [A–C, C–E, D–G, F–G, A–B]

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–C~~	~~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?



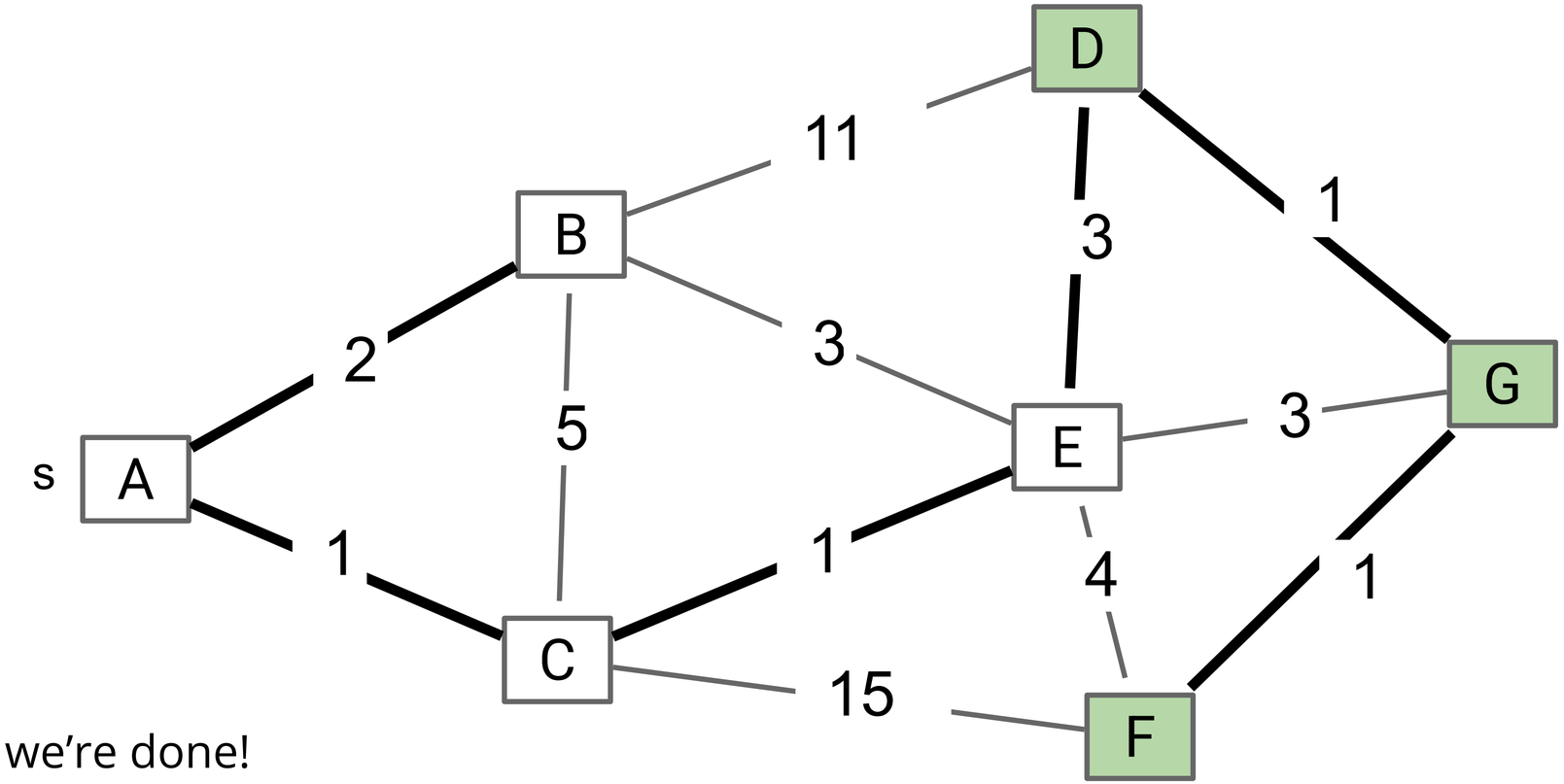
White and green colorings for vertices show cut being implicitly utilized by Kruskal's algorithm.

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.

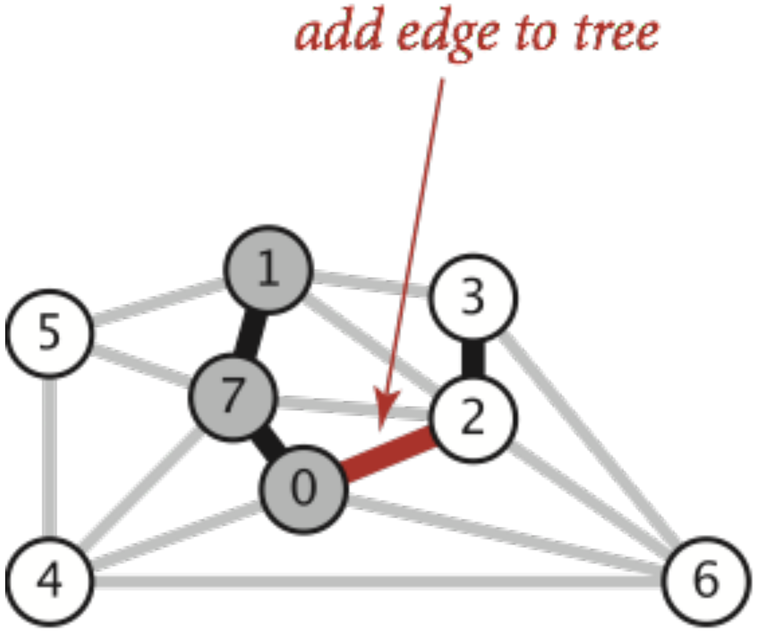
V-1 edges, so we're done!

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.



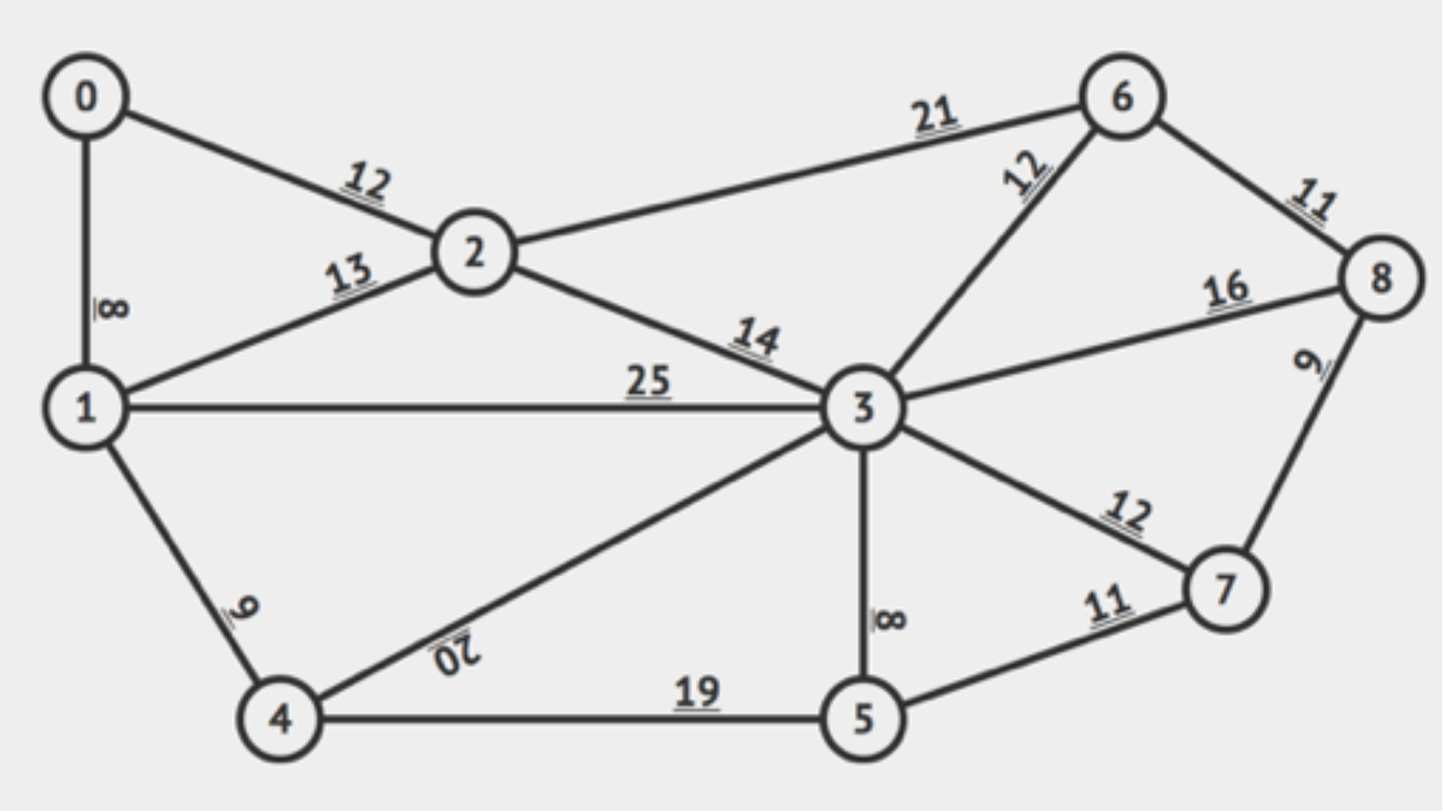
Why does Kruskal's work? Special case of generic MST algorithm (similar proof to Prim's).

- Suppose we add edge e = v->w.
- Side 1 of cut is all vertices connected to v, side 2 is everything else.
- No crossing edge is black (since we don't allow cycles).
- No crossing edge has lower weight (consider in increasing order).

How do we implement Kruskal's? Add the *edges* to a PQ (instead of vertices) and remove them one by one, checking for cycles, until V-1 edges have been added to the MST.

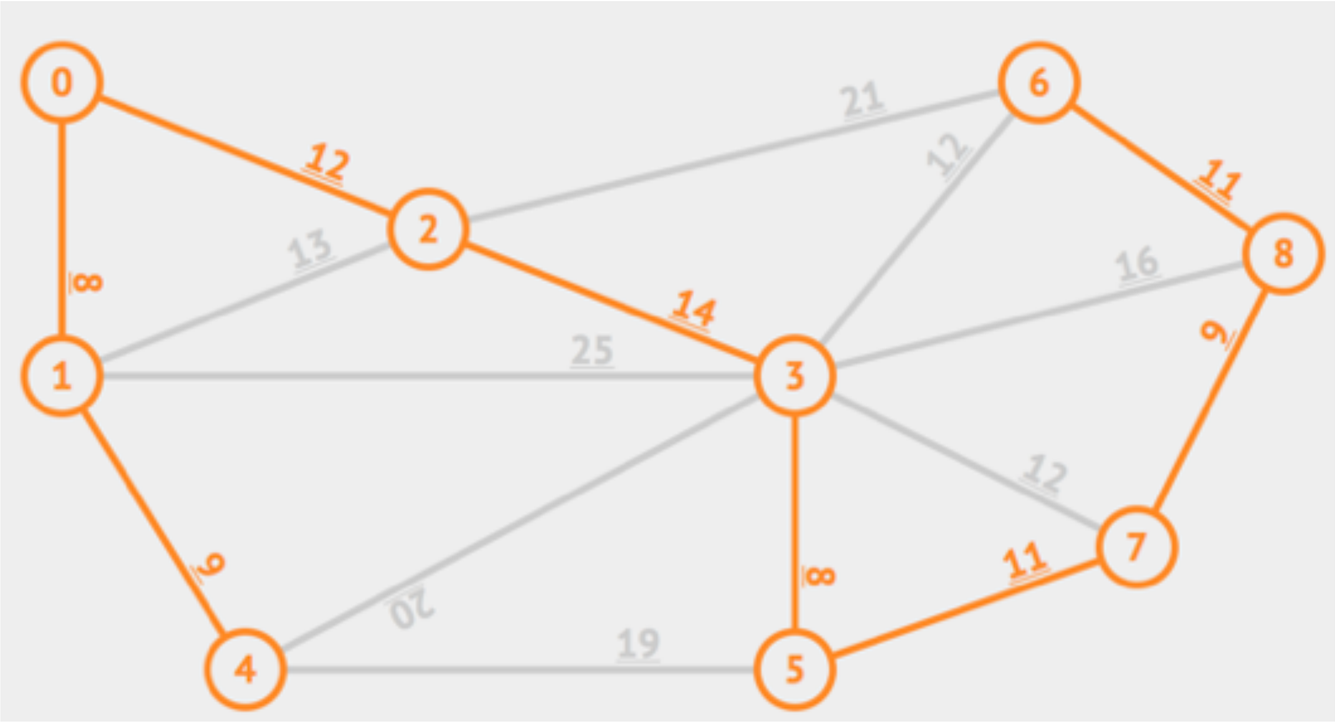
Worksheet time!

- Run Kruskal's on this graph.



Edge	Weight
0-1	8
3-5	8
1-4	9
7-8	9
5-7	11
6-8	11
0-2	12
3-6	12
3-7	12
1-2	13
2-3	14
3-8	16
4-5	19
4-3	20
2-6	21
1-3	25

Worksheet answer



Kruskal’s code and runtime

Kruskal's Implementation (Pseudocode)

```
public class KruskalMST {
  private List<Edge> mst = new ArrayList<Edge>();

  public KruskalMST(EdgeWeightedGraph G) {
    MinPQ<Edge> pq = new MinPQ<Edge>();
    for (Edge e : G.edges()) {
      pq.insert(e);
    }
    WeightedQuickUnionPC uf =
              new WeightedQuickUnionPC(G.V());
    while (!pq.isEmpty() && mst.size() < G.V() - 1)
{
      Edge e = pq.delMin();
      int v = e.from();
      int w = e.to();
      if (!uf.connected(v, w)) {
        uf.union(v, w);
        mst.add(e);
} } } }
```

We don't cover this data structure (quick union) in the course, see resources slide for more

Storing edges in PQ

Run time is **O(Elog(E))** if edges are not pre-sorted

Kruskal's Runtime

Kruskal's algorithm on previous slide is O(E log E).

Fast heap construction algorithm

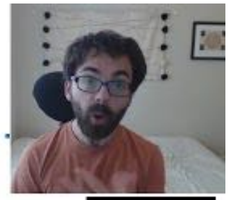
Operation	Number of Times	Time per Operation	Total Time
Insert	E	O(log E)	O(E)
Delete minimum	O(E)	O(log E)	O(E log E) O(E) if edges pre-sorted
union	O(V)	O(log* V)	O(V log* V)
isConnected	O(E)	O(log* V)	O(E log* V)

Note: A weighted quick union is represented as a tree, so most operations are log(H)

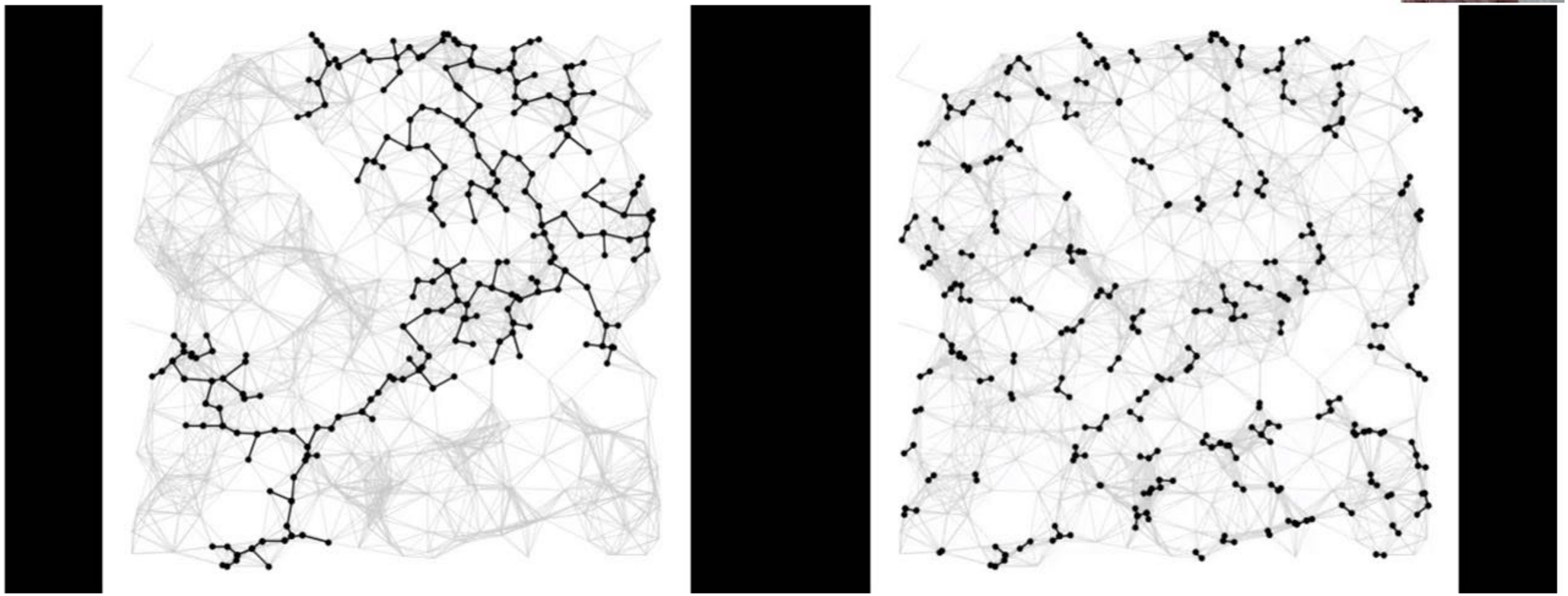
Note 1: If we use a **pre-sorted list of edges** (instead of a PQ), then we can simply iterate through the list in O(E) time (no need to delete minimum), so overall runtime is O(E + V log* V + E log* V) = **O(E log* V)**.

Note 2: $E < V^2$, so $\log E < \log V^2 = 2 \log V$, so O(E log E) = O(E log V). So while Kruskal's algorithm will be slower than Prim's algorithm for a worst-case unsorted set of edges, it won't be asymptotically slower.

Prim's vs. Kruskal's (visual)



Prim's vs. Kruskal's



Prim's Algorithm

Kruskal's Algorithm

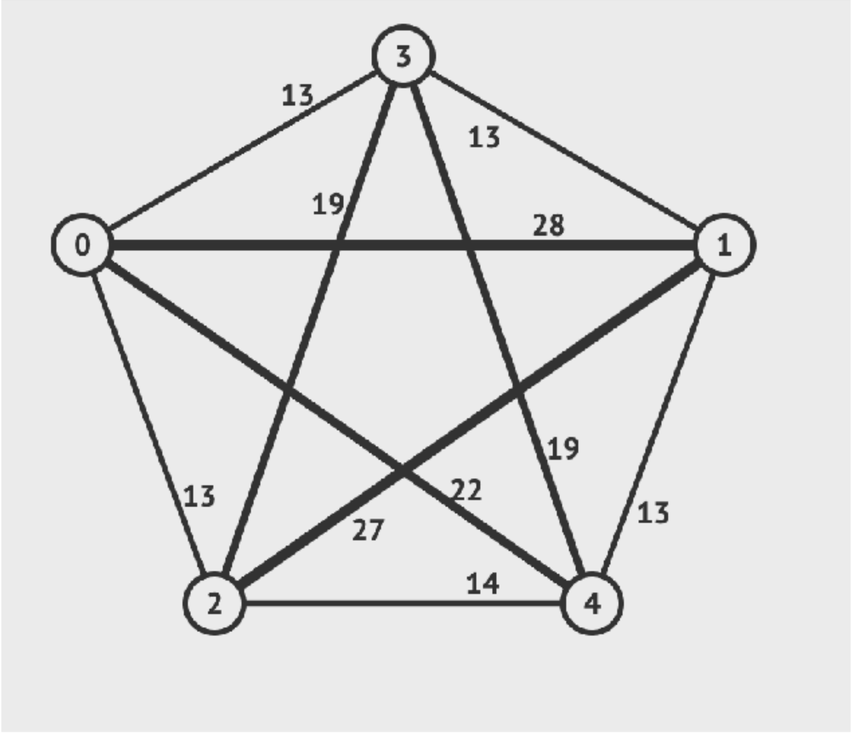
Demos courtesy of Kevin Wayne, Princeton University

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 with pre-sorted edges	O(E log V)	Uses weighted quick-union with path compression

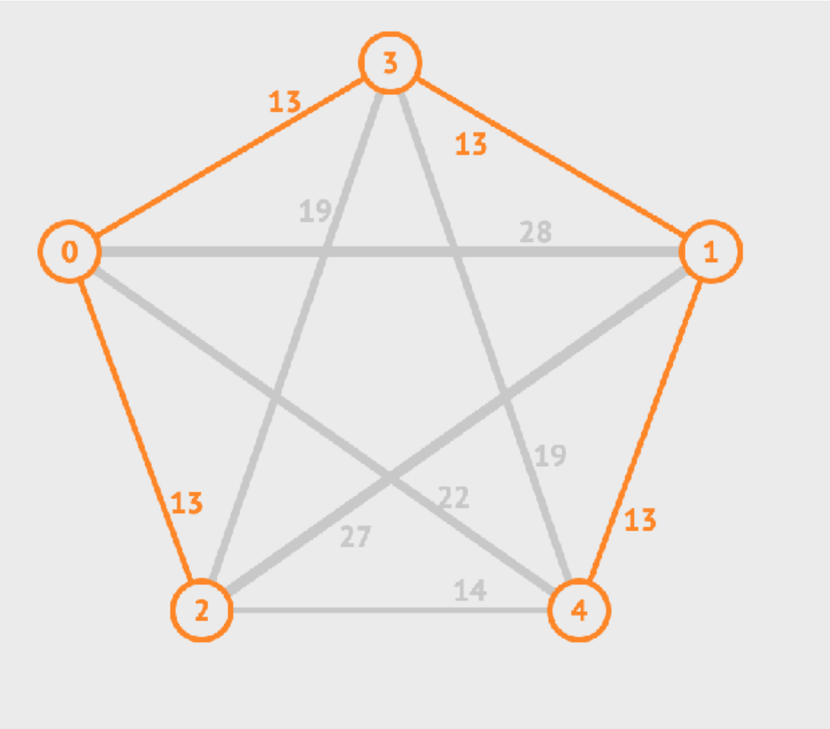
Worksheet time!

- Run Kruskal's and Prim's algorithm (starting at index 0) on the following graph. Do they give the same answer?



Worksheet answer

- Run Kruskal's and Prim's algorithm (starting at index 0) on the following graph.
- Yes, same MST.



Lecture 22 wrap-up

- Lab final project check-in tonight
- Final project part 1 (PDF write up with grading contract, dataset, interface file) due Fri 11:59pm
- HW10: On The Road due next Tues 11:59pm
- **Checkpoint 3 in two weeks** (12/3). Covers material up to next Monday (DAGs). Class Dec 1 will be course evals + review session + algo design practice (you'll come up with questions!)
- Too many resources, they are the next slide. :)

Resources

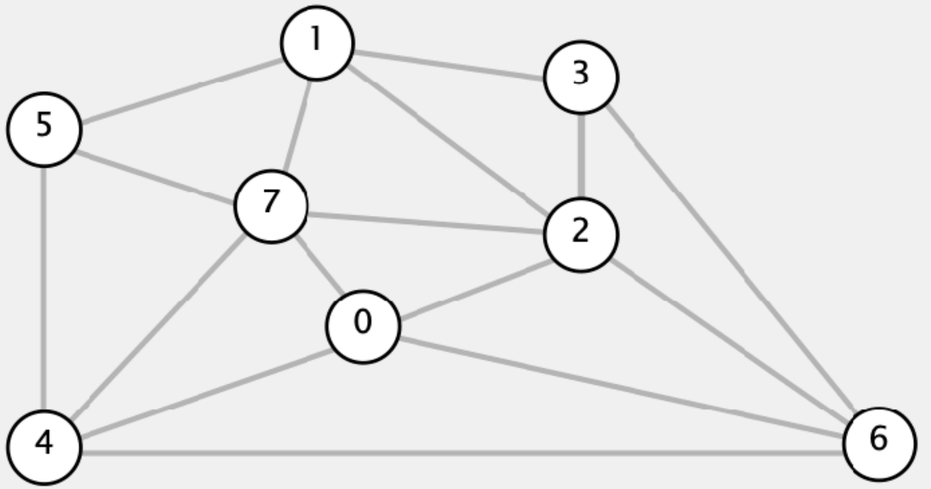
- Graph history: https://cs.pomona.edu/classes/cs62/history/graphs
  - Learn more about Dijkstra, Prim, and Kruskal
- Recommended Textbook: Chapter 4.3 (Pages 604-629)
- Website: https://algs4.cs.princeton.edu/43mst/
- Visualization: https://visualgo.net/en/mst
- Weighted quick union: https://joshhug.gitbooks.io/hug61b/content/chap9/chap94.html
- Practice problems behind this slide

Problem 1

- Run Kruskal's on this graph

Consider edges in ascending order of weight.

- Add next edge to tree T unless doing so would create a cycle.



an edge-weighted graph

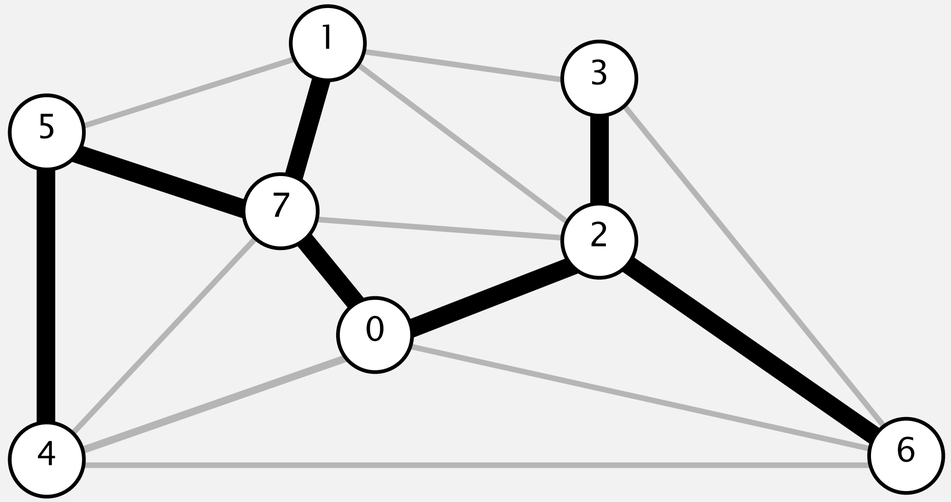
graph edges sorted by weight

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

Answer 1

Consider edges in ascending order of weight.

- Add next edge to tree T unless doing so would create a cycle.



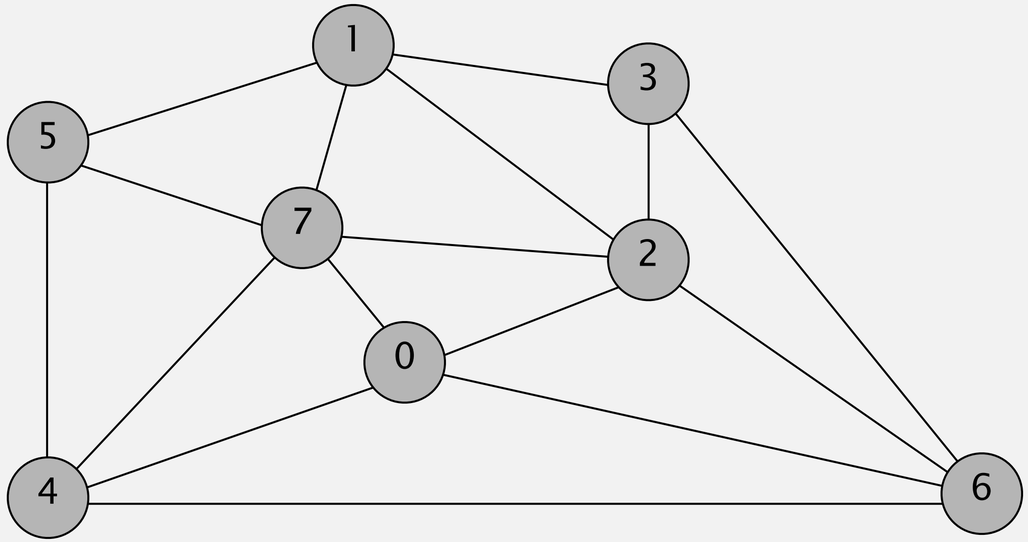
a minimum spanning tree

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

Problem 2

- Run Prim's on this graph starting at 0

- Start with vertex 0 and greedily grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until $V - 1$ edges.

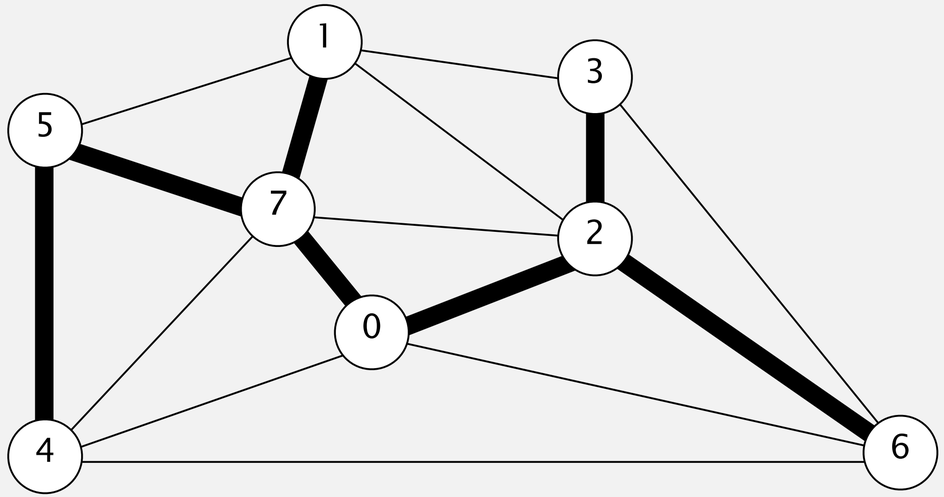


an edge-weighted graph

```
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
```

Answer 2

- Start with vertex 0 and greedily grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until $V - 1$ edges.



MST edges

0-7 1-7 0-2 2-3 5-7 4-5 6-2