Announcements

- HW2 was out last Friday
  - Slightly more than 2 weeks for completing it
  - A typo in last question, have updated draft in Collab

- Mid-term 1 will be in October 8'th, Thursday, 9 am to 11 pm
  - Please arrange your schedules properly
  - Should take you about 2.5 hours to complete
  - No class neither office hours on that day
  - Will cover everything including graph algorithms

CS6161: Design and Analysis of Algorithms (Fall 2020)

Graph Algorithms

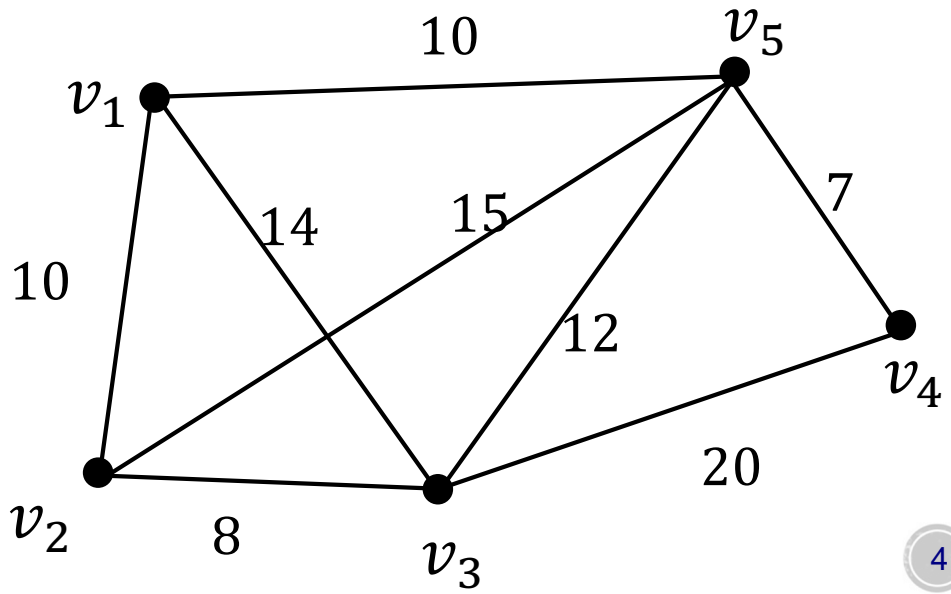
Instructor: Haifeng Xu

Outline

- Warm-up for Graph Algorithms
- Minimum Spanning Trees (MSTs)

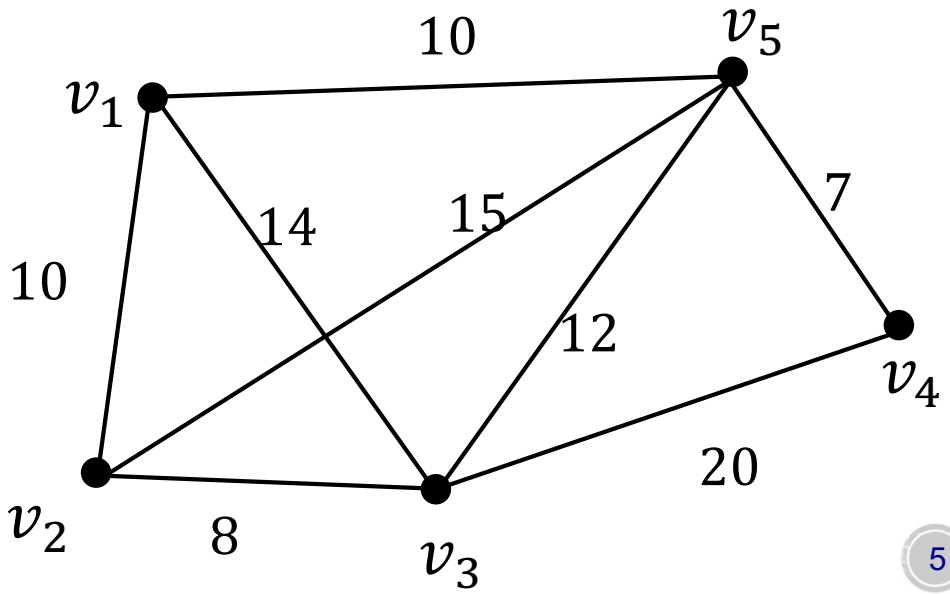
Recall graphs

- Node set V
- Edge set E, may have weights/costs/distances c_e
- Denoted as $G = (V, E, \{c_e\}_{e \in E})$



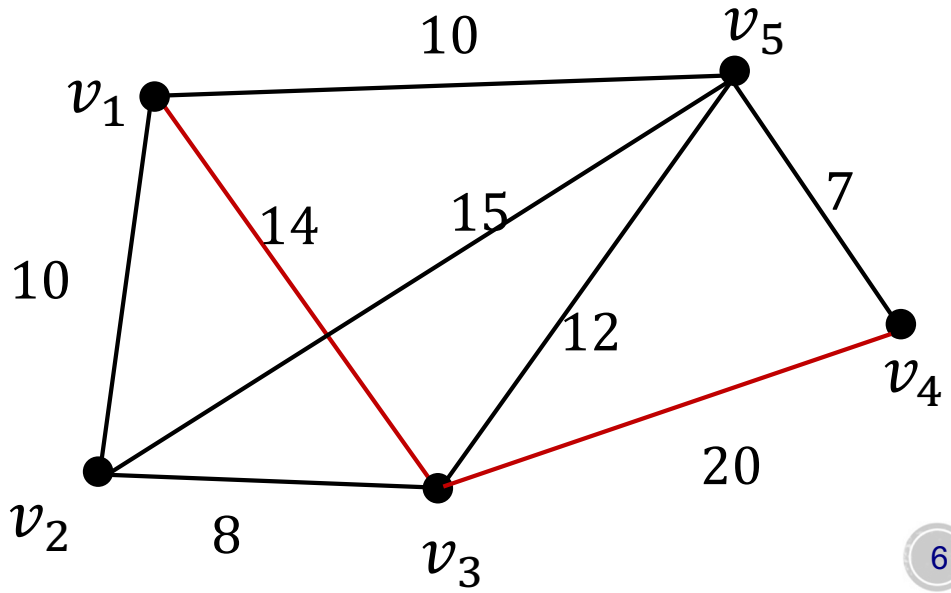
Notions

- Degree of v: how many edges it has
  - E.g., degree of v_1 is 3
  - Can has in-degree and out-degree for directed graph



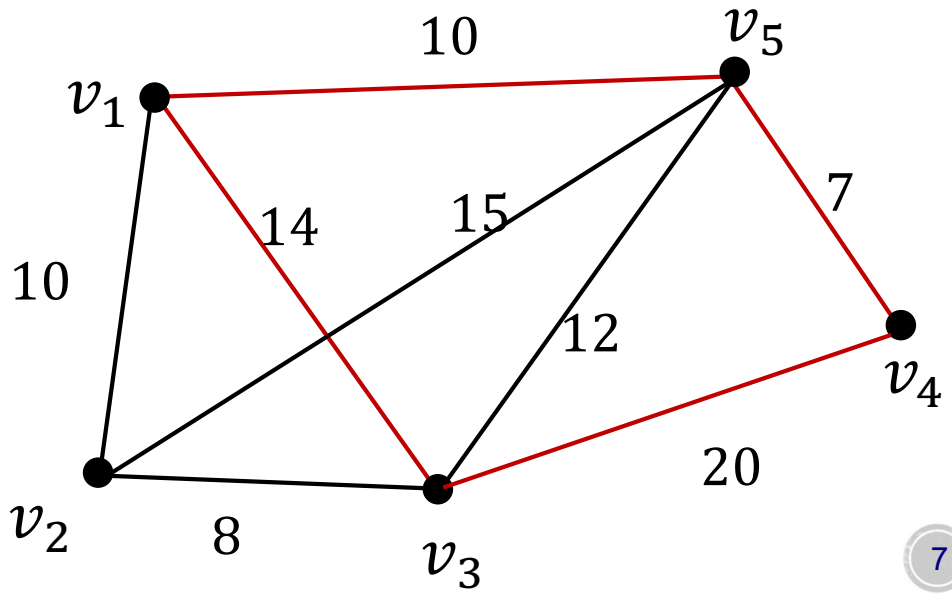
Notions

- Degree of v: how many edges it has
  - E.g., degree of v_1 is 3
  - Can has in-degree and out-degree for directed graph
- Path (of length k) from v_i to v_j
  - E.g., the path from v_1 to v_4 has length 2
  - Shortest path from v_i to v_j: the one with smallest length (maybe in weighted sense)



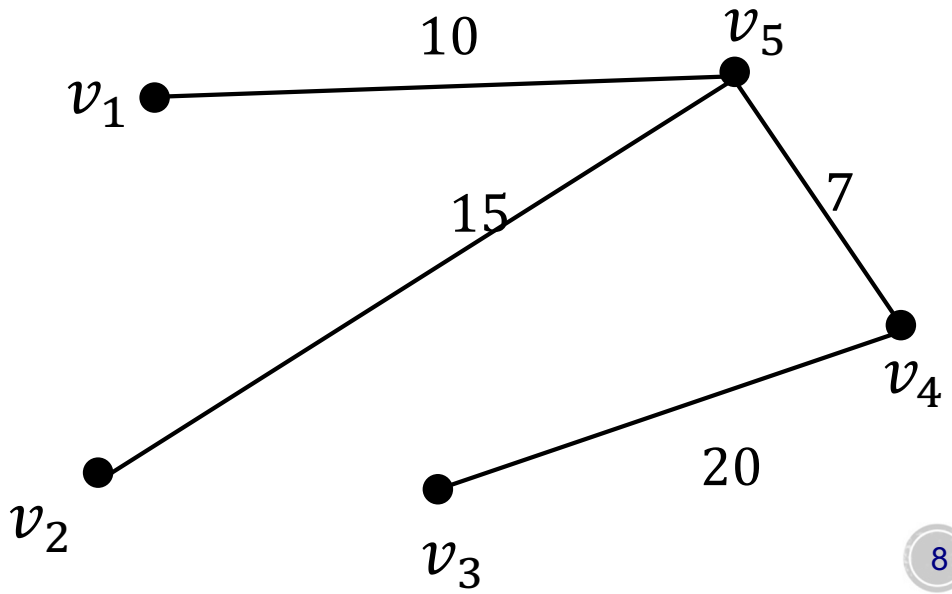
Notions

- Cycle: a sequence of edges that start and end at the same vertex
  - Simple cycle: visit its vertices only once
- Connectivity: whether all nodes are connected?



Notions

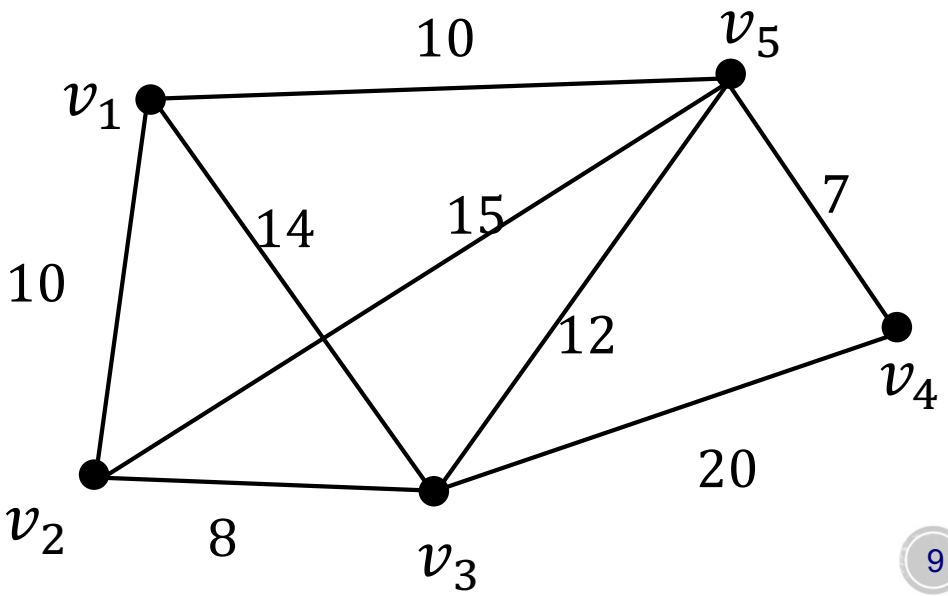
- Trees: a special type of graph



Representing Undirected Graphs

- Adjacent matrix A

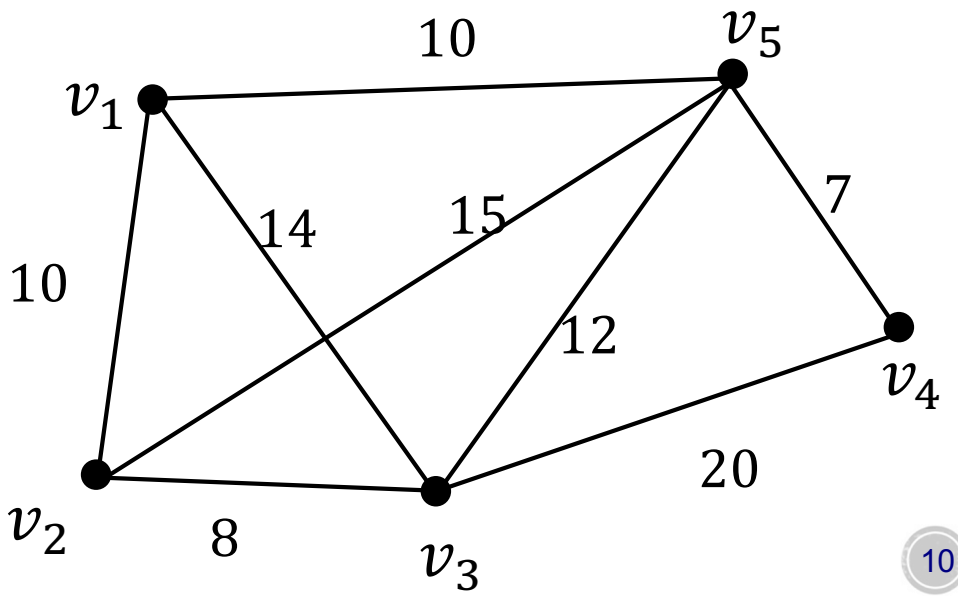
	1	2	3	4	5
1					
2					
3					
4					
5					



Representing Undirected Graphs

- Adjacent matrix A

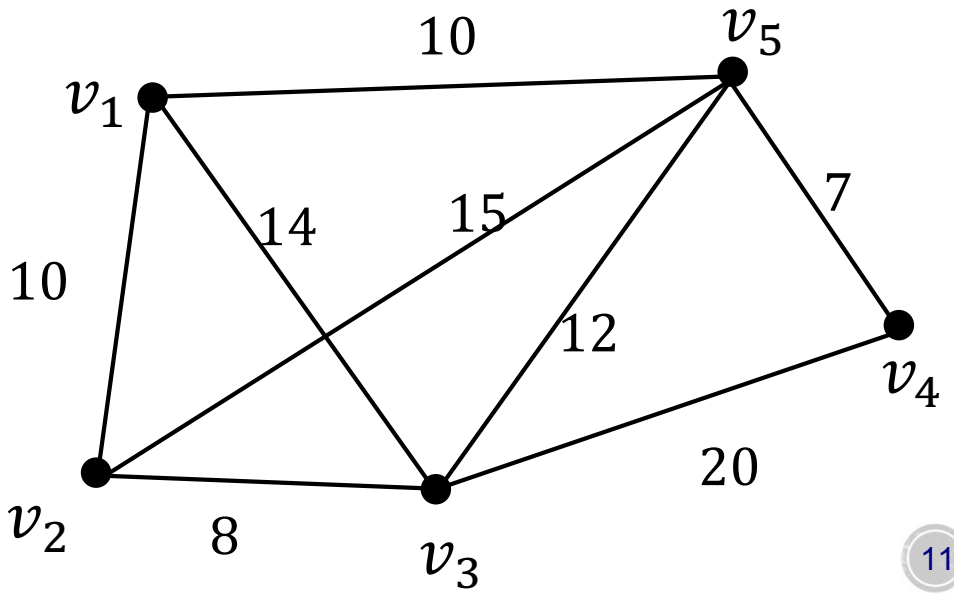
	1	2	3	4	5
1		10			
2	10				
3					
4					
5					



Representing Undirected Graphs

- Adjacent matrix A

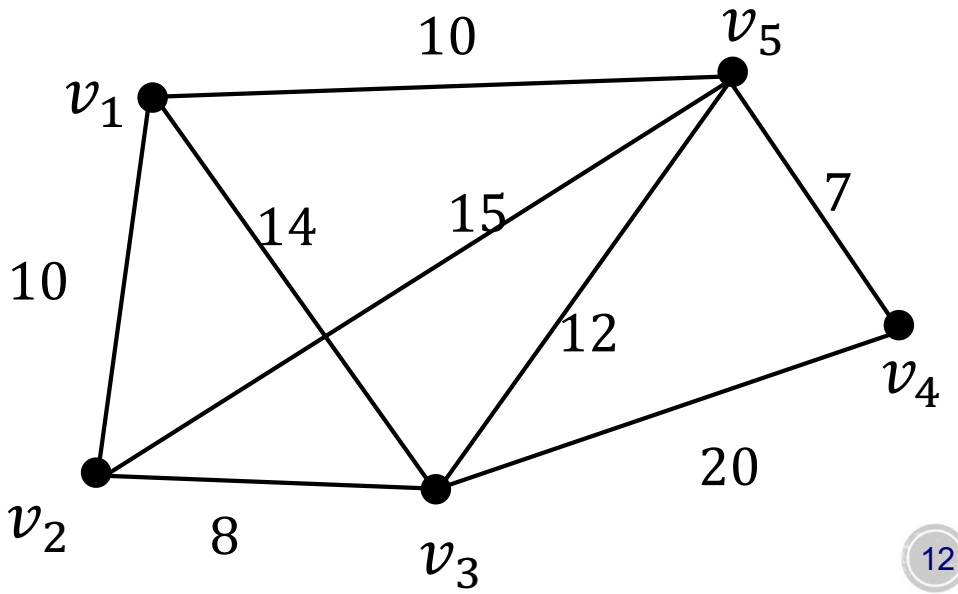
	1	2	3	4	5
1		10	14		
2	10				
3	14				
4					
5					



Representing Undirected Graphs

- Adjacent matrix A

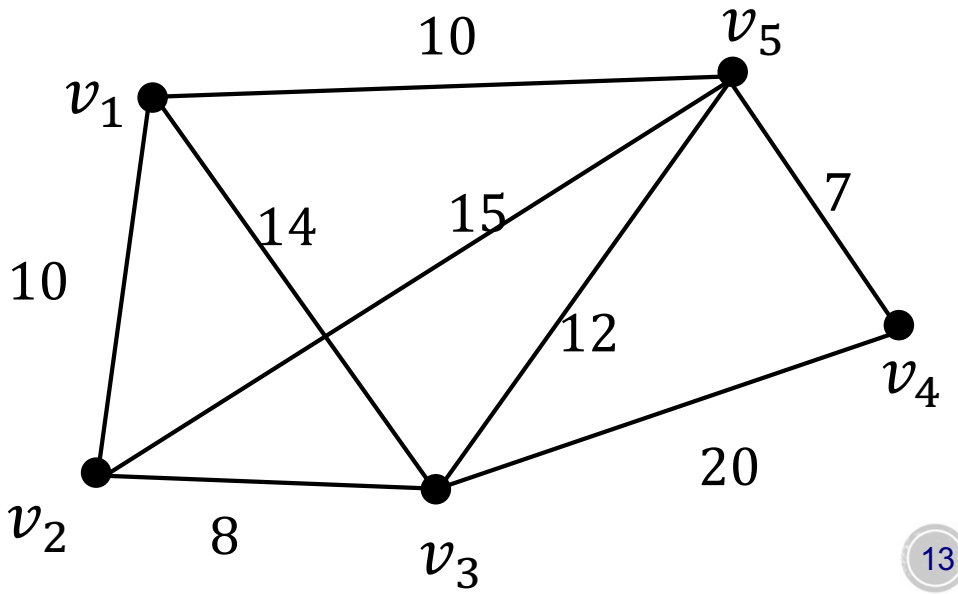
	1	2	3	4	5
1		10	14		
2	10				15
3	14				
4					
5		15			



Representing Undirected Graphs

- Adjacent matrix A

	1	2	3	4	5
1		10	14		10
2	10		8		15
3	14	8		20	12
4			20		7
5	10	15	12	7	

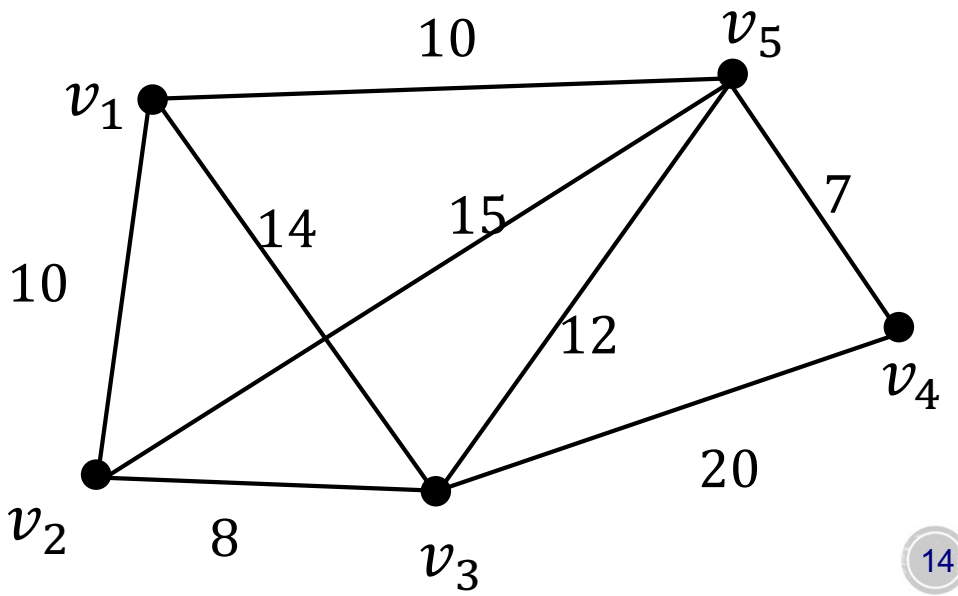


Representing Undirected Graphs

- Adjacent matrix A

	1	2	3	4	5
1	0	10	14	0	10
2	10	0	8	0	15
3	14	8	0	20	12
4	0	0	20	0	7
5	10	15	12	7	0

For sparse graphs (i.e., each node has only a few neighbors), can use linked list to save space

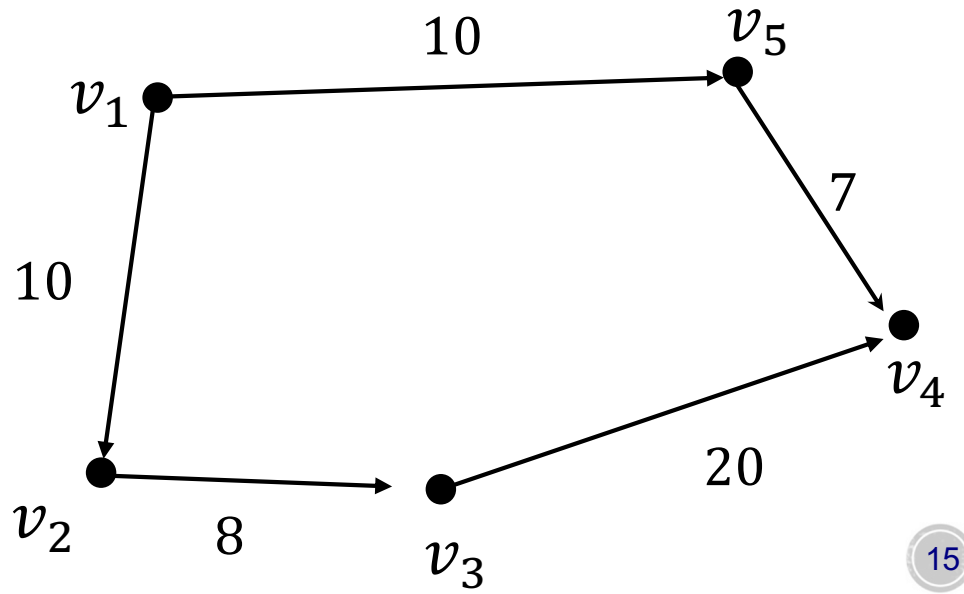


Representing Directed Graphs

- Adjacent matrix A

	1	2	3	4	5
1					
2					
3					
4					
5					

Not symmetric any more

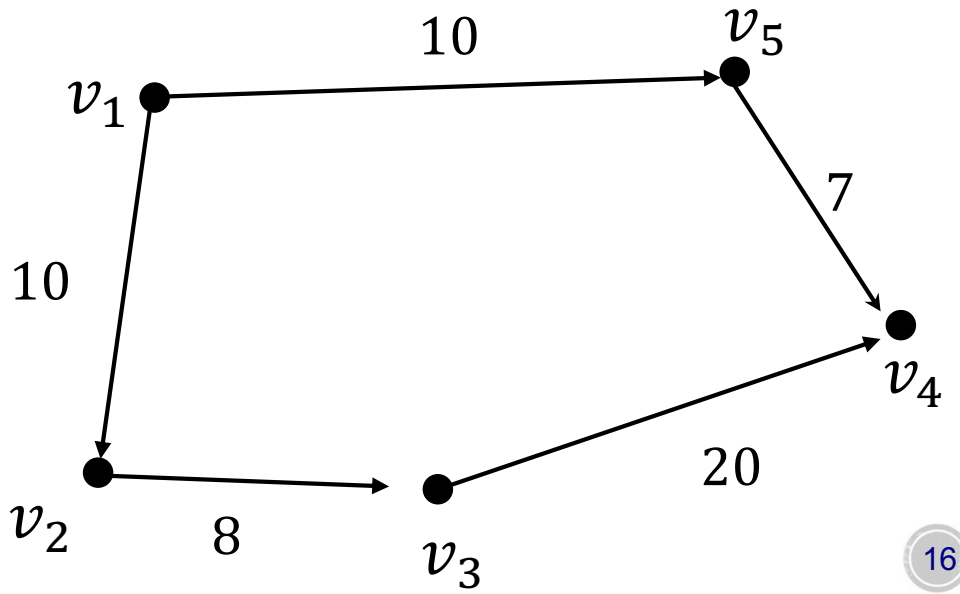


Representing Directed Graphs

- Adjacent matrix A

	1	2	3	4	5
1		10			
2	0				
3					
4					
5					

Not symmetric any more

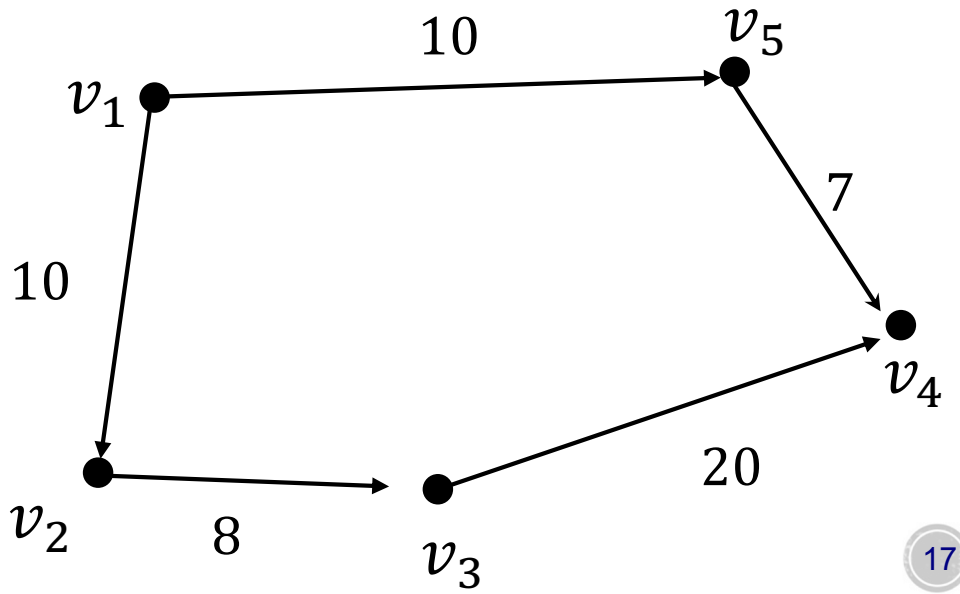


Representing Directed Graphs

- Adjacent matrix A

	1	2	3	4	5
1		10			10
2	0		8		
3				20	
4					
5				7	

Not symmetric any more

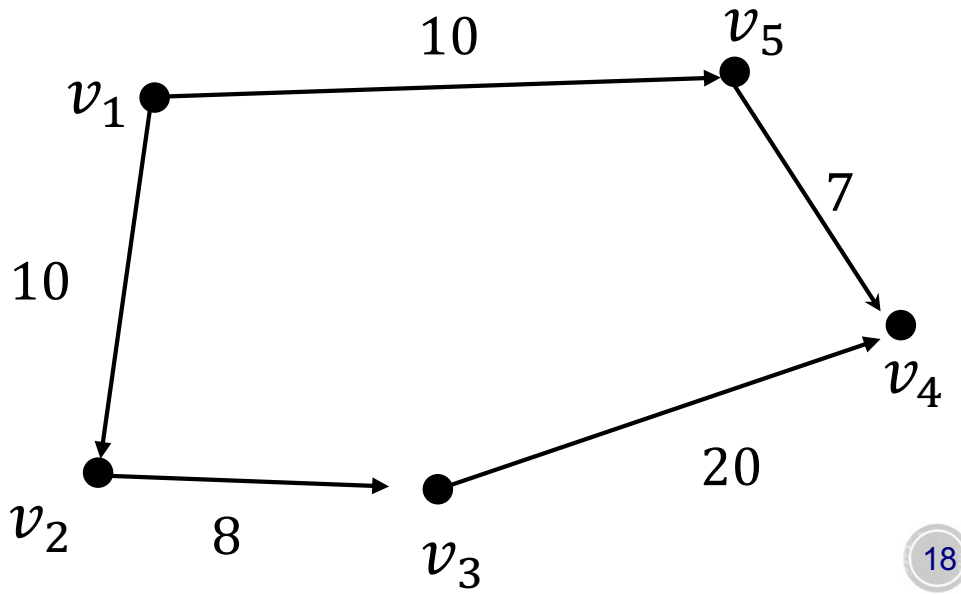


Representing Directed Graphs

- Adjacent matrix A

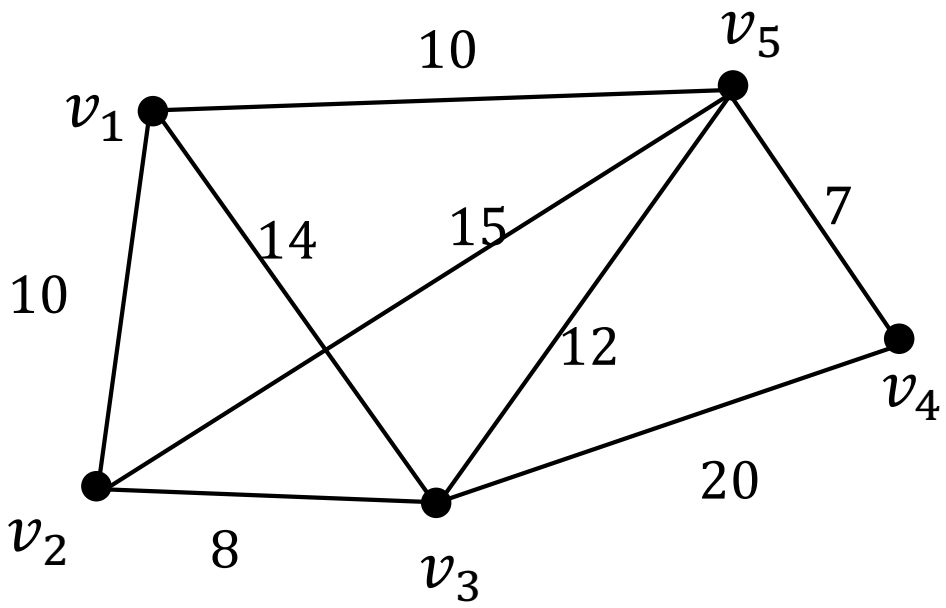
	1	2	3	4	5
1	0	10	0	0	10
2	0	0	8	0	0
3	0	0	0	20	0
4	0	0	0	0	0
5	0	0	0	7	0

Not symmetric any more



Recap: Graph Traversal Algorithms

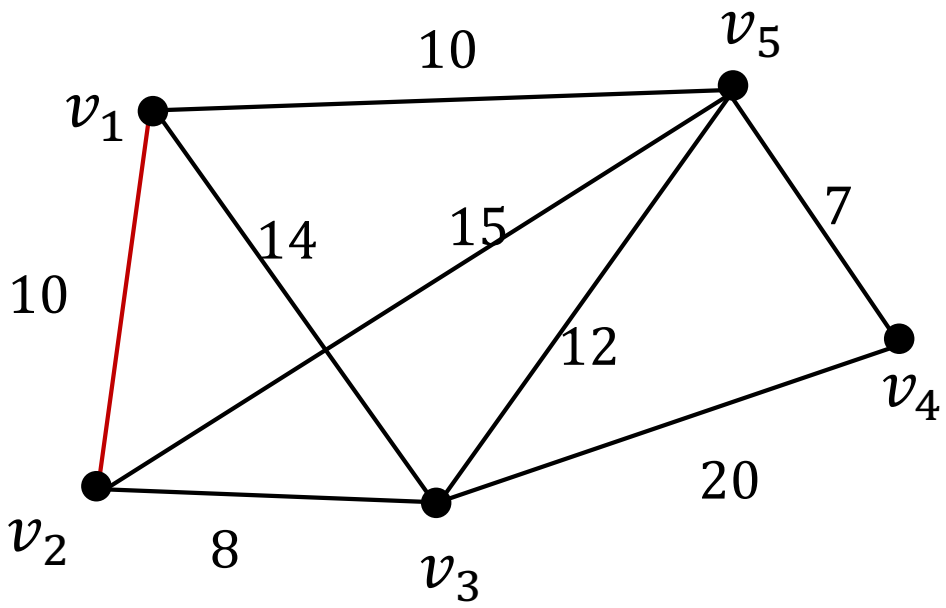
- Goal: traversing the graph
- Algorithms: depth-first search (DFS), breadth-first search (BFS), etc.



Recap: Graph Traversal Algorithms

- Goal: traversing the graph
- Algorithms: depth-first search (DFS), breadth-first search (BFS), etc.

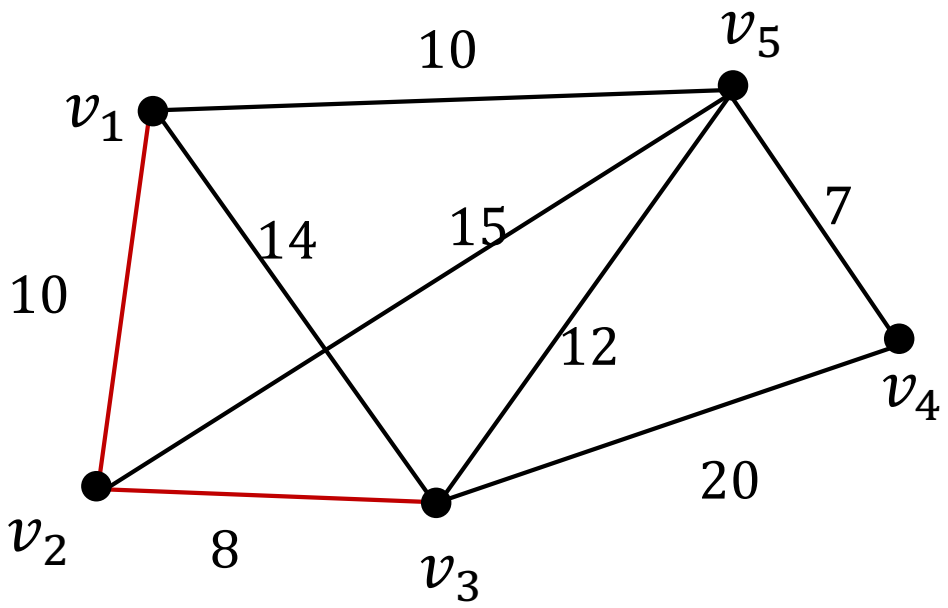
DFS:



Recap: Graph Traversal Algorithms

- Goal: traversing the graph
- Algorithms: depth-first search (DFS), breadth-first search (BFS), etc.

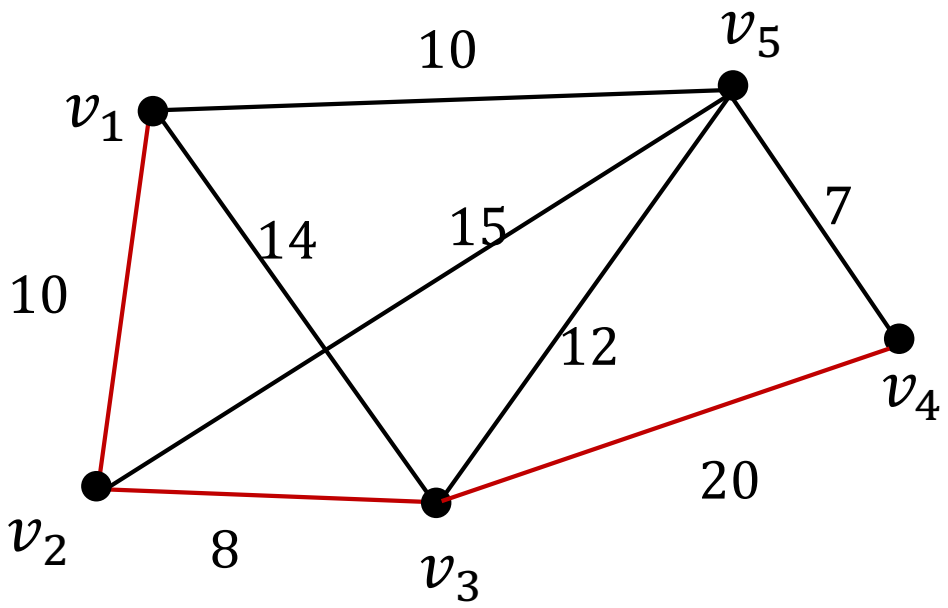
DFS:



Recap: Graph Traversal Algorithms

- Goal: traversing the graph
- Algorithms: depth-first search (DFS), breadth-first search (BFS), etc.

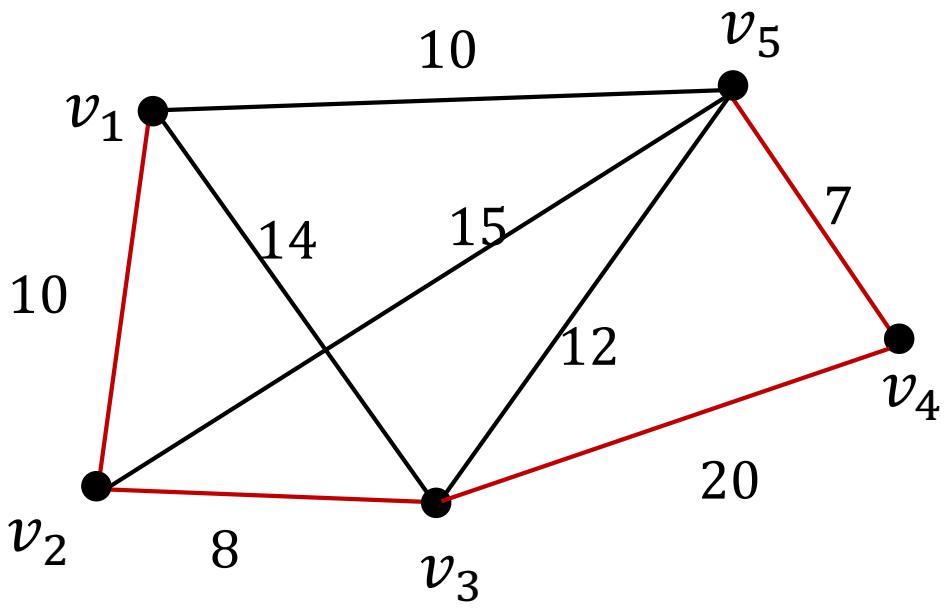
DFS:



Recap: Graph Traversal Algorithms

- Goal: traversing the graph
- Algorithms: depth-first search (DFS), breadth-first search (BFS), etc.

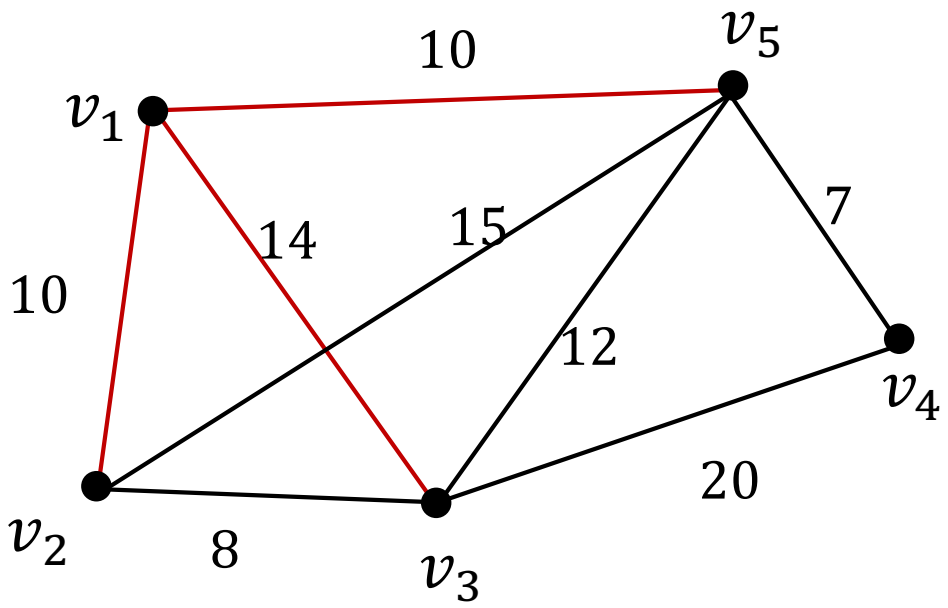
DFS:



Recap: Graph Traversal Algorithms

- Goal: traversing the graph
- Algorithms: depth-first search (DFS), breadth-first search (BFS), etc.

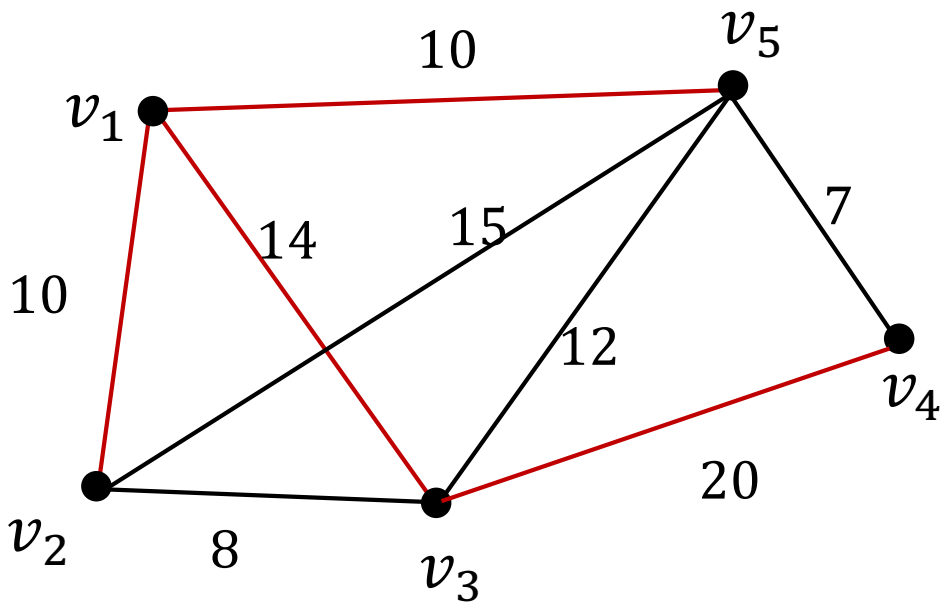
BFS:



Recap: Graph Traversal Algorithms

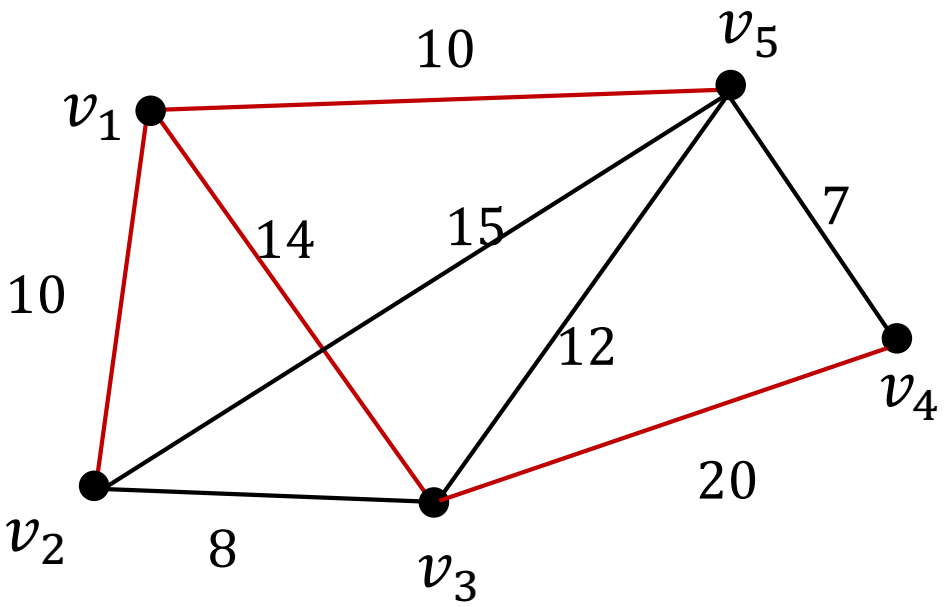
- Goal: traversing the graph
- Algorithms: depth-first search (DFS), breadth-first search (BFS), etc.

BFS:



Recap: Graph Traversal Algorithms

- Goal: traversing the graph
- Algorithms: depth-first search (DFS), breadth-first search (BFS), etc.
- These are important basic algorithms that are used as building blocks for many problems
  - E.g., checking connectivity, finding topological order, etc.
  - Do some readings if you are not familiar with these concepts



Outline

- Warm-up for Graph Algorithms
- Minimum Spanning Trees (MSTs)
  - A more interesting greedy algorithm

Minimum-Spanning Trees (MSTs)

Spanning Trees

A **spanning tree** of an undirected connected graph $G = (V, E)$ is a set of edges $T \subseteq E$ such that

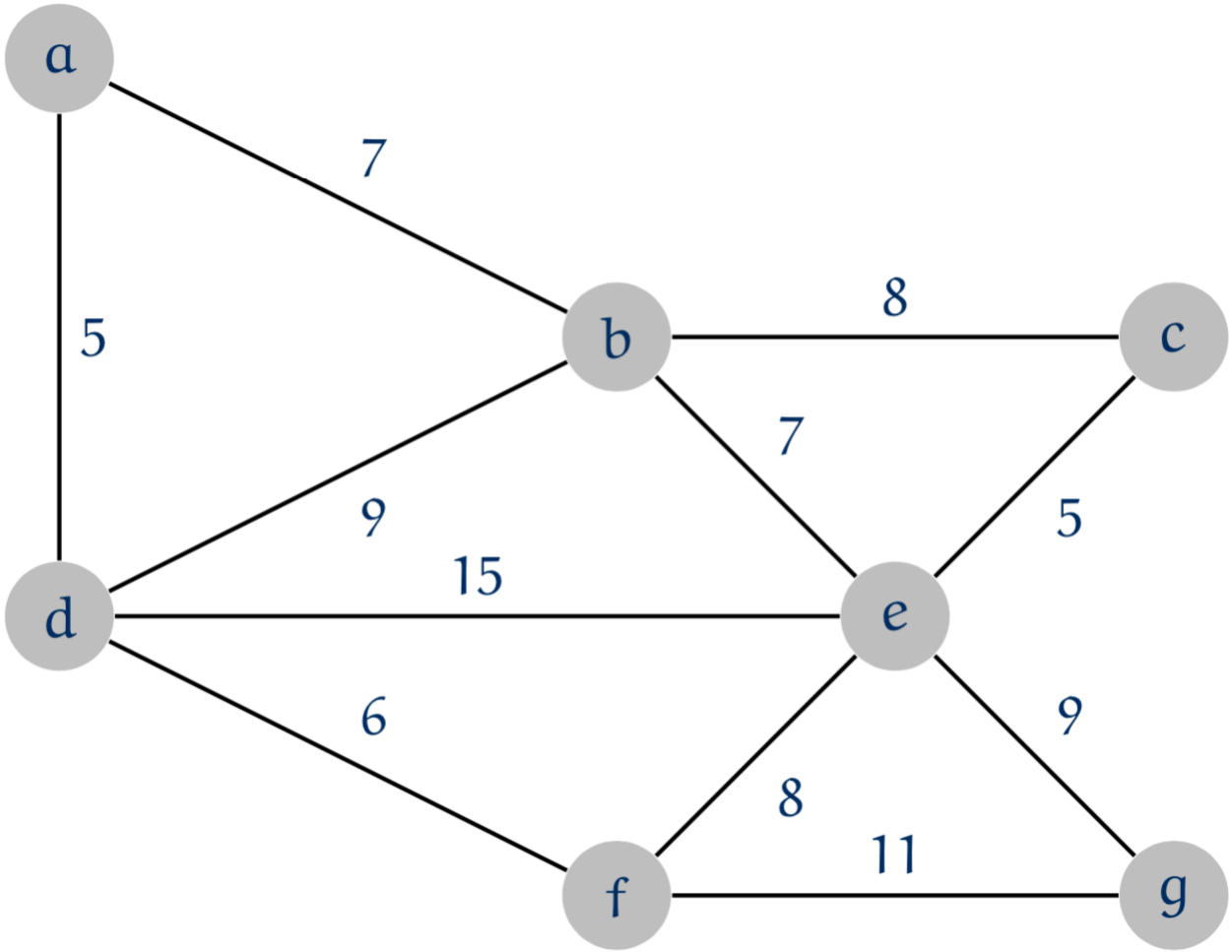
- T forms a tree, and
- (V, T) is connected.

In a weighted graph G, a **minimum spanning tree** is a spanning tree with smallest sum of edge weights

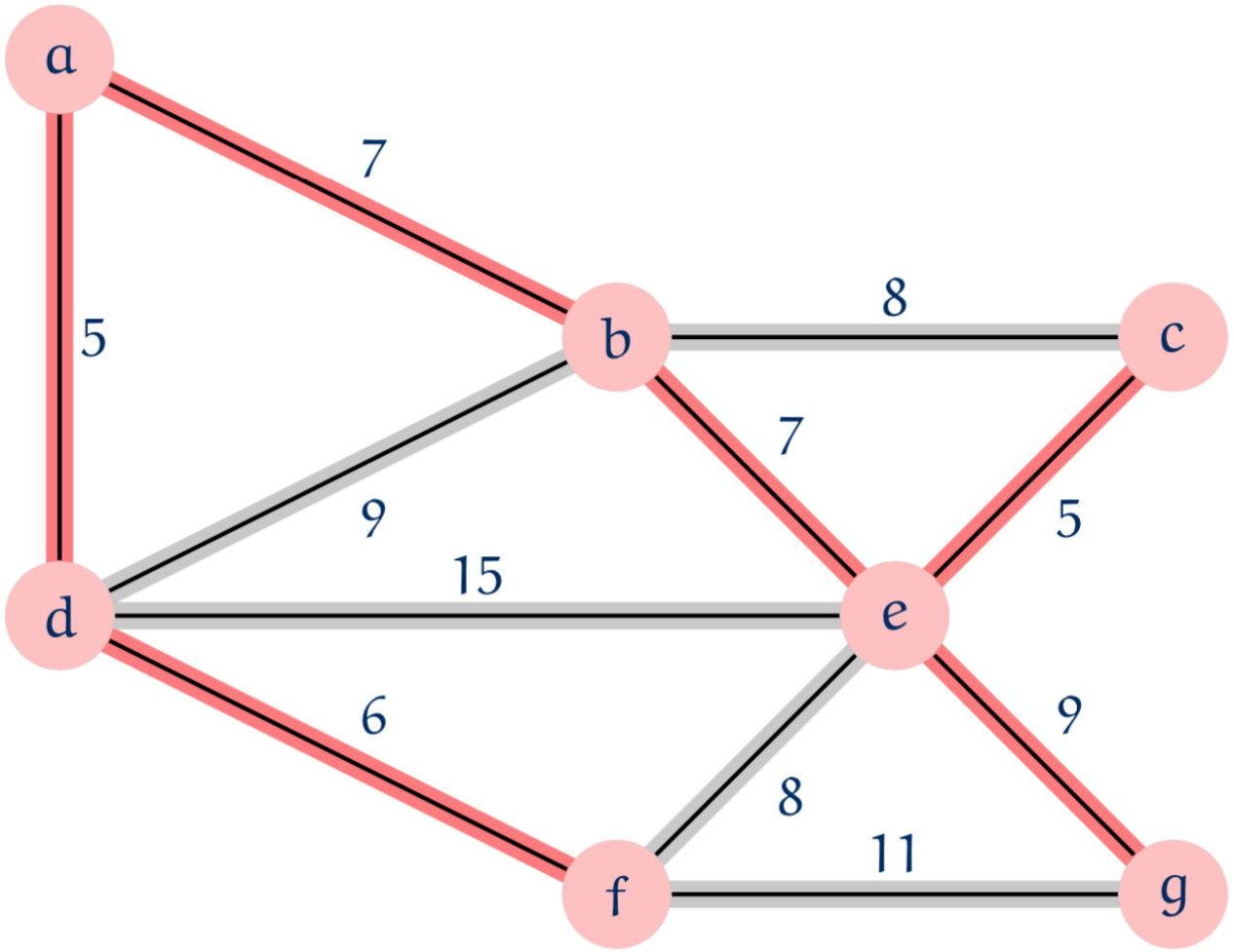
The MST problem

Find a minimum spanning tree of a weighted graph G

MST: An Example



MST: An Example



Applications

A basic and fundamental problem in graph theory, often used to measure costs to establish connections between vertices.

- Network design (e.g. telephone or cable networks): connect nodes with minimum cost
- Approximation algorithm: can be used to approximate other hard problems such as Traveling Salesman Problem, Steiner Trees, etc.
- Also an interesting application of greedy algorithms

Greedy Framework

```
SomeGreedyMSTAlgo(G)
1
2
3
4
5
```

Greedy Framework

```
SomeGreedyMSTAlgo(G)
1    Initialize T = ∅;    // T will store edges of MST
2
3
4
5
```

Greedy Framework

```
SomeGreedyMSTAlgo(G)
1    Initialize T = ∅;    // T will store edges of MST
2    while T is not a spanning tree of G
3        choose e ∈ E that "improves" solution;
4        add e to T
5
```

Greedy Framework

```
SomeGreedyMSTAlgo(G)
1   Initialize T = ∅;    // T will store edges of MST
2   while T is not a spanning tree of G
3       choose e ∈ E that "improves" solution;
4       add e to T
5   return T
```

Q: which edges, and in what order, to be added to T?

- Different ways will lead to different MST algorithms

Kruskal's MST Algorithm

- Pick edges in increasing order of their costs, and add edges to T as long as they do not form a cycle

Kruskal's MST Algorithm

- Pick edges in increasing order of their costs, and add edges to T as long as they do not form a cycle

Instantiated Algorithm

```
KruskalMST(G)
1    Initialize T = ∅;    // T will store edges of MST
2    while T is not a spanning tree of G
3
4
5
6
7    return T
```

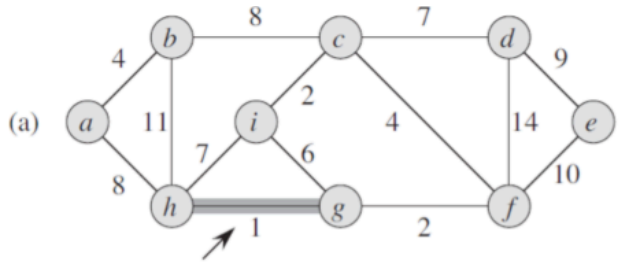
Kruskal's MST Algorithm

- Pick edges in increasing order of their costs, and add edges to T as long as they do not form a cycle

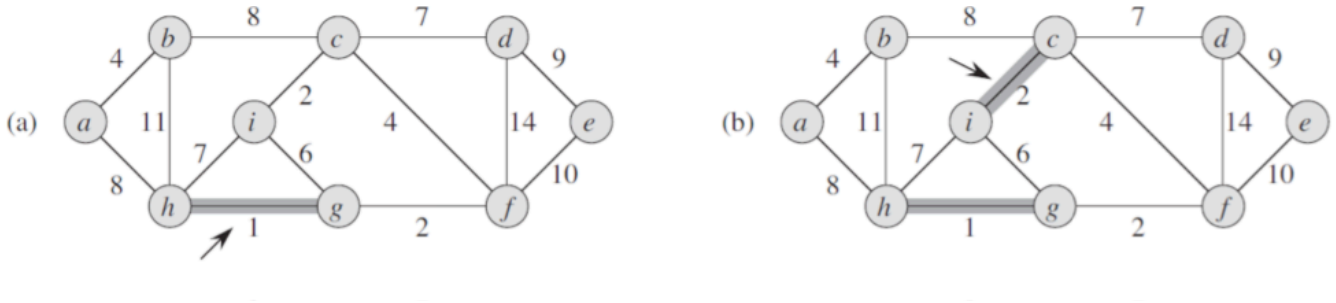
Instantiated Algorithm

```
KruskalMST(G)
1    Initialize T = ∅;    // T will store edges of MST
2    while T is not a spanning tree of G
3        choose e ∈ E of minimum cost;
4        remove e from E;
5        if T ∪ {e} does not contain cycles then
6            add e to T;
7    return T
```

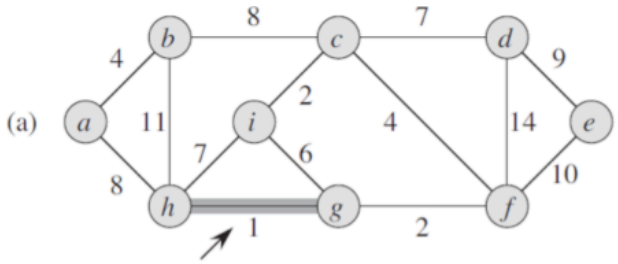
Kruskal's Algorithm: Example Execution

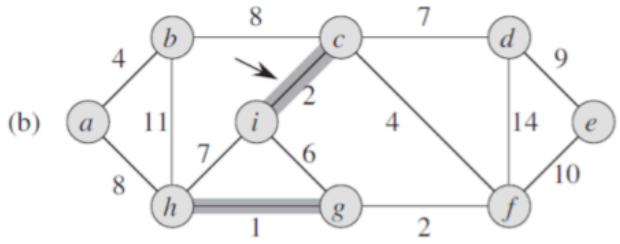


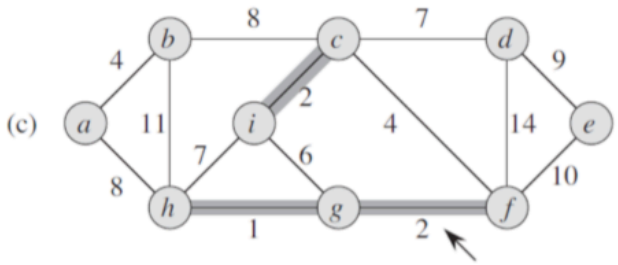
Kruskal's Algorithm: Example Execution



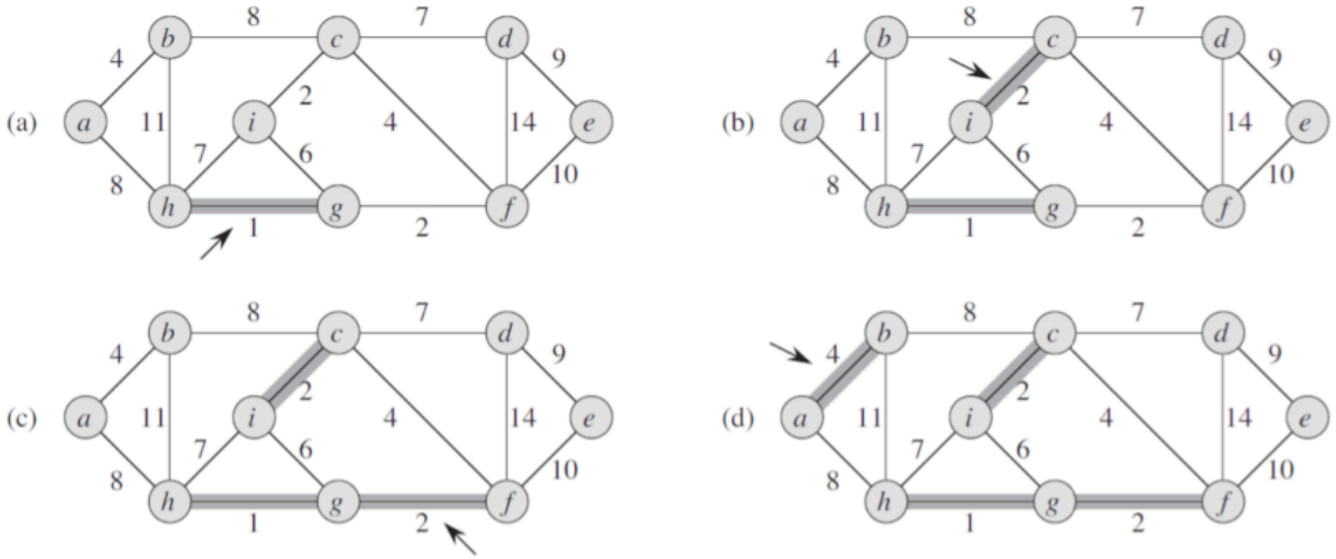
Kruskal's Algorithm: Example Execution



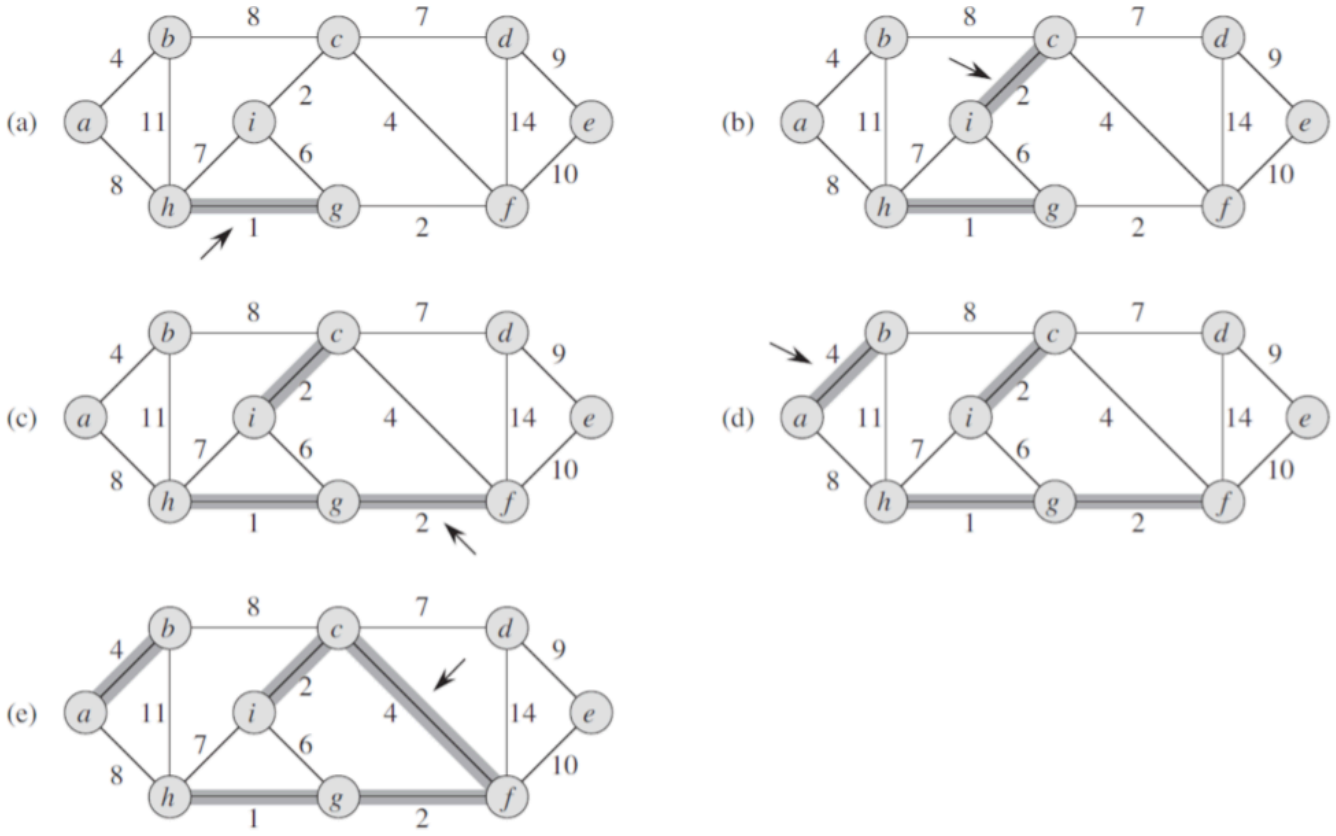




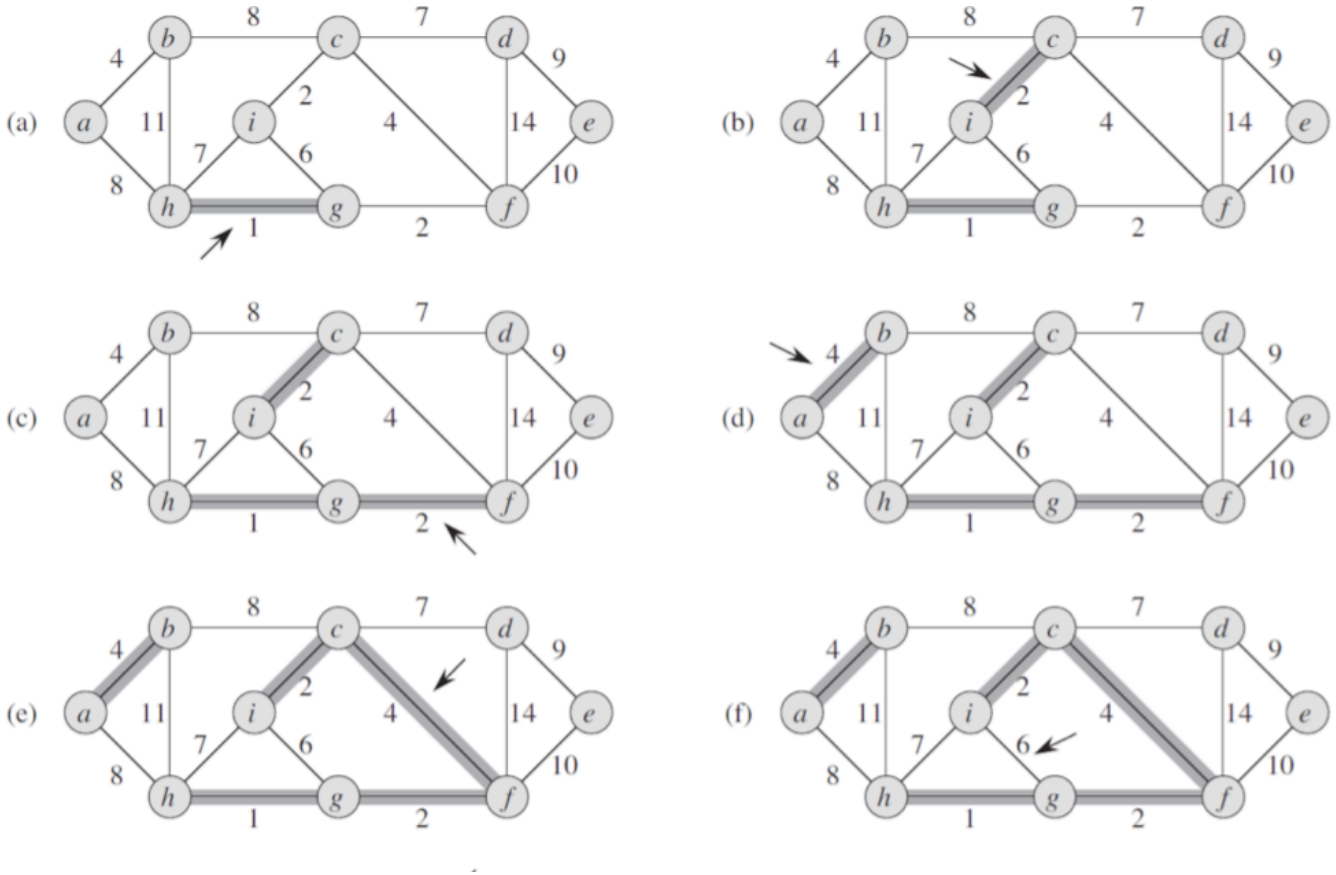
Kruskal's Algorithm: Example Execution



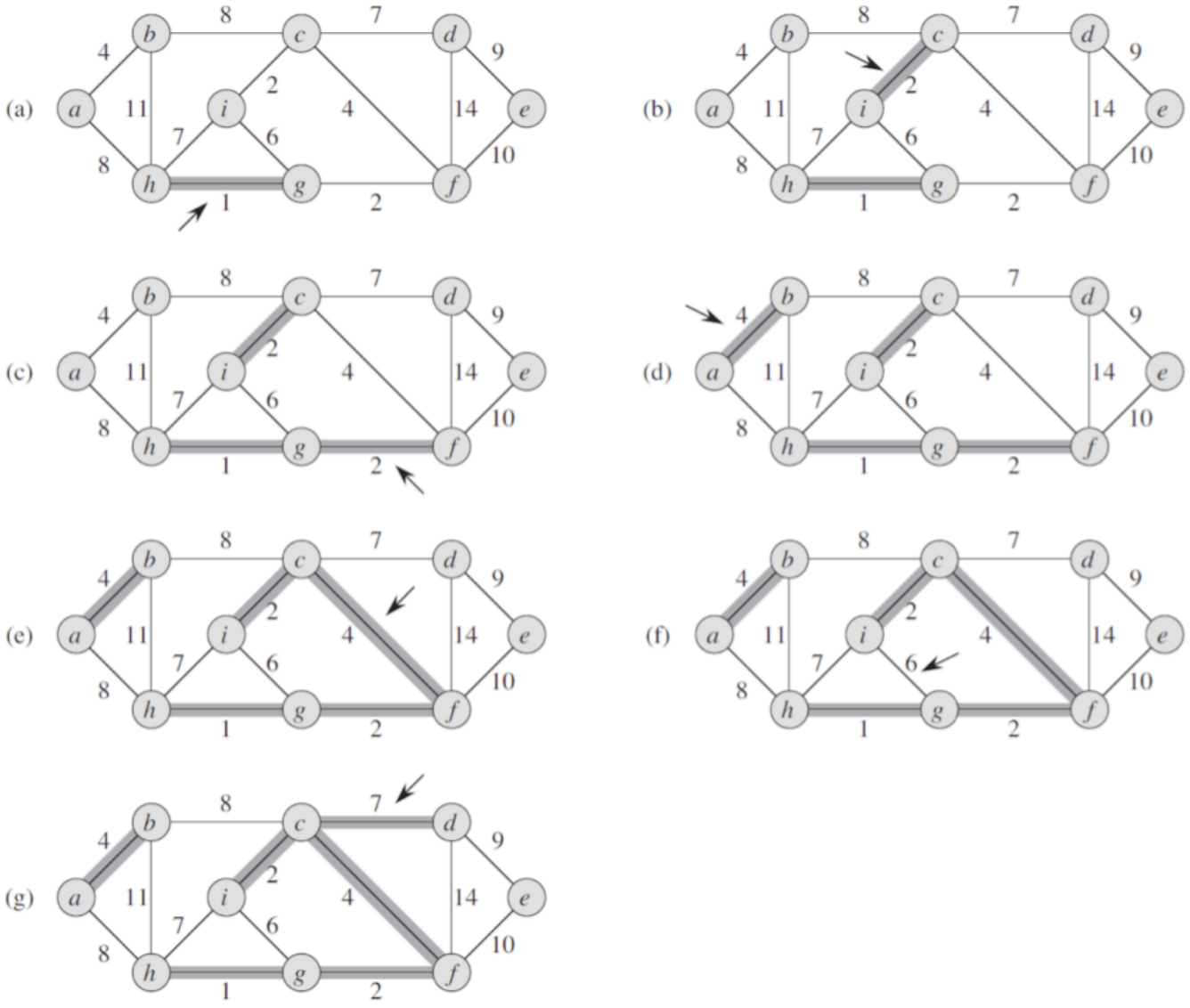
Kruskal's Algorithm: Example Execution



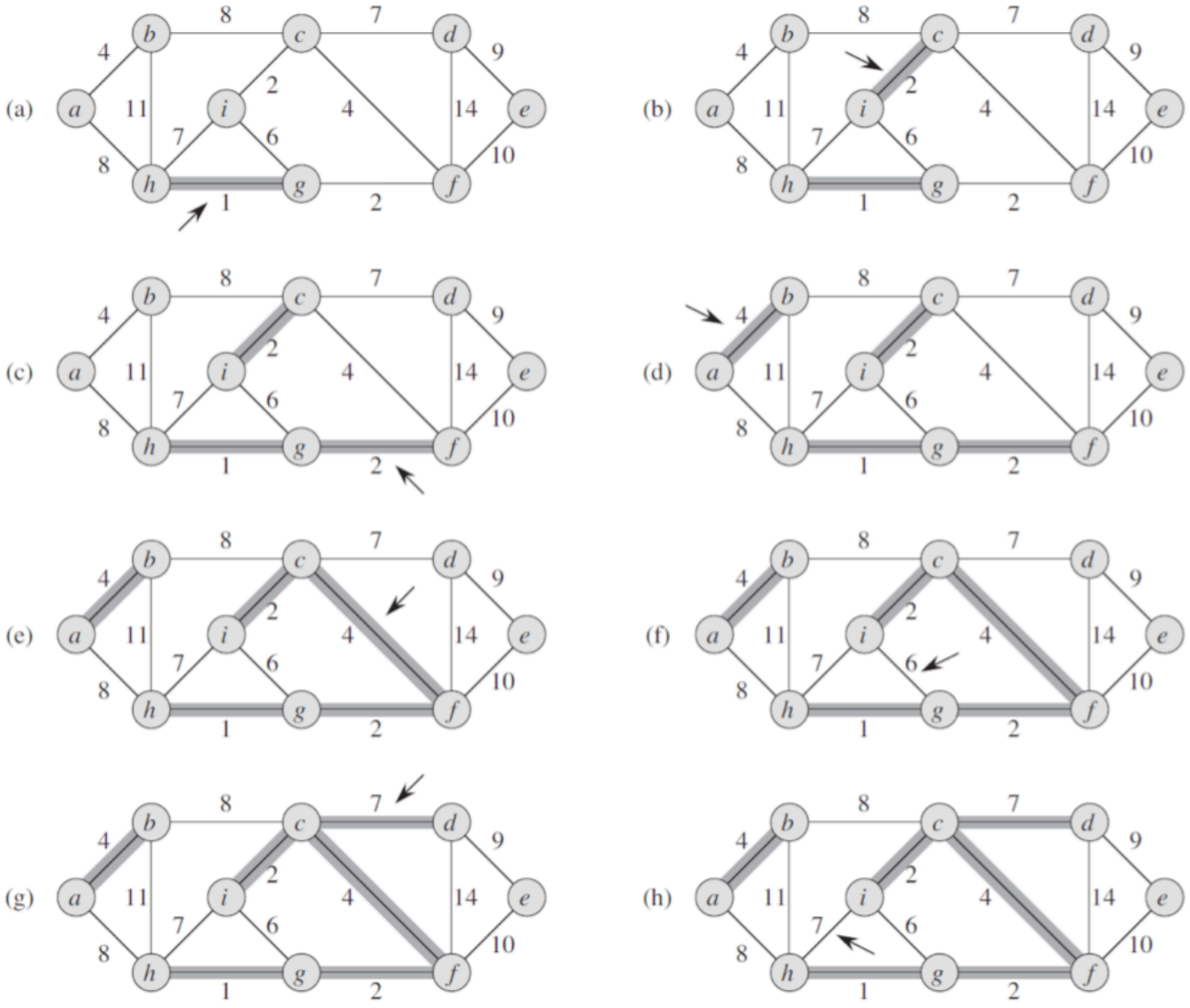
Kruskal's Algorithm: Example Execution



Kruskal's Algorithm: Example Execution

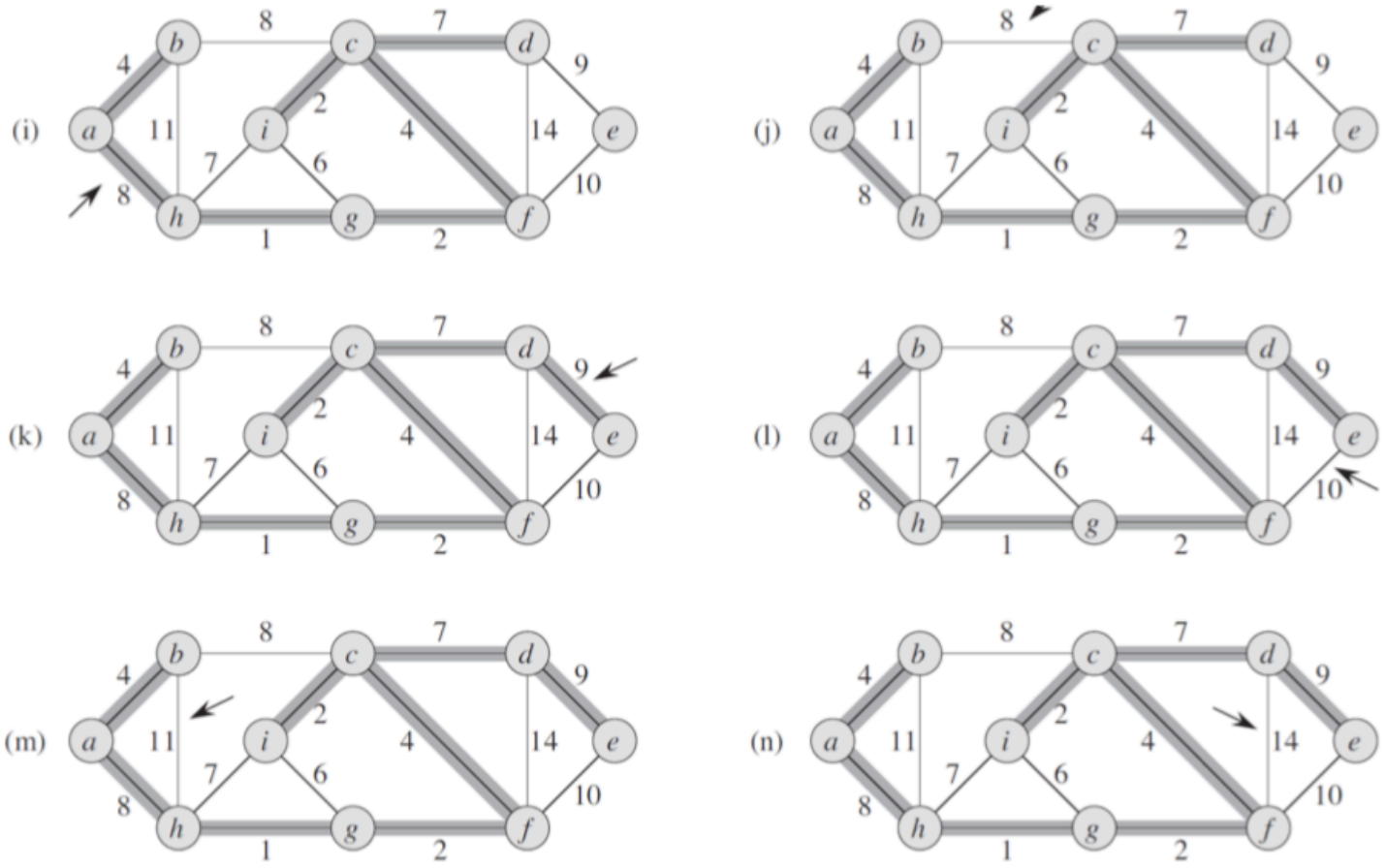


Kruskal's Algorithm: Example Execution



Kruskal's Algorithm: Example Execution

Similar remaining steps….



Correctness

- Indeed making greedy choices at each step
- Correctness proof relies on a key property of MSTs
  - Its proof needs some new graph notions we introduce next

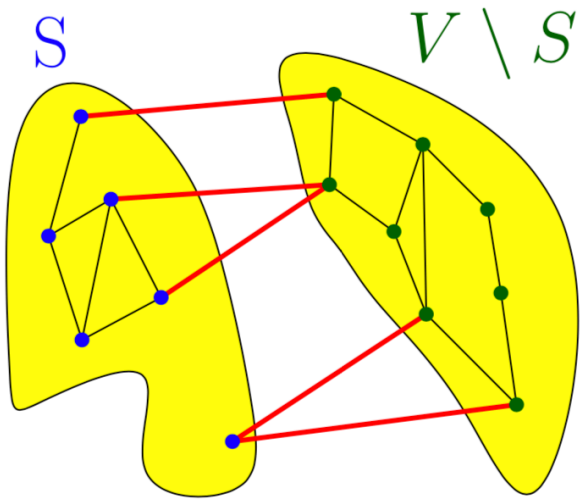
```
KruskalMST(G)
1    Initialize T = ∅;    // T will store edges of MST
2    while T is not a spanning tree of G
3        choose e ∈ E of minimum cost;
4        remove e from E;
5        if T ∪ {e} does not contain cycles then
6            add e to T;
7    return T
```

Cut

Cut

A cut of a graph $G = (V, E)$ is a partition of vertices into two disjoint sets $(S, V \setminus S)$

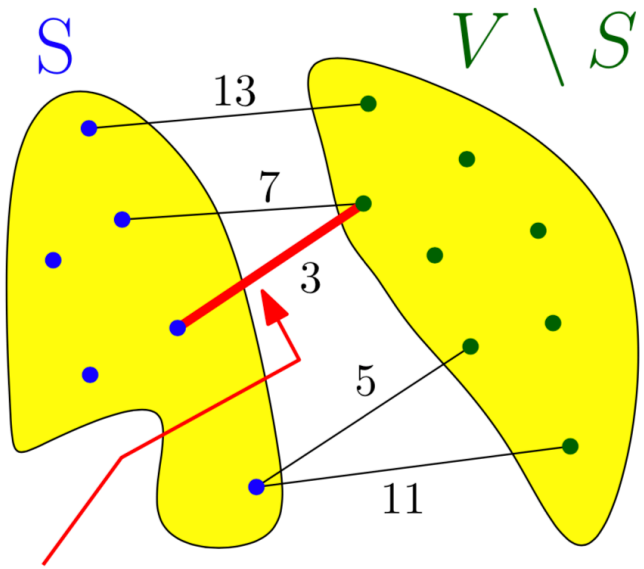
Edges that have one endpoint in each subset of the partition are the edges of the cut, and these edges are said to cross the cut.



Safe Edges

Safe Edges

An edge e is a safe edge if there exists a cut $(S, V \setminus S)$ such that e is the unique minimum cost edge crossing this cut.



Relation Between MST and Safe Edges

Theorem: Given an undirected connected graph G with distinct edge costs, the set of its safe edges form the unique MST of G.

Proof is a consequence of two lemmas:

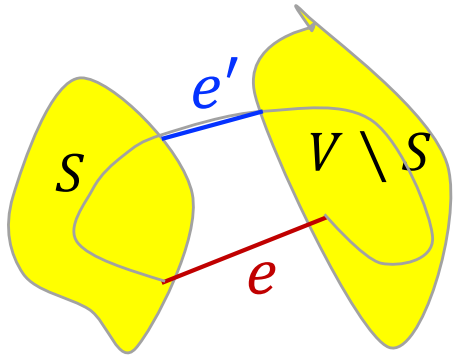
- **Lem 1**: Let e be any safe edge, then every MST contains e
- **Lem 2**: The set of safe edges forms a a connected graph G_s that covers all vertices in V
- **Lem 1** implies that there are at most $|V| - 1$ safe edges.
- Thus, the G_s in **Lem 2** must be the unique MST
  - It is a spanning tree since it connects all nodes with at most – in fact, exactly – $(|V| - 1)$ edges
  - It is the MST since no other spanning trees can contain all safe edges

Relation Between MST and Safe Edges

Lemma 1: Let e be any safe edge, then every MST contains e.

Proof:

- For sake of contradiction, assume e is *not* in some MST T
- Adding e to T creates a cycle C in $T \cup \{e\}$ that contains e
- e is safe ⇒ there is a cut $(S, V \setminus S)$ such that e is the unique minimum cost edge crossing this cut
- There must exist another edge $e' \in C$ crossing the cut as well
- $\{T \setminus \{e'\}\} \cup \{e\}$ is a new spanning tree of lower cost, contradiction.



Relation Between MST and Safe Edges

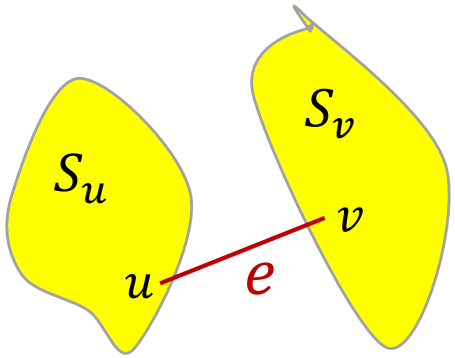
Lemma 2: The set of safe edges forms a a connected graph that covers all vertices in V.

Proof:

- Assume not. Let $S \subset V$ be a connected component in the graph induced by safe edges.
- Let e be the smallest cost edge crossing $S \Rightarrow e$ is a safe edge, contradiction.

Correctness of Kruskal's Algorithm

- Let us assume all edges have different costs (not essential)
- By previous theorem, only need to show all added edges are safe
  - **Proof**: when $e = (u, v)$ is about being added to T, let S_u and S_v be the connected components containing u and v
  - e has minimum cost among all edges forming no cycle, hence e has minimum cost among all edges crossing the cut S_v (and also S_u)
  - e is a safe edge



u, v cannot in the same component

```
KruskalMST(G)
1    Initialize T = ∅;    // T will store edges of MST
2    while T is not a spanning tree of G
3        choose e ∈ E of minimum cost;
4        remove e from E;
5        if T ∪ {e} does not contain cycles then
6            add e to T;
7    return T
```

Running Time of Kruskal's Algorithm

- While loop is executed $m = |E|$ times
- Step 5 takes $n = |V|$ time (**Q**: how to detect cycles of a graph?)
- In total $O(mn)$ time

```
KruskalMST(G)
1    Initialize T = ∅;    // T will store edges of MST
2    while T is not a spanning tree of G
3        choose e ∈ E of minimum cost;
4        remove e from E;
5        if T ∪ {e} does not contain cycles then
6            add e to T;
7    return T
```

Next Lecture

- Acceleration of Kruskal's algorithm via better data structure
- Prim's MST algorithm, and its acceleration

Thank You

Haifeng Xu

University of Virginia

hx4ad@virginia.edu