Greedy Technique

Greedy Technique

- Only applicable to optimization problems
- Constructs a solution through a sequence of steps
- Expanding the partially constructed solution
- The choice for each step should be:
  - Feasible → satisfies the problem constraints
  - Locally optimal → best local choice among all available choices
  - Irrevocable → once made, it may not be changed

Greedy Technique

- On each step do a greedy grab
- In hope of by local optimal steps, reaching to global optimal
- Intuitively appealing and simple
- Difficult to prove it always leads to optimal solutions
  1. Use of induction
  2. Show in each step it does at least as good as any other algorithm
  3. Show the final result obtained is optimal based on the result rather than the way it operates

Greedy Technique

- Greed is good or bad?
- For some yes, yields an optimal solution for every instance.
  - change making for “normal” coin denominations
  - minimum spanning tree (MST)
  - single-source shortest paths
  - simple scheduling problems
  - Huffman codes

- For most no, can be useful for fast approximations.
  - traveling salesman problem (TSP)
  - knapsack problem

Change-Making Problem

- Given unlimited amounts of coins of $d_1 > \ldots > d_m$,
- give change for amount n with the least number of coins

- Greedy approach:
  - Start with biggest coin
  - Give as much as possible
  - Go and try the next big coin
  - Repeat until remaining is 0

- Example: $d_1 = 25¢,\ d_2 = 10¢,\ d_3 = 5¢,\ d_4 = 1¢$ and $n = 41¢$
- Example: $d_1 = 25¢,\ d_2 = 10¢,\ d_3 = 5¢,\ d_4 = 1¢$, and $n = 30¢$

Knight-Moving Problem

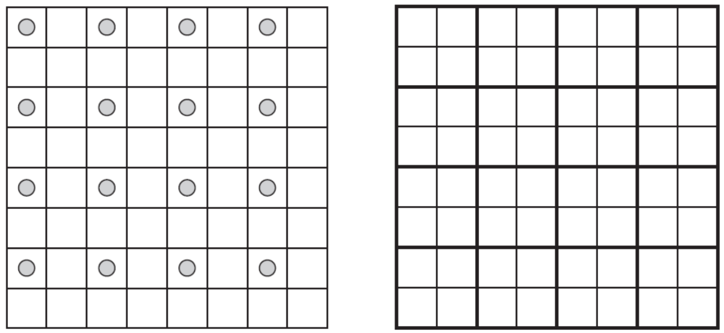
- A knight wants to move from top-left corner to bottom-right corner in an 100×100 board
- give the minimum number of moves needed

- Greedy approach:
  - Jump as close to the goal as possible
  - From (1,1) to (100,100)
  - It needs 66 moves $(1,1) - (3,2) - (4,4) - \cdots - (97,97) - (99,98) - (100,100)$

- Optimal? Yes
- Why?
  - Manhattan distance = rows difference + columns difference)
  - Greedy algorithm decreases the distance by 3 in each move
  - The best a knight can do

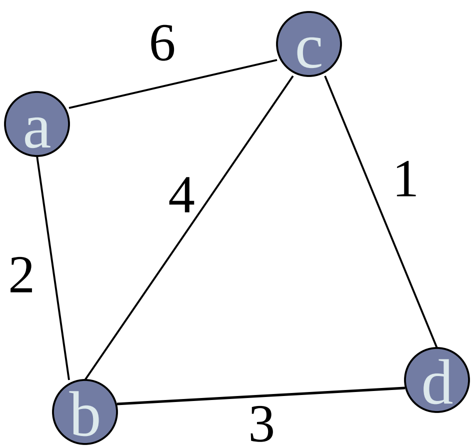
Chips-Placing Problem

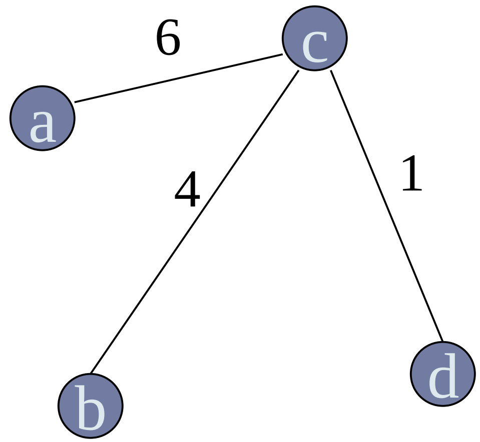
- Place maximum number of chips on an 8×8 board
- Can not be vertically, horizontally or diagonally adjacent
-
- Greedy approach:
  - place each new chip so as to leave as many available cells as possible for next chip
  - Put chips in add columns and odd rows
  - We can get 16 chips on it
- Optimal? Yes
- Why?
  - Break board into 4×4 squares
  - Impossible to put more than 1 chip in each
  - We can have 16 of such squares in an 8×8 board



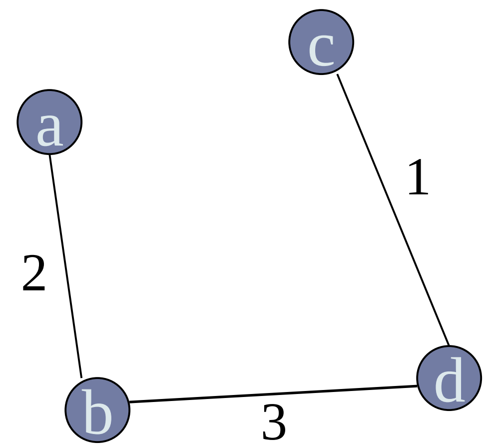
Minimum Spanning Tree (MST)

- *Spanning tree*
  - for connected, undirected graph
  - a connected acyclic subgraph that includes all of nodes
- *Minimum spanning tree*
  - For weighted, connected, undirected graph
  - a spanning tree of minimum total weight





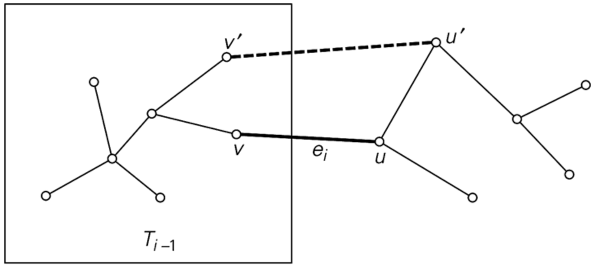
ST, weight 11



ST, weight 6

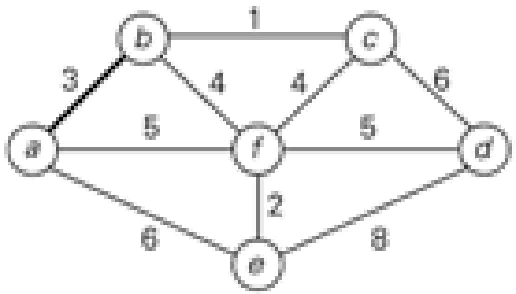
Prim's MST algorithm

- Start with tree T_1 consisting of one (any) node
- "grow" tree one vertex at a time to produce MST
- through a series of expanding subtrees $T_1, T_2, \ldots, T_n$



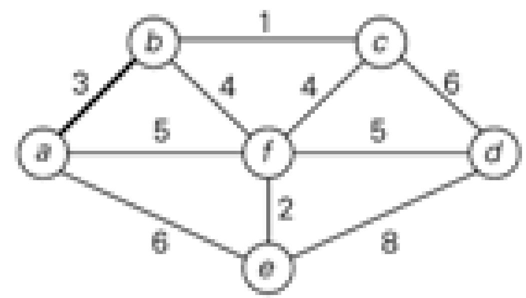
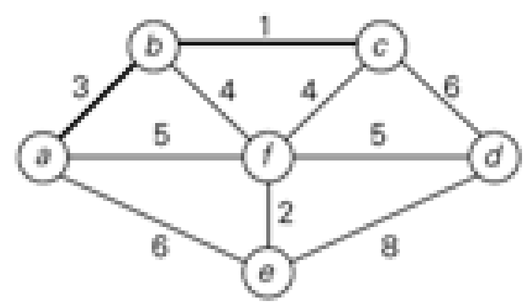
- On each iteration:
  - construct T_{i+1} from T_i
  - by adding a node not in T_i
  - that is closest to those already in T_i (this is a "greedy" step!)
- Stop when all nodes are included

Prim’s MST algorithm



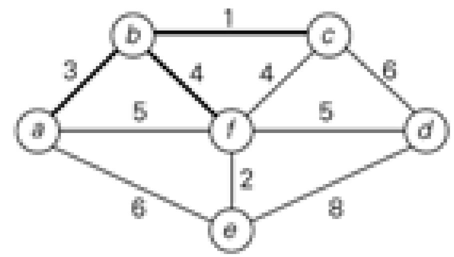
Prim's MST algorithm

priority queue

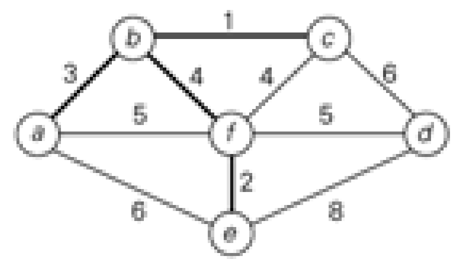
Tree vertices	Remaining vertices	Illustration
a(−, −)	**b(a, 3)** c(−, ∞) d(−, ∞) e(a, 6) f(a, 5)	
b(a, 3)	**c(b, 1)** d(−, ∞) e(a, 6) f(b, 4)	

Prim’s MST algorithm

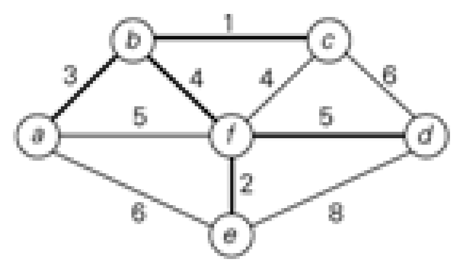
c(b, 1)   d(c, 6) e(a, 6) **f(b, 4)**



f(b, 4)   d(f, 5) **e(f, 2)**



e(f, 2)   **d(f, 5)**



d(f, 5)

Prim's: Implementation 1

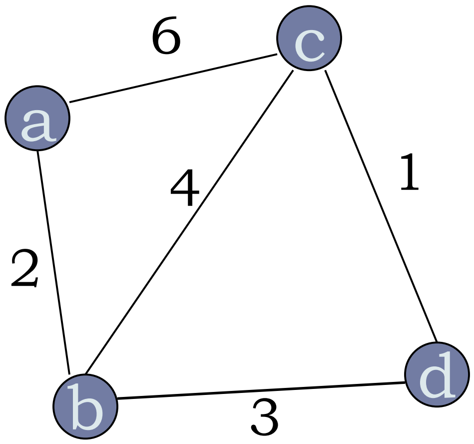
- Graph is adjacency matrix
- Priority queue is unordered array
- Each step: traverse array
  - remove the u_i (vertex just added)
  - update priorities of remaining vertices
- $\Theta(|V|^2)$

Prim's: Implementation 2

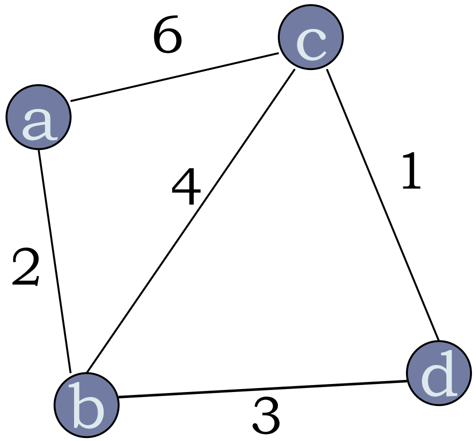
- Graph is adjacency list
- Priority queue is a min-heap
- Each step: update heap (log(n) operations)
  - remove top element u_i (vertex just added)
  - update priorities of remaining vertices

- $O(|V|-1+|E|)(log|V|) = O(|E|log|V|)$

Prim's MST algorithm

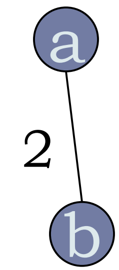


Prim’s MST algorithm

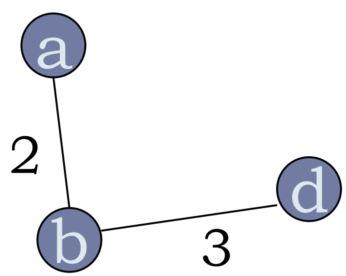


T_1 is
any node

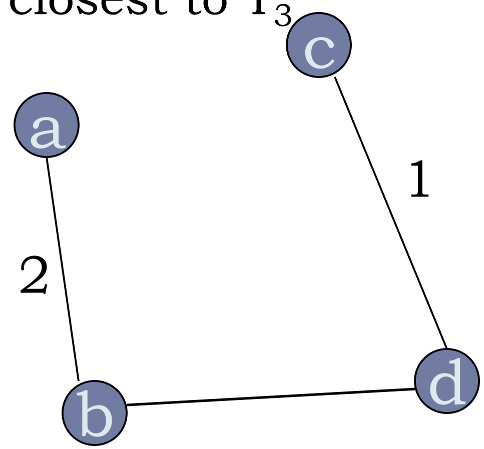
T_2 adds node
closest to T_1



T_3 adds node
closest to T_2



T_4 adds node
closest to T_3



3

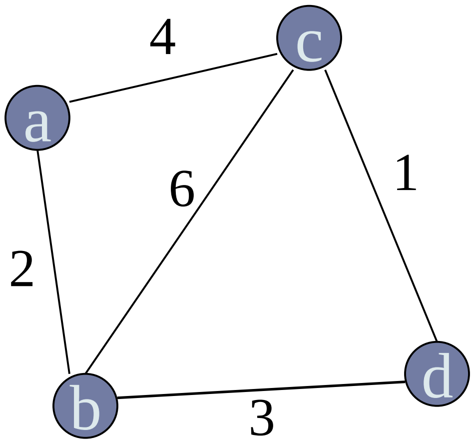
Kruskal's MST algorithm

- Sort the edges in non-decreasing order of lengths
- "Grow" tree one edge at a time to produce MST through a series of expanding forests $F_1, F_2, \ldots, F_{n-1}$
- On each iteration:
  - add the next edge on the sorted list
  - unless this would create a cycle
  - Repeat until you have all the nodes covered

Notes about Kruskal's algorithm

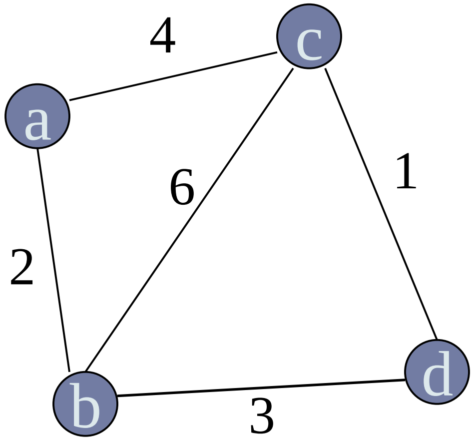
- Algorithm looks easier than Prim's but is harder to implement (checking for cycles!)
- Cycle checking: a cycle is created iff added edge connects vertices in the same connected component
- *Union-find* algorithms

Kruskal's MST algorithm



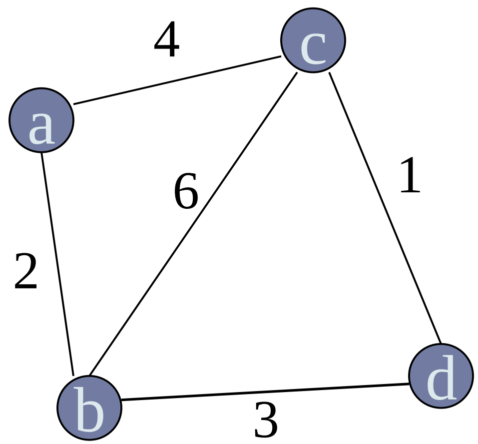
$(a, c, 4), (a, b, 2), (b, c, 6), (b, d, 3), (d, c, 1)$

Kruskal's MST algorithm

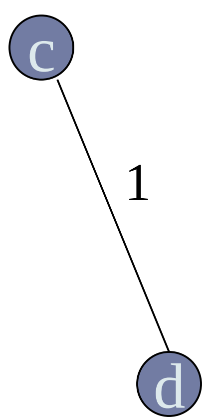


$(a, c, 4), (a, b, 2), (b, c, 6), (b, d, 3), (d, c, 1)$
$(d, c, 1), (a, b, 2), (b, d, 3), (a, c, 4), (b, c, 6)$

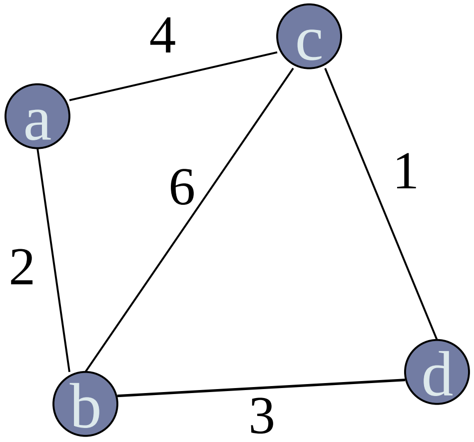
Kruskal's MST algorithm



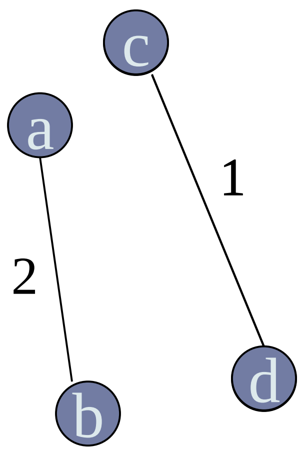
$(a, c, 4), (a, b, 2), (b, c, 6), (b, d, 3), (d, c, 1)$
$(a, b, 2), (b, d, 3), (a, c, 4), (b, c, 6)$



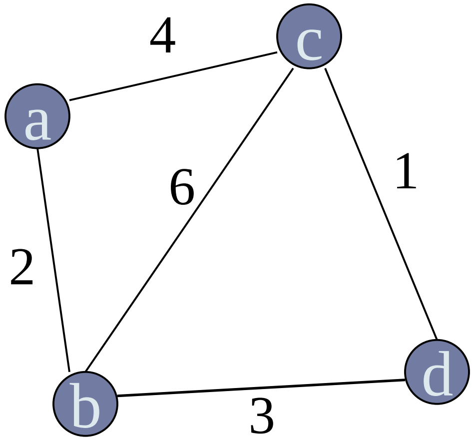
Kruskal's MST algorithm



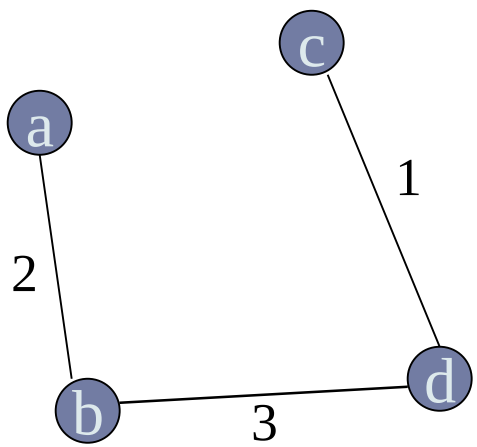
$(a, c, 4), (a, b, 2), (b, c, 6), (b, d, 3), (d, c, 1)$
$(b, d, 3), (a, c, 4), (b, c, 6)$



Kruskal's MST algorithm



$(a, c, 4), (a, b, 2), (b, c, 6), (b, d, 3), (d, c, 1)$
$(a, c, 4), (b, c, 6)$



Dijkstra's algorithm

- Single Source Shortest Paths Problem:
- Given a weighted connected graph G
- Find shortest paths from source s to each of the other nodes

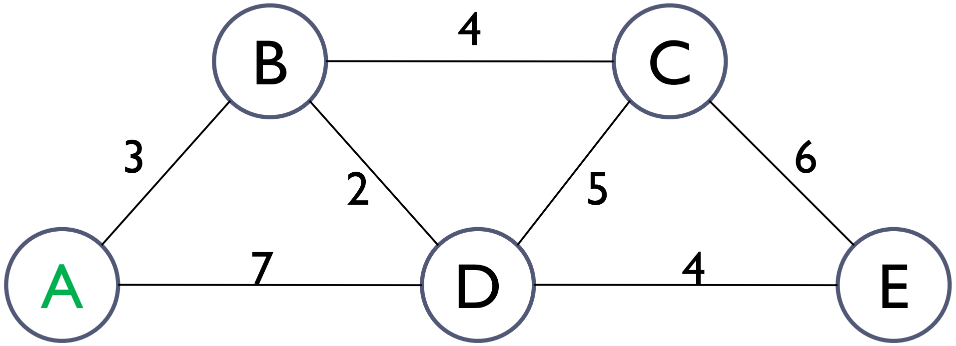
- Dijkstra's algorithm:
- Similar to Prim's MST algorithm
- Among nodes not already in the tree, add node u with

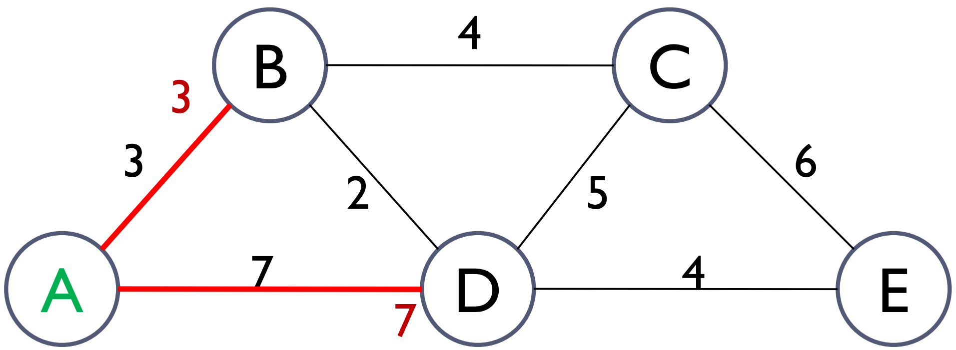
$$min\ \{d_v + w(v,u)\}$$

- where
  - d_v is the length of the shortest path from source s to v
  - $w(v,u)$ is the weight of edge from v to u
  - v has already been covered
  - u has not been covered

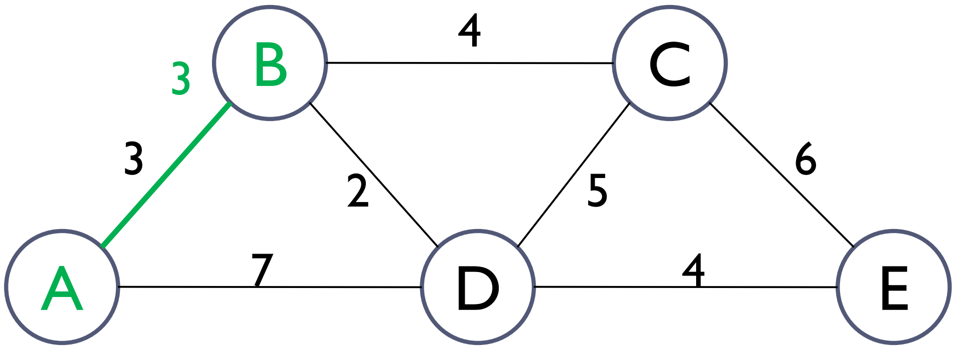
Dijkstra's algorithm

- Doesn't work for graphs with negative weights
- Applicable to both undirected and directed graphs
- Efficiency
  - $O(|V|^2)$ for graphs represented by weight matrix and array implementation of priority queue
  - $O(|E|\log|V|)$ for graphs represented by adj. lists and min-heap implementation of priority queue
- Don't mix up Dijkstra's algorithm with Prim's algorithm!

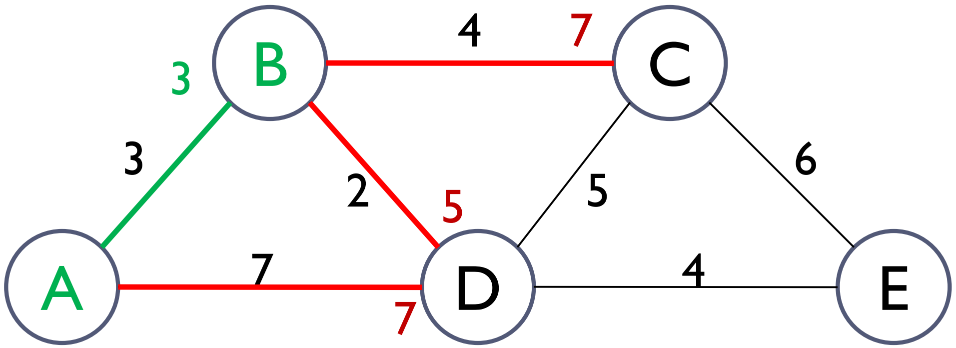

B
4
C
3
2
5
6
A
7
D
4
E



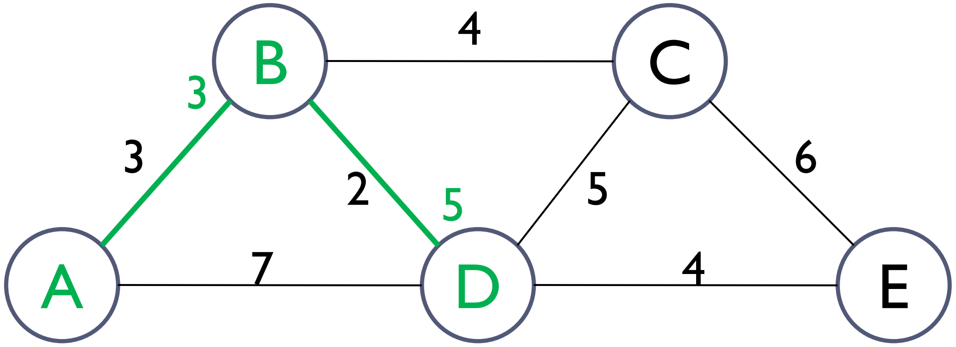
B
C
A
D
E
4
3
3
2
5
6
7
4
7

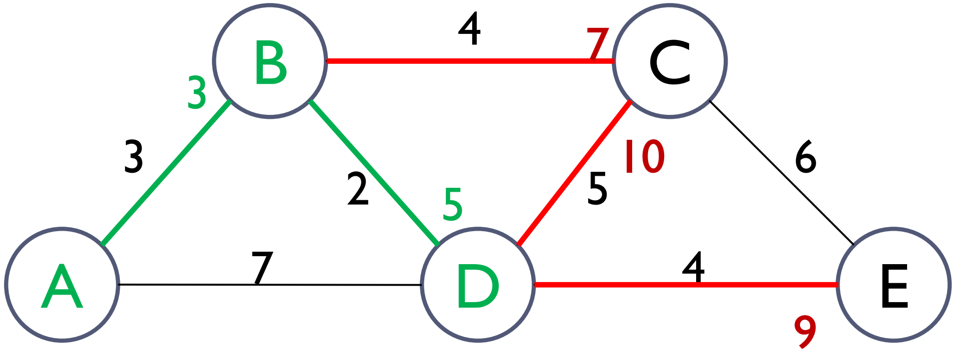


A
B
C
D
E
3
3
4
2
5
6
7
4

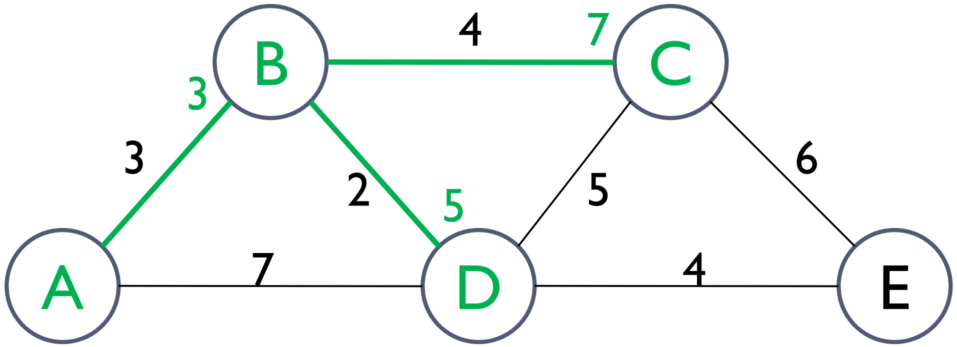


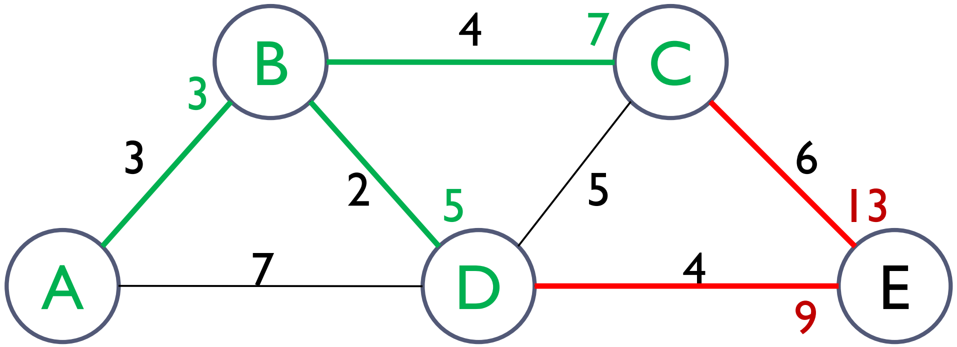
A
B
C
D
E
3
3
4
7
2
5
5
6
7
7
4


A
B
C
D
E
3
3
4
2
5
5
7
4
6

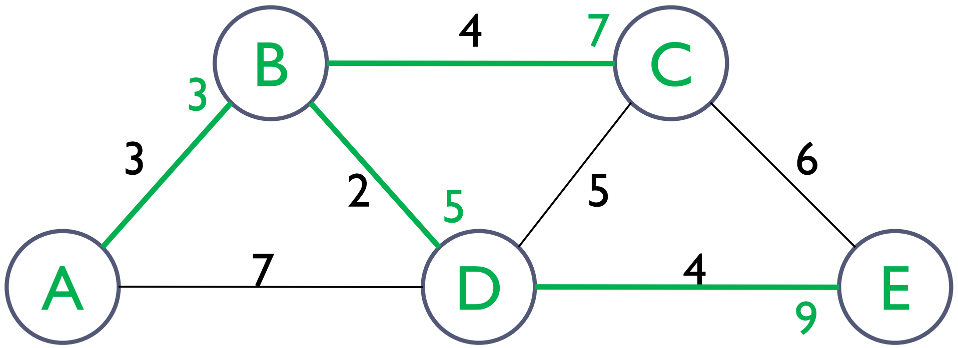


A
B
C
D
E
3
3
4
7
2
5
7
5
10
6
4
9


B
C
A
D
E
3
3
4
7
2
5
5
6
7
4

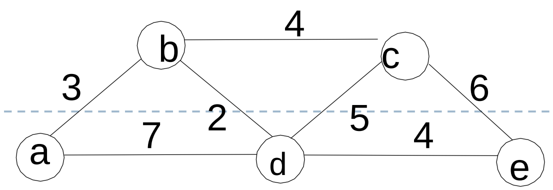


B
C
A
D
E
3
4
7
3
2
5
5
6
13
7
4
9

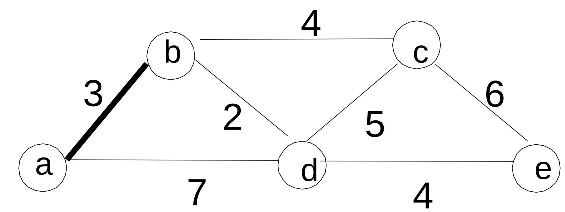


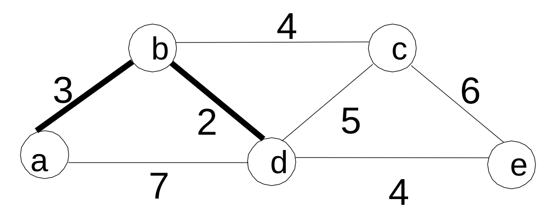
A
B
C
D
E
3
3
4
7
2
5
5
6
7
4
9

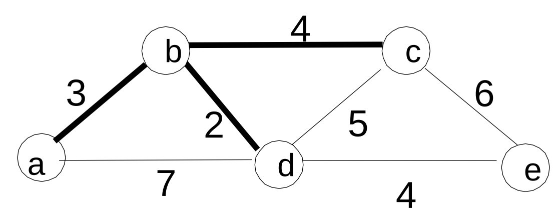
Example

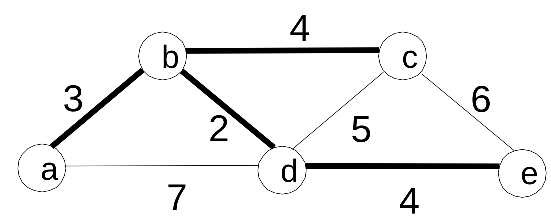


Tree vertices	Remaining vertices
a(-,0)	b(a,3) c(-,∞) d(a,7) e(-,∞)
b(a,3)	c(b,3+4) d(b,3+2) e(-,∞)
d(b,5)	c(b,7) e(d,5+4)
c(b,7)	e(d,9)
e(d,9)	





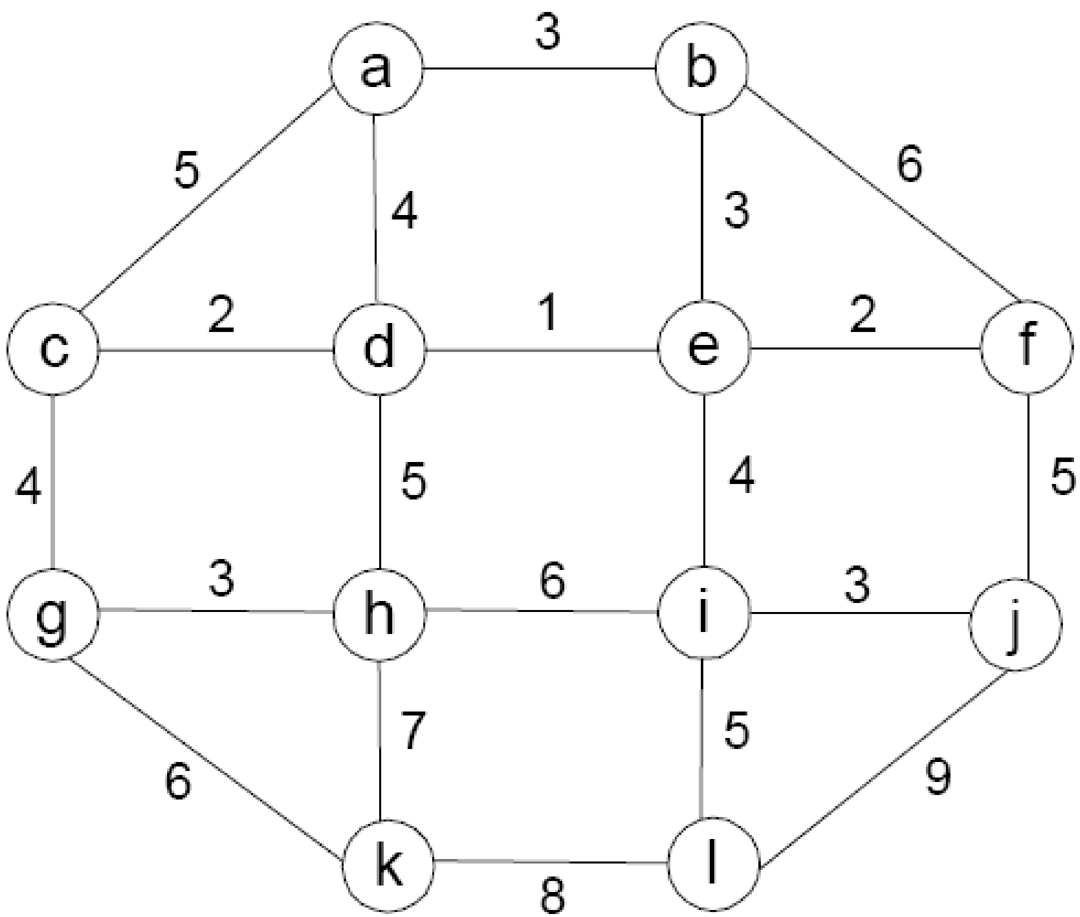




Graph Algorithms

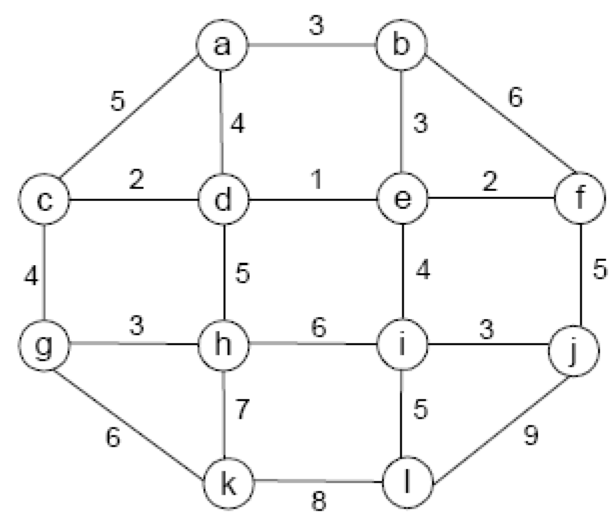
- **DFS, BFS**: visit all nodes
  - adj matrix: $\Theta(|V^2|)$, adj list: $\Theta(|V|+|E|)$
- **TSP** (Brute Force): shortest tour
  - $O(|V|!)$
- **Warshall** (DP): transitive closure
  Floyd (DP): all pairs shortest path
  - $\Theta(|V^3|)$
- **Prim** (greedy): minimum spanning tree
  Dijkstra (greedy): single source shortest path
  - matrix/array: $\Theta(|V^2|)$, list/heap: $O(|E|\log|V|)$
- **Kruskal** (greedy): minimum spanning tree
  - $O(|E|\log|E|)$

Class Discussion
Apply Prim's Algorithm



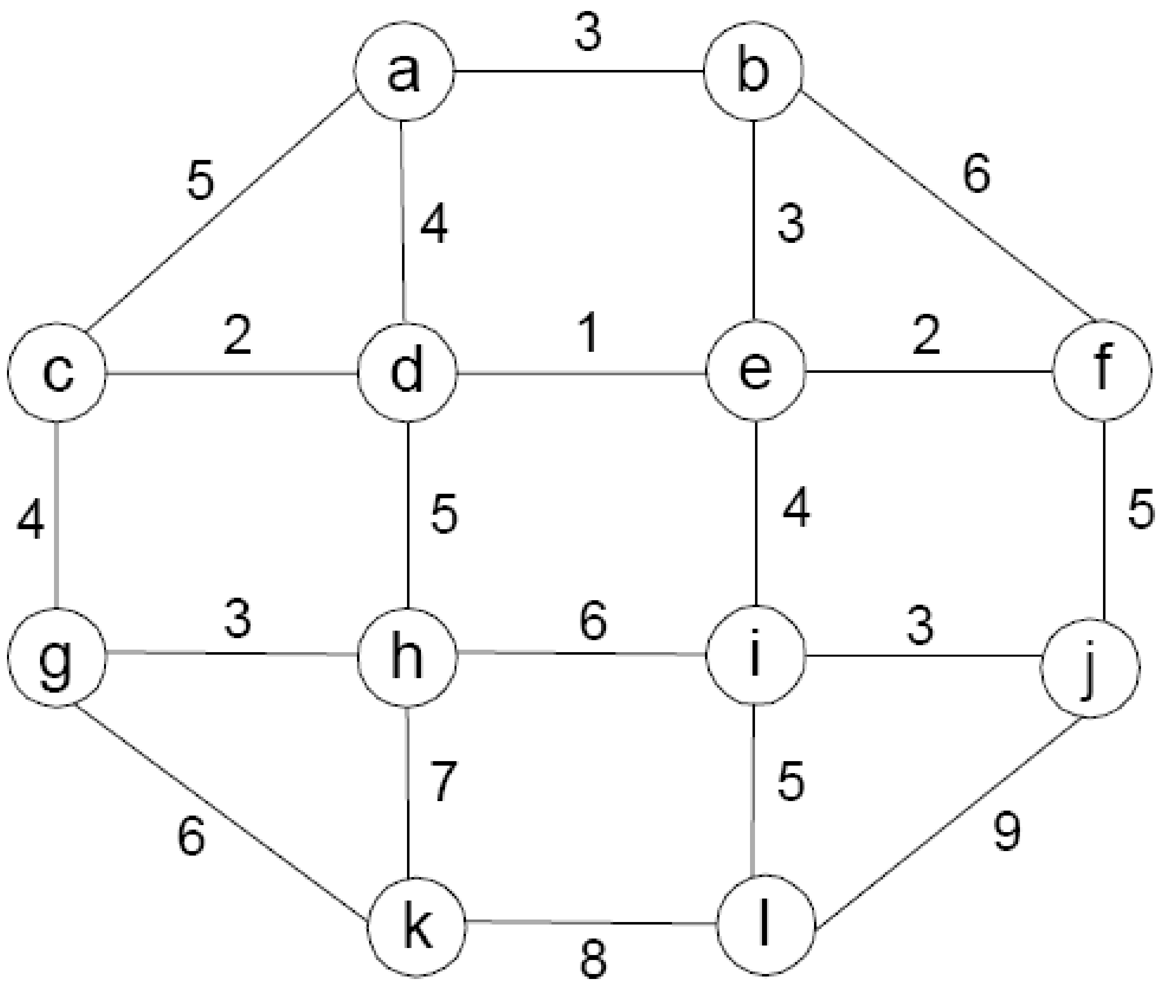
Class Discussion
Apply Prim's Algorithm

Tree vertices	Priority queue of fringe vertices
a(-,-)	**b(a,3)** c(a,5) d(a,4)
b(a,3)	c(a,5) d(a,4) **e(b,3)** f(b,6)
e(b,3)	*c(a,5)* **d(e,1)** f(e,2) i(e,4)
d(e,1)	**c(d,2)** f(e,2) i(e,4) h(d,5)
c(d,2)	**f(e,2)** i(e,4) h(d,5) g(c,4)
f(e,2)	**i(e,4)** h(d,5) g(c,4) j(f,5)
i(e,4)	h(d,5) g(c,4) **j(i,3)** l(i,5)
j(i,3)	h(d,5) **g(c,4)** l(i,5)
g(c,4)	**h(g,3)** l(i,5) k(g,6)
h(g,3)	**l(i,5)** k(g,6)
l(i,5)	**k(g,6)**
k(g,6)	

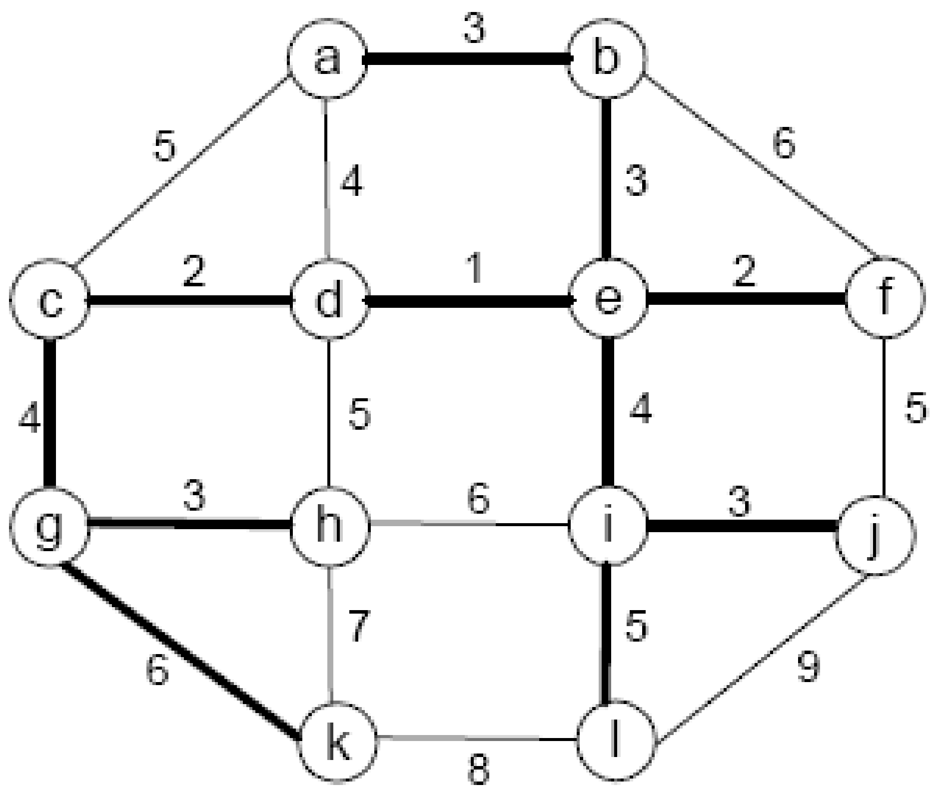


The minimum spanning tree found by the algorithm comprises the edges
ab, be, ed, dc, ef, ei, ij, cg, gh, il, gk.

Class Discussion
Apply Kruskal's Algorithm



Class Discussion
Apply Kruskal's Algorithm



Class Discussion

2. Indicate whether the following statements are true or false:

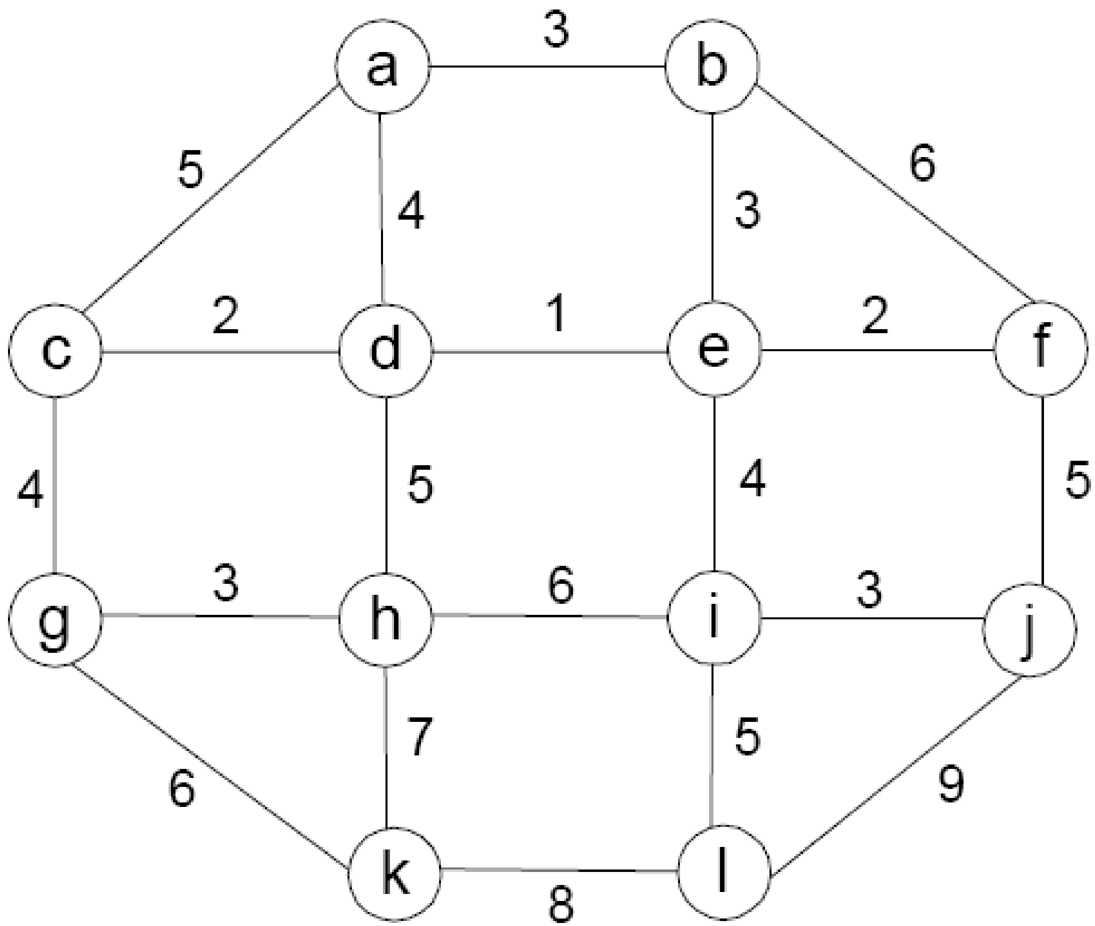
   a. If e is a minimum-weight edge in a connected weighted graph, it must be among edges of at least one minimum spanning tree of the graph.

   b. If e is a minimum-weight edge in a connected weighted graph, it must be among edges of each minimum spanning tree of the graph.

   c. If edge weights of a connected weighted graph are all distinct, the graph must have exactly one minimum spanning tree.

   d. If edge weights of a connected weighted graph are not all distinct, the graph must have more than one minimum spanning tree.

Class Discussion
Find shortest paths from a



Class Discussion
Find shortest paths from a

Tree vertices	Fringe vertices	Shortest paths from a
a(-,0)	**b(a,3)** c(a,5) d(a,4)	to b: $a - b$ of length 3
b(a,3)	c(a,5) **d(a,4)** e(b,3+3) f(b,3+6)	to d: $a - d$ of length 4
d(a,4)	**c(a,5)** e(d,4+1) f(a,9) h(d,4+5)	to c: $a - c$ of length 5
c(a,5)	**e(d,5)** f(a,9) h(d,9) g(c,5+4))	to e: $a - d - e$ of length 5
e(d,5)	**f(e,5+2)** h(d,9) g(c,9) i(e,5+4)	to f: $a - d - e - f$ of length 7
f(e,7)	**h(d,9)** g(c,9) i(e,9) j(f,7+5)	to h: $a - d - h$ of length 9
h(d,9)	**g(c,9)** i(e,9) j(f,12) k(h,9+7))	to g: $a - c - g$ of length 9
g(c,9)	**i(e,9)** j(f,12) k(g,9+6)	to i: $a - d - e - i$ of length 9
i(e,9)	**j(f,12)** k(g,15) l(i,9+5)	to j: $a - d - e - f - j$ of length 12
j(f,12)	k(g,15) **l(i,14)**	to l: $a - d - e - i - l$ of length 14
l(i,14)	**k(g,15)**	to k: $a - c - g - k$ of length 15
k(g,15)		