Graphs

Lesson 7

Outline

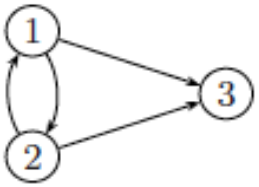
- Graphs
  - Basic Concepts
  - Sequential Representation
  - Linked Representation
- Graph Traversal
- Applications
  - Minimum(-cost) Spanning Tree
  - Shortest Path Problems

Outline

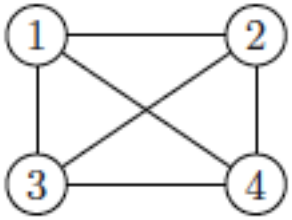
- **Graphs**
  - **Basic Concepts**
  - **Sequential Representation**
  - **Linked Representation**
- Graph Traversal
- Applications
  - Minimum(-cost) Spanning Tree
  - Shortest Path Problems

Graphs

- A ***graph*** consists of a finite nonempty set of ***vertices***, and a finite possibly empty set of ***edges***
  - ***Usually denoted by G = (V,E)***
- ***Directed graph (or digraph)***

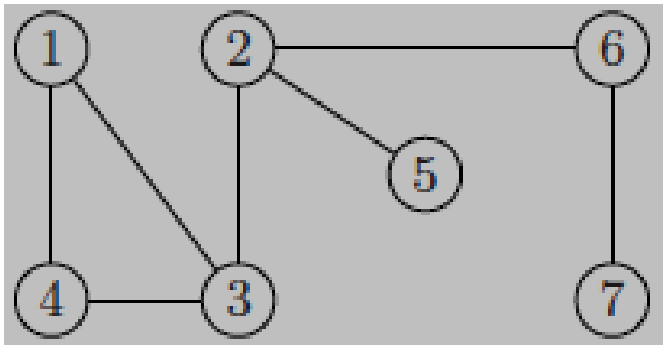


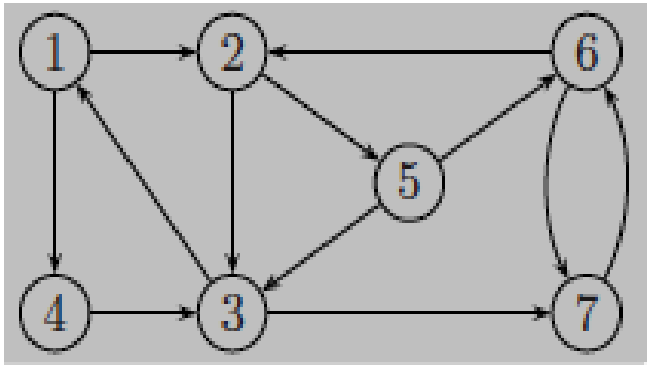
- ***Undirected graph***



Basic Concepts

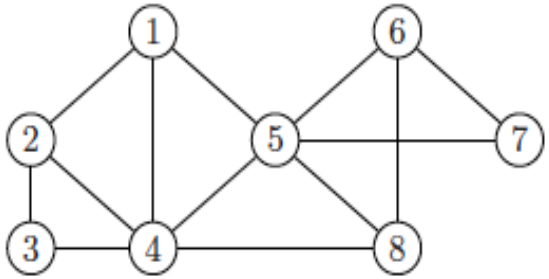
- ***Subgraph***
  - ***Induced subgraph***
- ***Degree of a vertex***
  - ***Out-degree***
  - ***In-degree***
- ***Path***
  - ***Simple path***
  - ***Cycle***
- ***Connected (components)***
  - ***Strongly connected***



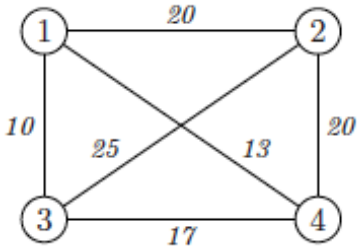


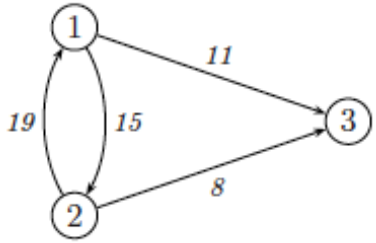
Basic Concepts

- ***Articulation points***
  - ***Biconnected***

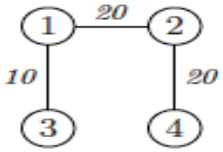


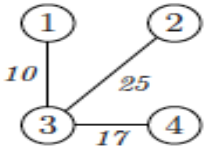
- ***Weighted Graph (Network)***

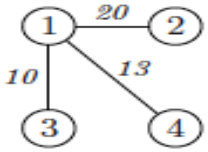


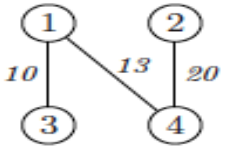


- ***Spanning Tree***



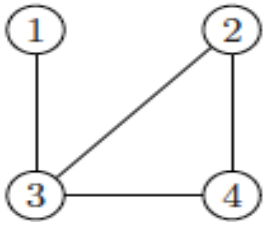




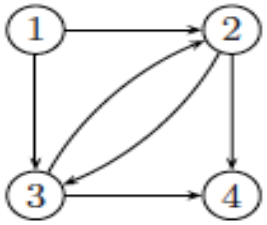


Sequential Representation

- ***Adjacency matrix***

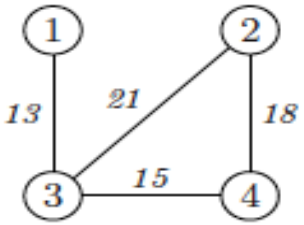


$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

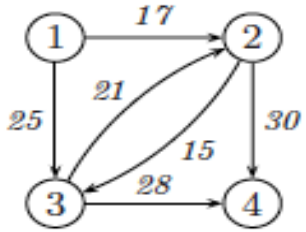
- ***Cost Adjacency matrix***



G_1

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & \infty & 13 & \infty \\ \infty & 0 & 21 & 18 \\ 13 & 21 & 0 & 15 \\ \infty & 18 & 15 & 0 \end{bmatrix} \end{array}$$

C_1



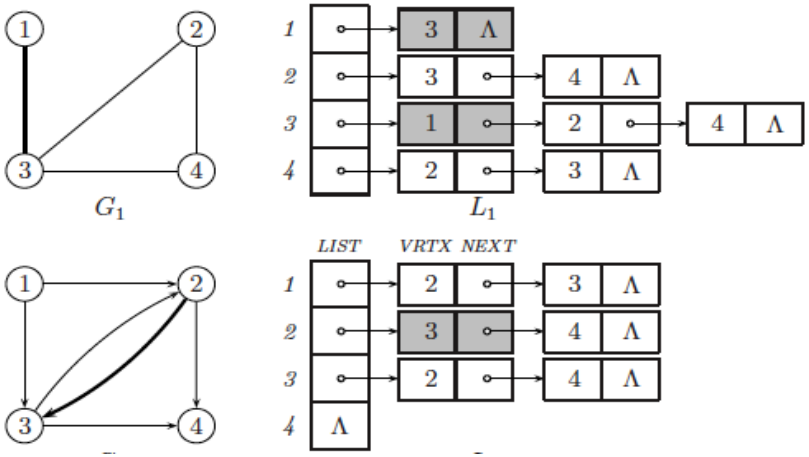
G_2

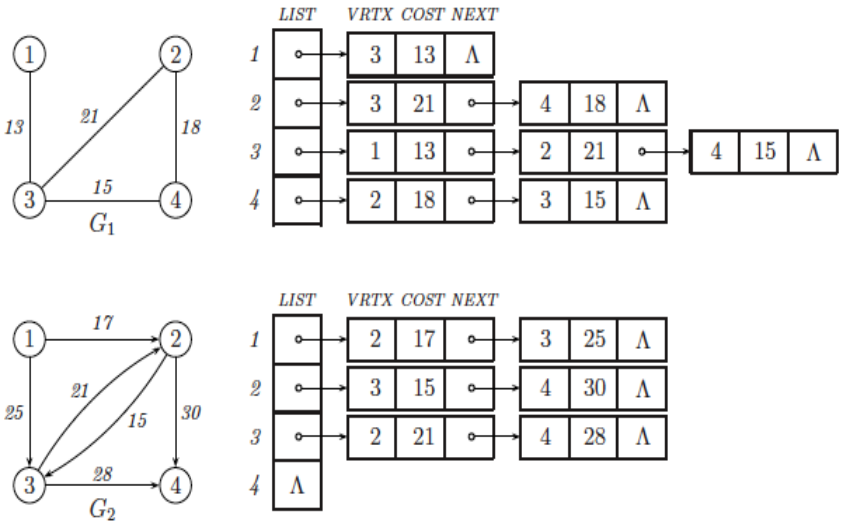
$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 17 & 25 & \infty \\ \infty & 0 & 15 & 30 \\ \infty & 21 & 0 & 28 \\ \infty & \infty & \infty & 0 \end{bmatrix} \end{array}$$

C_2

Linked Representation

- ***Adjacency list***



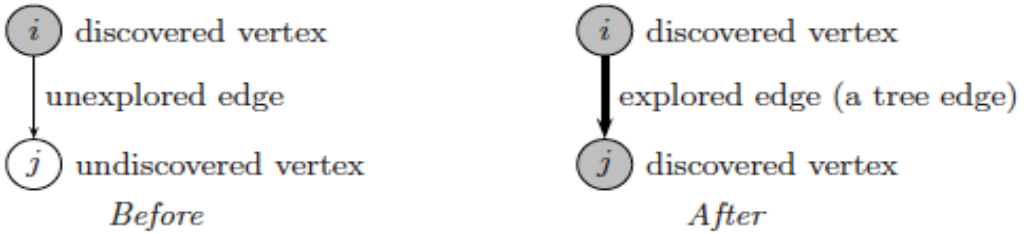


Outline

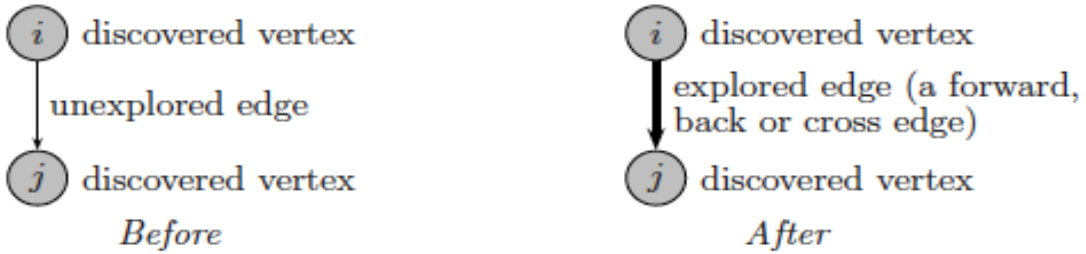
- Graphs
  - Basic Concepts
  - Sequential Representation
  - Linked Representation
- **Graph Traversal**
- Applications
  - Minimum(-cost) Spanning Tree
  - Shortest Path Problems

Basic Concepts

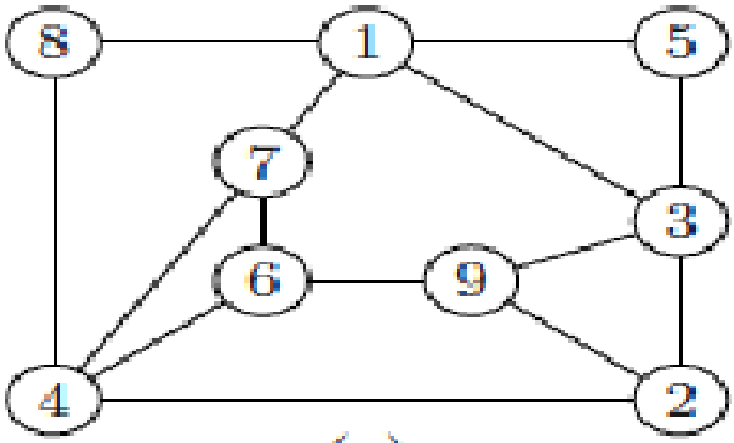
- ***Discovery edge (or tree edge)***



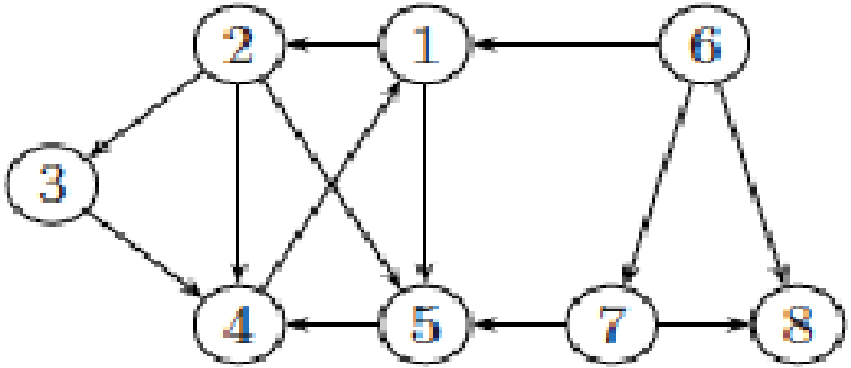
- ***Non-discovery edge***



Depth-First Search (DFS)

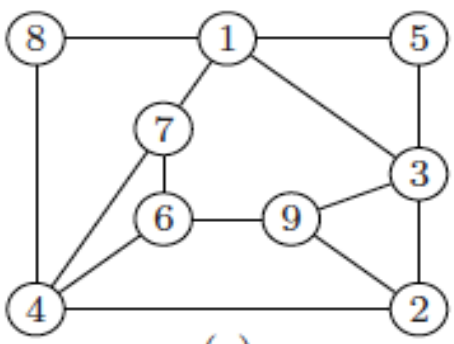


(a)

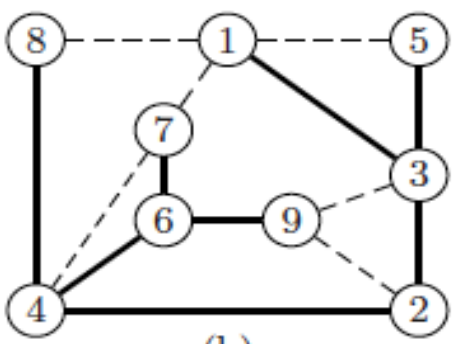


(a)

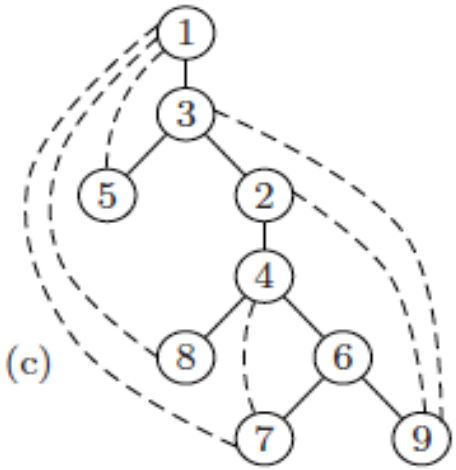
Depth-First Search (DFS)



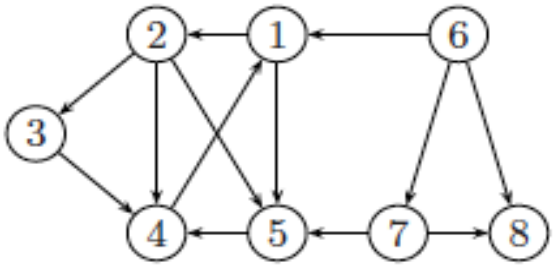
(a)



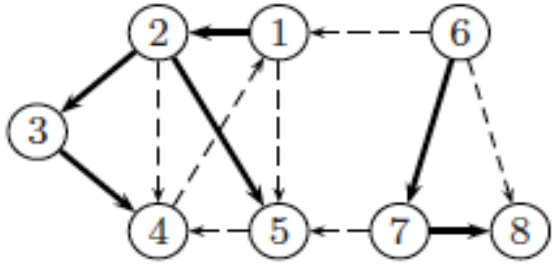
(b)



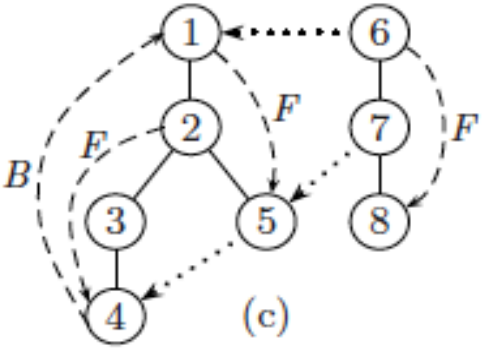
(c)



(a)



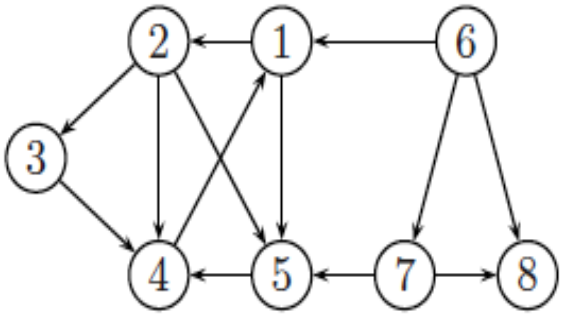
(b)

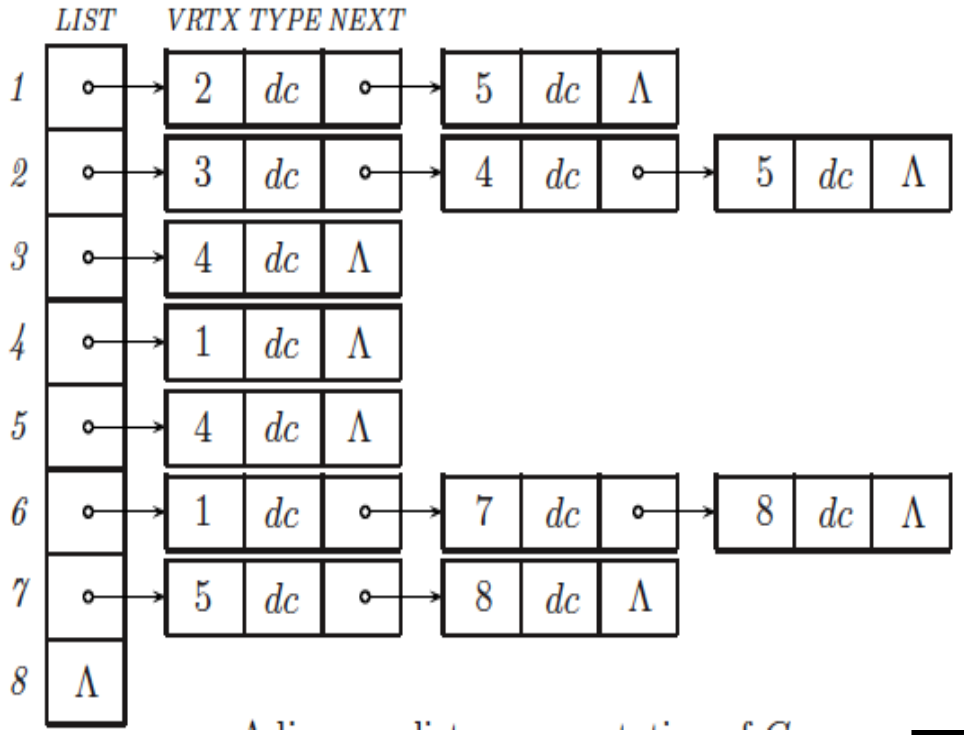


(c)

“Time-stamped” DFS

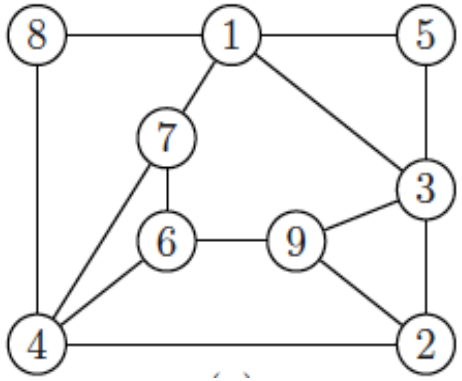
- G = {LIST(1:n) -> (VRTX, TYPE, NEXT), pred(1:n), f(1:n), d(1:n), color(1:n)}

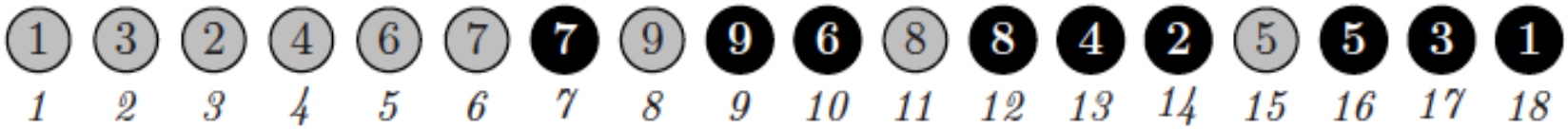




“Time-stamped” DFS

State	Status	Color
Undiscovered	All edges entering the vertex are unexplored.	white
Discovered		
Unfinished	Some edges leaving the vertex are still unexplored.	gray
Finished	All edges leaving the vertex are already explored.	black



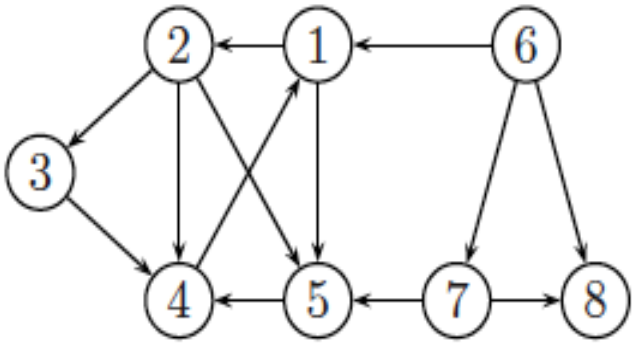


"Time-stamped" DFS

```
procedure DFS(𝔾, i)
color(i) ← gray
d(i) ← time ← time + 1
α ← LIST(i)
while α ≠ Λ do
   j ← VRTX(α)
   case
      : color(j) = white : [ TYPE(α) ← T; pred(j) ← i; call DFS(𝔾, j) ]
      : color(j) = gray  : if pred(i) ≠ j then TYPE(α) ← B
      : color(j) = black : if d(i) < d(j) then TYPE(α) ← F else TYPE(α) ← X
   endcase
   α ← NEXT(α)
endwhile
color(i) ← black
f(i) ← time ← time + 1
end DFS
```

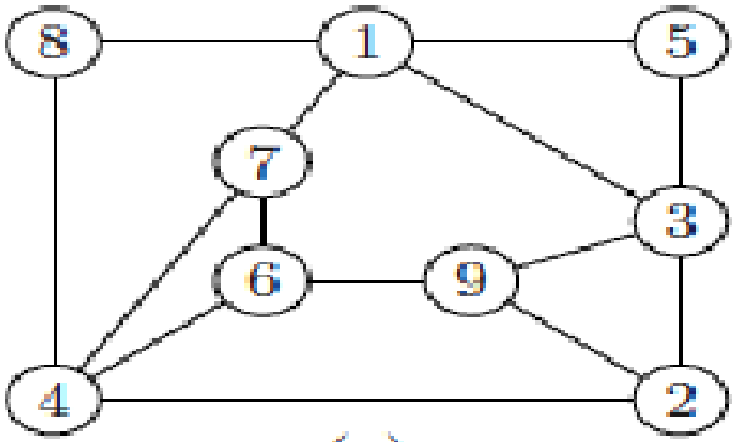
"Time-stamped" DFS

```
procedure DFS_DRIVER(𝔾)
time ← 0          ▷ global variable
color ← white     ▷ global array
pred ← 0
for i ← 1 to n do
   if color(i) = white then call DFS(𝔾, i)
endfor
end DFS_DRIVER
```

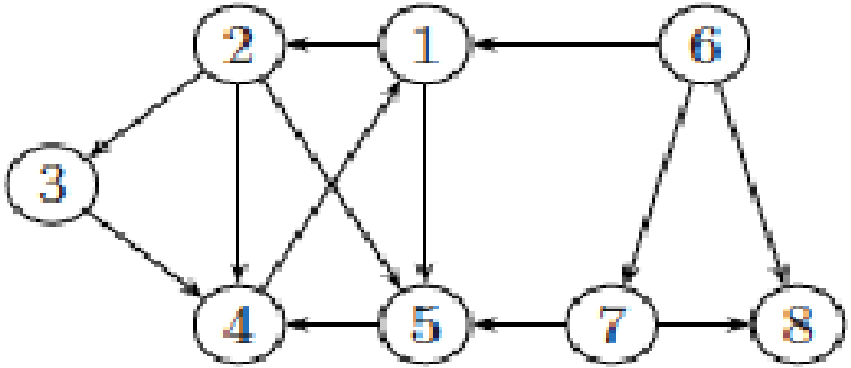


	1	2	3	4	5	6	7	8
$pred(1:8)$	0	1	2	3	2	0	6	7
$d(1:8)$	1	2	3	4	7	11	12	13
$f(1:8)$	10	9	6	5	8	16	15	14

Breadth-First Search (BFS)

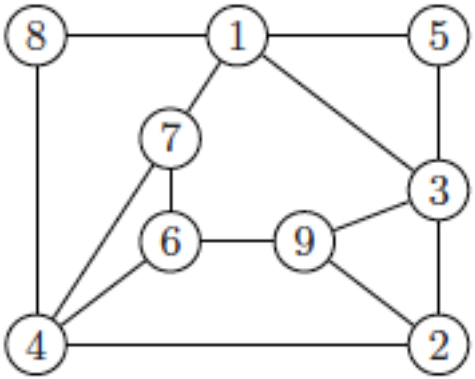


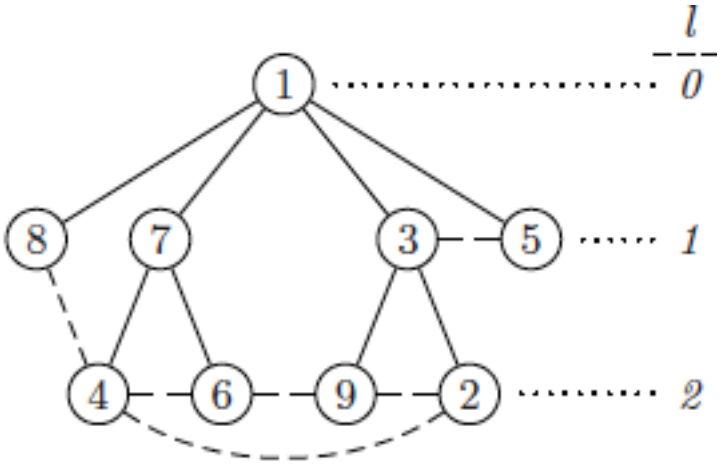
(a)

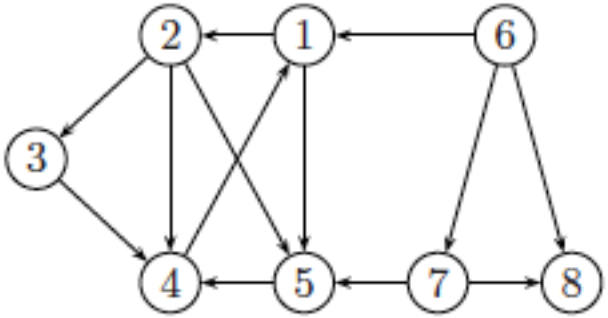


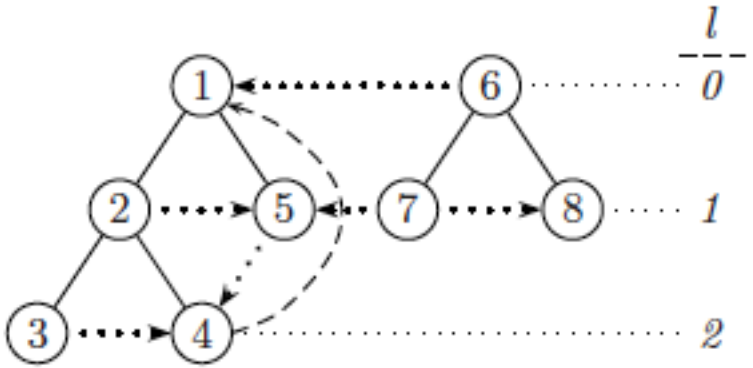
(a)

Breadth-First Search (BFS)



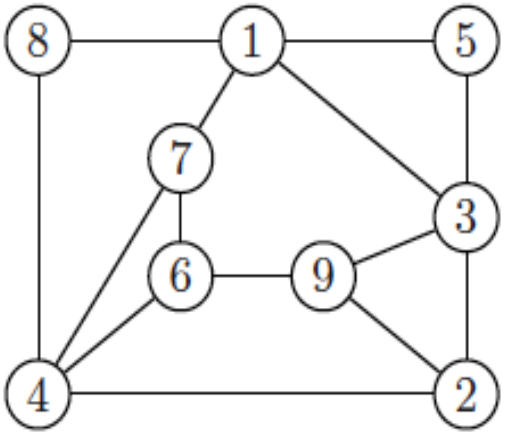


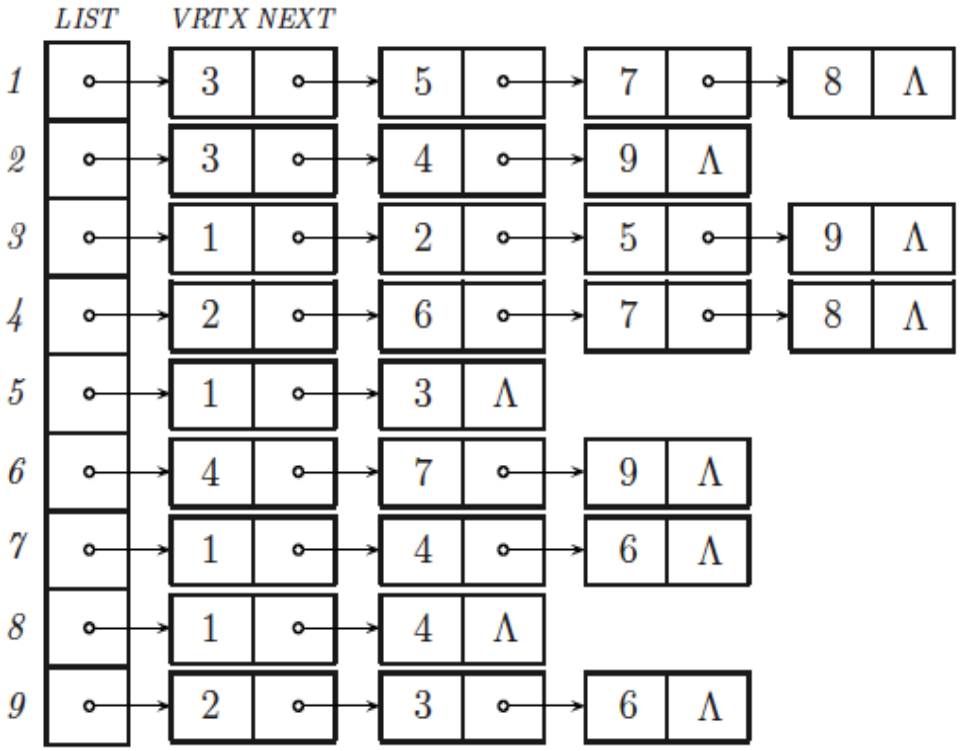




“Time-stamped” BFS

- G = {LIST(1:n) -> (VRTX, NEXT), pred(1:n), l(1:n)}





“Time-stamped” BFS

- ***Fringe is a queue containing vertices that have yet to be expanded***

State	Status	Color
Undiscovered	All edges entering the vertex are unexplored.	white
Discovered		
Fringe	Some edges leaving the vertex are still unexplored.	gray
Finished	All edges leaving the vertex are already explored.	black

"Time-stamped" BFS

```
procedure BFS(𝔾, s)
call InitQueue(ℚ)
color(s) ← gray
call ENQUEUE(ℚ, s)
l(s) ← 0
while not IsEmptyQueue(ℚ) do
   call DEQUEUE(ℚ, i)
   α ← LIST(i)
   while α ≠ Λ do
      j ← VRTX(α)
      if color(j) = white then [ color(j) ← gray; pred(j) ← i
                                 l(j) ← l(i) + 1; call ENQUEUE(ℚ, j) ]
      α ← NEXT(α)
   endwhile
   color(i) ← black
endwhile
end BFS
```

```
procedure BFS_DRIVER(𝔾)
color ← white        ▷ global array
pred ← 0
l ← ∞
for s ← 1 to n do
   if color(s) = white then call BFS(𝔾, s)
endfor
end BFS_DRIVER
```

Notes on BFS and DFS

- Time complexity (BOTH DFS and BFS)
  - Linked = $O(|V| + |E|)$
  - Seq = $O(|V|^2)$
- For finite trees, DFS and BFS have comparable performance.
- For extremely large trees / “infinite” tree, BFS will outperform DFS
  - Or alternatively, $\#steps_{BFS} \leq \#steps_{DFS}$

Outline

- Graphs
  - Basic Concepts
  - Sequential Representation
  - Linked Representation
- Graph Traversal
- **Applications**
  - **Minimum(-cost) Spanning Tree**
  - **Shortest Path Problems**

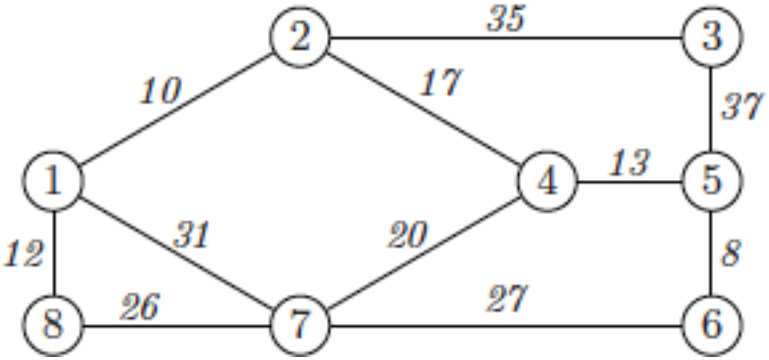
Minimum(-cost) Spanning Trees

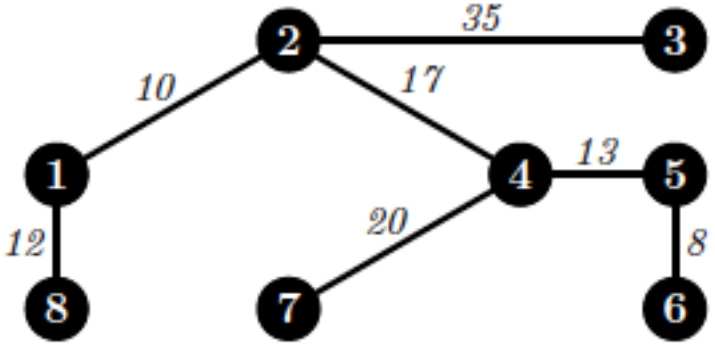
MST theorem: Let $G = (V, E)$ be a connected, weighted, undirected graph. Let U be some proper subset of V and (i, j) be an edge of least cost such that $i \in U$ and $j \in V - U$. There exists a minimum cost spanning tree T such that (i, j) is an edge in T.

- MST Algorithms
  - ***Prim's Algorithm***
  - ***Kruskal's Algorithm***

Prim's Algorithm

1. [Start vertex] Choose any vertex in V and place it in U.
2. [Next vertex] From among the vertices in $V - U$ choose that vertex, say j, which is connected to some vertex, say i, in U by an edge of least cost. Add vertex j to U and edge (i, j) to T.
3. [All vertices considered?] Repeat Step 2 until $U = V$. Then, T is a minimum-cost spanning tree for G.

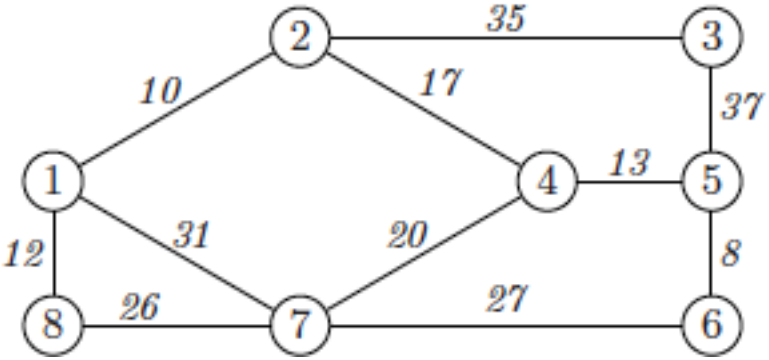


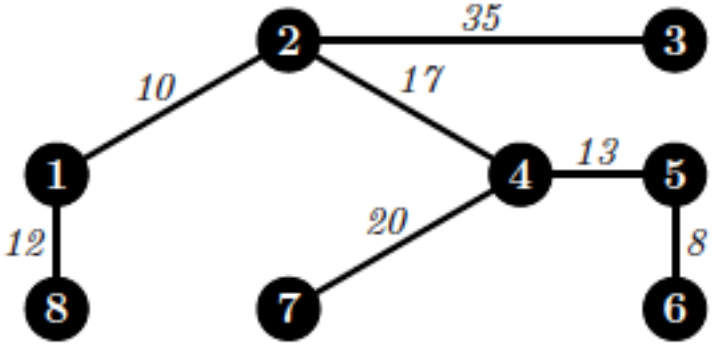


Cost = 115

Kruskal's Algorithm

1. [Initial edge.] Choose the edge of least cost among all the edges in E and place it in T.
2. [Next edge.] From among the remaining edges in E choose the edge of least cost, say edge (i, j). If including edge (i, j) in T creates a cycle with the edges already in T, discard (i, j); otherwise, include (i, j) in T.
3. [Enough edges in T?] Repeat Step 2 until there are $n - 1$ edges in T. Then T is a minimum-cost spanning tree for G.





Cost = 115

Analysis of MST Algorithms

- Based on EASY code in the book
- Prim = O(|E| log |V|)
  - Initialization = O(|V|)
  - Extract operation = O(|V| log |V|)
  - Change operation = O (|E| log |V|)
- Kruskal = O(|E| log |E|)
  - Initialization = O(|V|)
  - Extract operation = O (|E| log |E|)
  - Heap processing = O(|E| G(|E|), G(|E|) ≤ 5

Shortest path problem

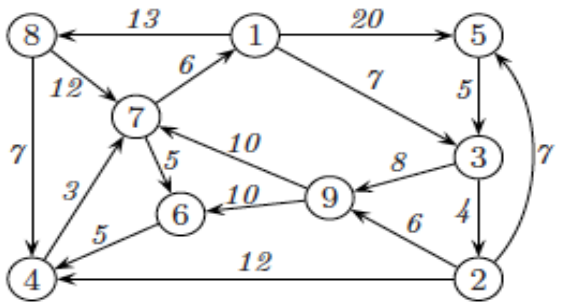
- Given a start node i and an end node j (usually, i ≠ j), find a path with least cost from i to j
  - Algorithms extend the concept s.t. all shortest paths from a start node to any node is searched
- Two versions
  - ***Single source shortest path (SSSP)***
    - Dijkstra's algorithm
  - ***All pairs shortest path (APSP)***
    - Floyd's Algorithm

Dijkstra's Algorithm

1. Place vertex s in class 1 and all other vertices in class 2.
2. Set the value of vertex s to zero and the value of all other vertices to ∞.
3. Do the following until all vertices v in V that are reachable from s are placed in class 1:
   a. Denote by u the vertex most recently placed in class 1.
   b. Adjust all vertices v in class 2 as follows:
      (i) If vertex v is not adjacent to u, retain the current value of $d(v)$.
      (ii) If vertex v is adjacent to u, adjust $d(v)$ as follows:

$$\textbf{if } d(v) > \delta(u)+w((u,v)) \textbf{ then } d(v) \leftarrow \delta(u)+w((u,v)) \qquad (10.1)$$

   c. Choose a class 2 vertex with *minimal* value and place it in class 1.

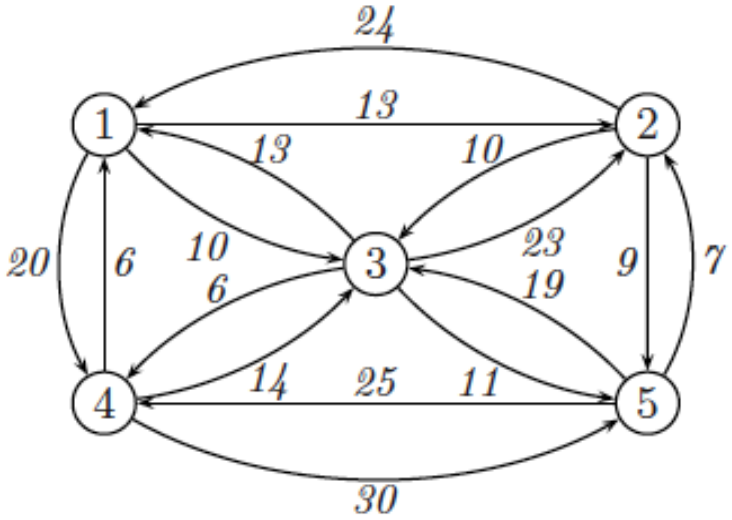


Floyd's Algorithm

1. [Initialize] $D^{(0)} \leftarrow C$
2. [Iterate] Repeat for $k = 1, 2, 3, \ldots, n$

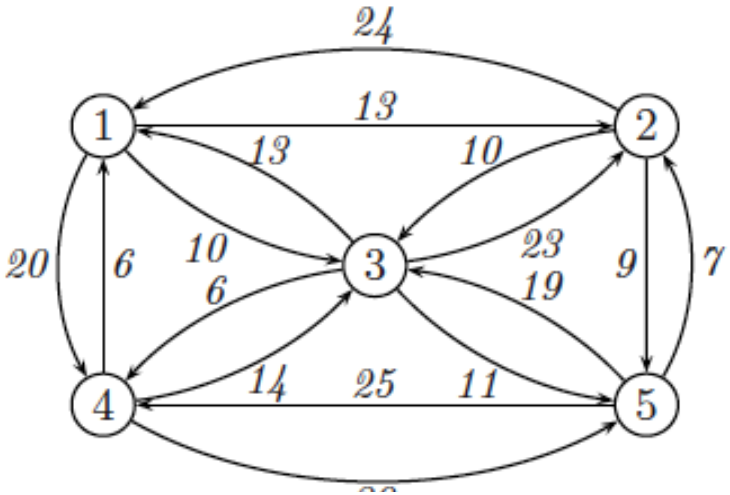
$$d_{ij}^{(k)} \leftarrow \text{minimum}[\, d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \,], \quad 1 \le i, j \le n$$

Then, $D^{(n)}$ contains the cost of the shortest path between every pair of vertices i and j in G.



$$\begin{bmatrix} 0 & 13 & 10 & 20 & \infty \\ 24 & 0 & 10 & \infty & 9 \\ 13 & 23 & 0 & 6 & 11 \\ 6 & \infty & 14 & 0 & 30 \\ \infty & 7 & 19 & 25 & 0 \end{bmatrix}$$

Floyd's Algorithm



$$\begin{bmatrix} 0 & 13 & 10 & 20 & \infty \\ 24 & 0 & 10 & \infty & 9 \\ 13 & 23 & 0 & 6 & 11 \\ 6 & \infty & 14 & 0 & 30 \\ \infty & 7 & 19 & 25 & 0 \end{bmatrix}$$

$$d_{23}^{(1)} = \min\left[d_{23}^{(0)}, d_{21}^{(0)} + d_{13}^{(0)}\right] = \min\left[10, 24+10\right] = 10$$

$$d_{24}^{(1)} = \min\left[d_{24}^{(0)}, d_{21}^{(0)} + d_{14}^{(0)}\right] = \min\left[\infty, 24+20\right] = 44$$

$$d_{42}^{(1)} = \min\left[d_{42}^{(0)}, d_{41}^{(0)} + d_{12}^{(0)}\right] = \min\left[\infty, \ 6+13\right] = 19$$

$$D^{(1)} = \begin{bmatrix} 0 & 13 & 10 & 20 & \infty \\ 24 & 0 & 10 & \mathit{44} & 9 \\ 13 & 23 & 0 & 6 & 11 \\ 6 & \mathit{19} & 14 & 0 & 30 \\ \infty & 7 & 19 & 25 & 0 \end{bmatrix}$$

Floyd's Algorithm

$$\begin{bmatrix} 0 & 13 & 10 & 20 & \infty \\ 24 & 0 & 10 & \infty & 9 \\ 13 & 23 & 0 & 6 & 11 \\ 6 & \infty & 14 & 0 & 30 \\ \infty & 7 & 19 & 25 & 0 \end{bmatrix} \quad D^{(0)}$$

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 2 & 0 & 2 & 0 & 2 \\ 3 & 3 & 0 & 3 & 3 \\ 4 & 0 & 4 & 0 & 4 \\ 0 & 5 & 5 & 5 & 0 \end{bmatrix} \quad pred^{(0)}$$

$$\begin{bmatrix} 0 & 13 & 10 & 20 & \infty \\ 24 & 0 & 10 & \mathit{44} & 9 \\ 13 & 23 & 0 & 6 & 11 \\ 6 & \mathit{19} & 14 & 0 & 30 \\ \infty & 7 & 19 & 25 & 0 \end{bmatrix} \quad D^{(1)}$$

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 2 & 0 & 2 & \mathit{1} & 2 \\ 3 & 3 & 0 & 3 & 3 \\ 4 & \mathit{1} & 4 & 0 & 4 \\ 0 & 5 & 5 & 5 & 0 \end{bmatrix} \quad pred^{(1)}$$

Analysis of Shortest Path Algorithms

- Based on EASY code in the book
- Dijkstra = O(|E| log |V|)
  - Initialization = O(|V|)
  - Extract operation = O(|V| log |V|)
  - Change operation = O (|E| log |V|)
- Floyd = $O(|V|^3)$
  - Can be made to run at O(|V||E| log |V|) if Floyd is implemented as an iterated version of Dijkstra
  - Much simpler to code though

Thank you
for your attention.

Questions ...