Graph

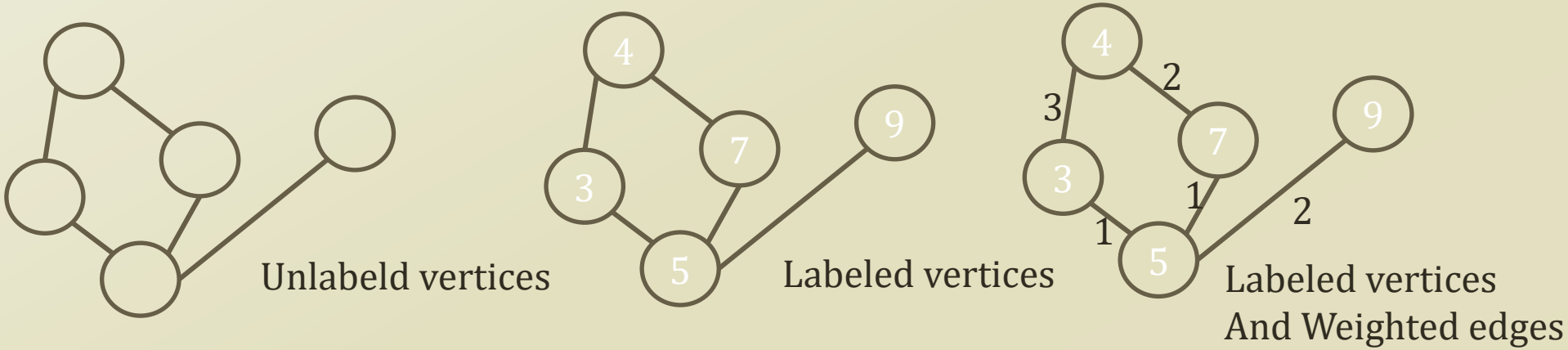
Il-Chul Moon
Dept. of Industrial and Systems Engineering
KAIST
icmoon@kaist.ac.kr

Weekly Objectives

- This week, we study graphs.
- Objectives are
  - Understanding the data structure of graphs
    - Able to implement the data structure for dense graphs
    - Able to implement the data structure for sparse graphs
  - Understanding the operations of graphs
    - BFS and DFS traverse
  - Understanding the algorithms on graphs
    - Dijkstra's shortest path algorithm
    - Minimum Spanning Tree

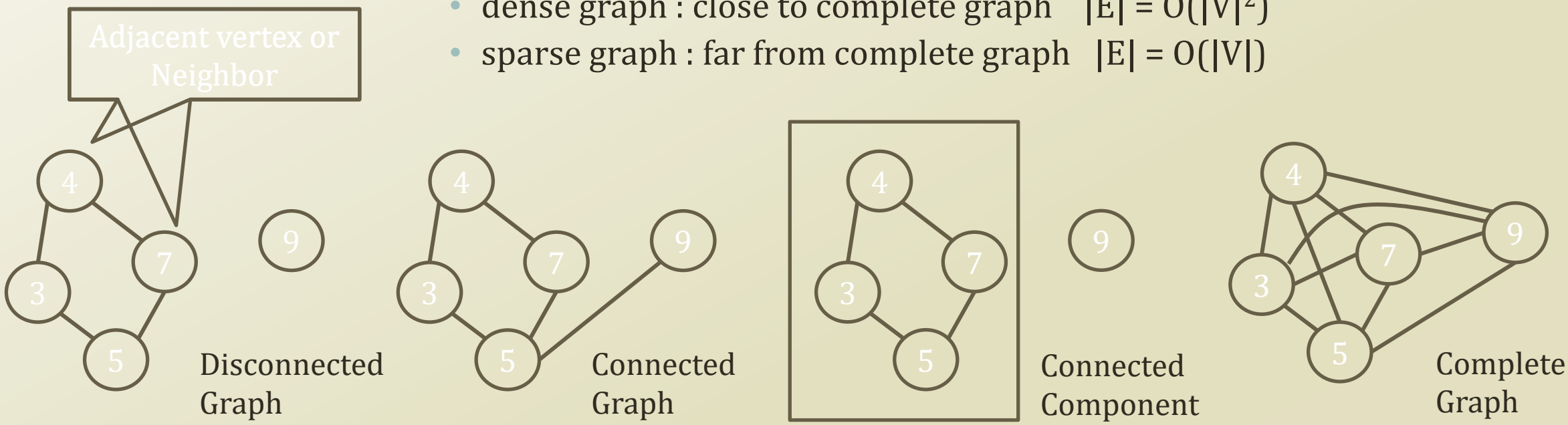
Graphs

- Examples of ordered collections, where each item may have successors and predecessors :
  - List : one predecessor, one successor at most
  - Tree : one predecessor (parent), several successors (children)
  - Graph : several predecessors and successors
- Graph G = (V, E)
  - V = { v_i } : a finite non-empty set of vertices (or nodes)
  - E = { e_i } : a finite (possibly empty) set of edges (or arcs)
    - e_i connects two vertices in V



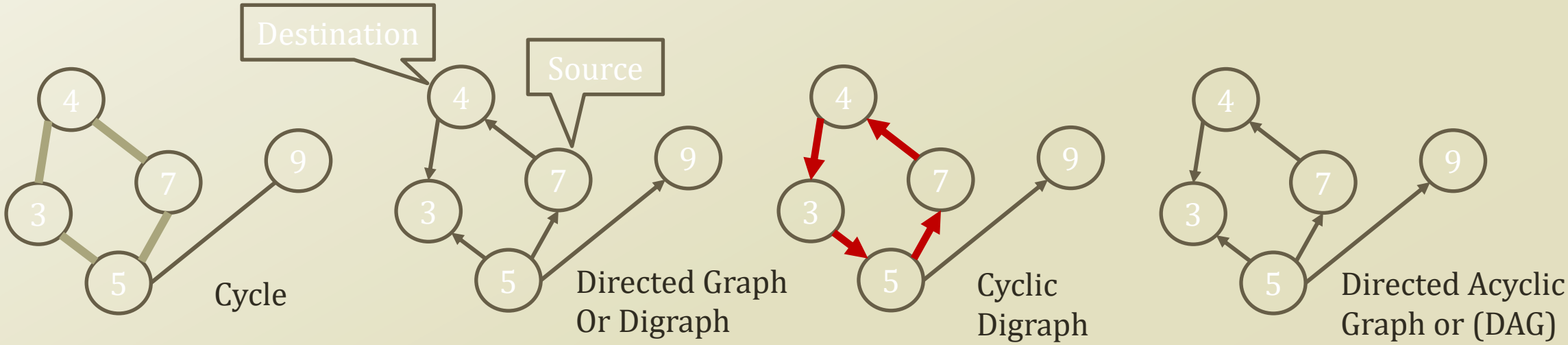
Graph terminology

- **adjacent**, **neighbor**, **incident**
- **path** between A and B : a sequence of edges connecting A and B
- **connected graph** : path from each to every other vertex
- **connected component** : graph subset containing the set of vertices reachable from a vertex and their edges
- **complete graph** : edge for every pair of vertices
  - dense graph : close to complete graph $|E| = O(|V|^2)$
  - sparse graph : far from complete graph $|E| = O(|V|)$



Graph terminology

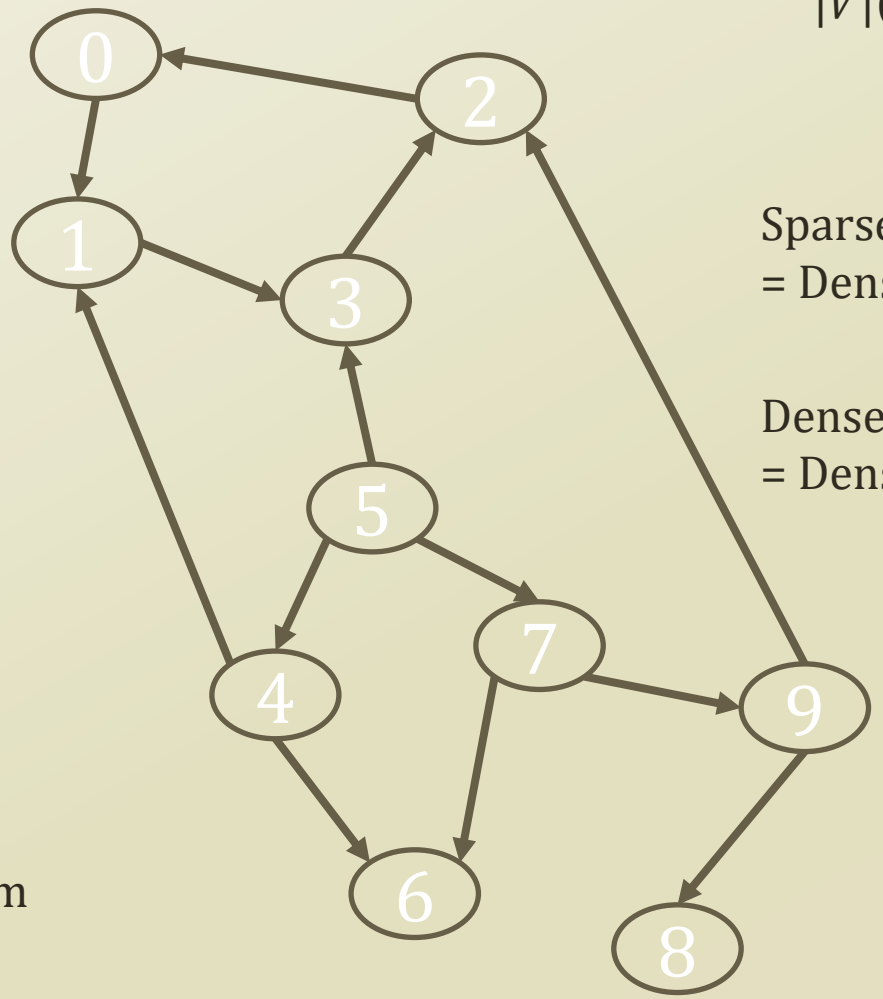
- **Cycle** : a path starting from a node and ending the node itself
- **Directed edge** : an edge with direction (source and destination)
- **Digraph** : Directed graph.
  - Graph with directed edges
- **DAG** : Directed Acyclic Graph
  - Directed graph without cycle



Data structure for graphs

- To store a graph
  - Store a set of vertexes
    - 0,1,2,3,4,5,6,7,8,9
  - Store a set of edges
    - (0,1), (1,3), (2,0), (5,3)…
- How to store?
  - Storing vertexes
    - Simple.
    - Linked list, BST, Hash….
  - Store edges
    - Fundamentally, a pair of values
    - Initially, a two-dimensional matrix
      - Space: O(V^2)
      - Time: O(1)
    - However….
    - Graph density becomes problem
    - So, adjacency list
      - Space: O(E)
      - Time: O(E)

$$Density = \frac{|E|}{|V|(|V|-1)}$$



Sparse graph
= Density→0

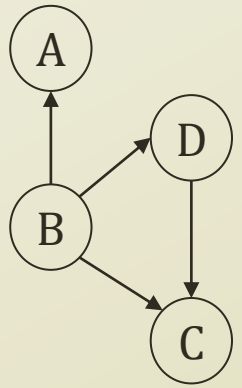
Dense graph
= Density→1

Matrix Representation for Dense Graph

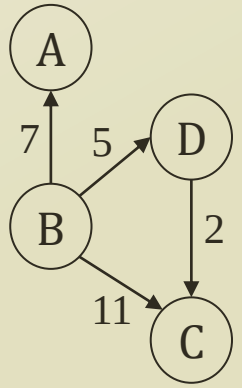
- Array Representation : Adjacency Matrix
- Linked Representation : Adjacency List

- Adjacency Matrix
  - $A[i][j] = 1$ if $(v_i, v_j) \in E$
    0 otherwise

- Adjacency Matrix for weighted graph
  - edge weight value instead of 0/1



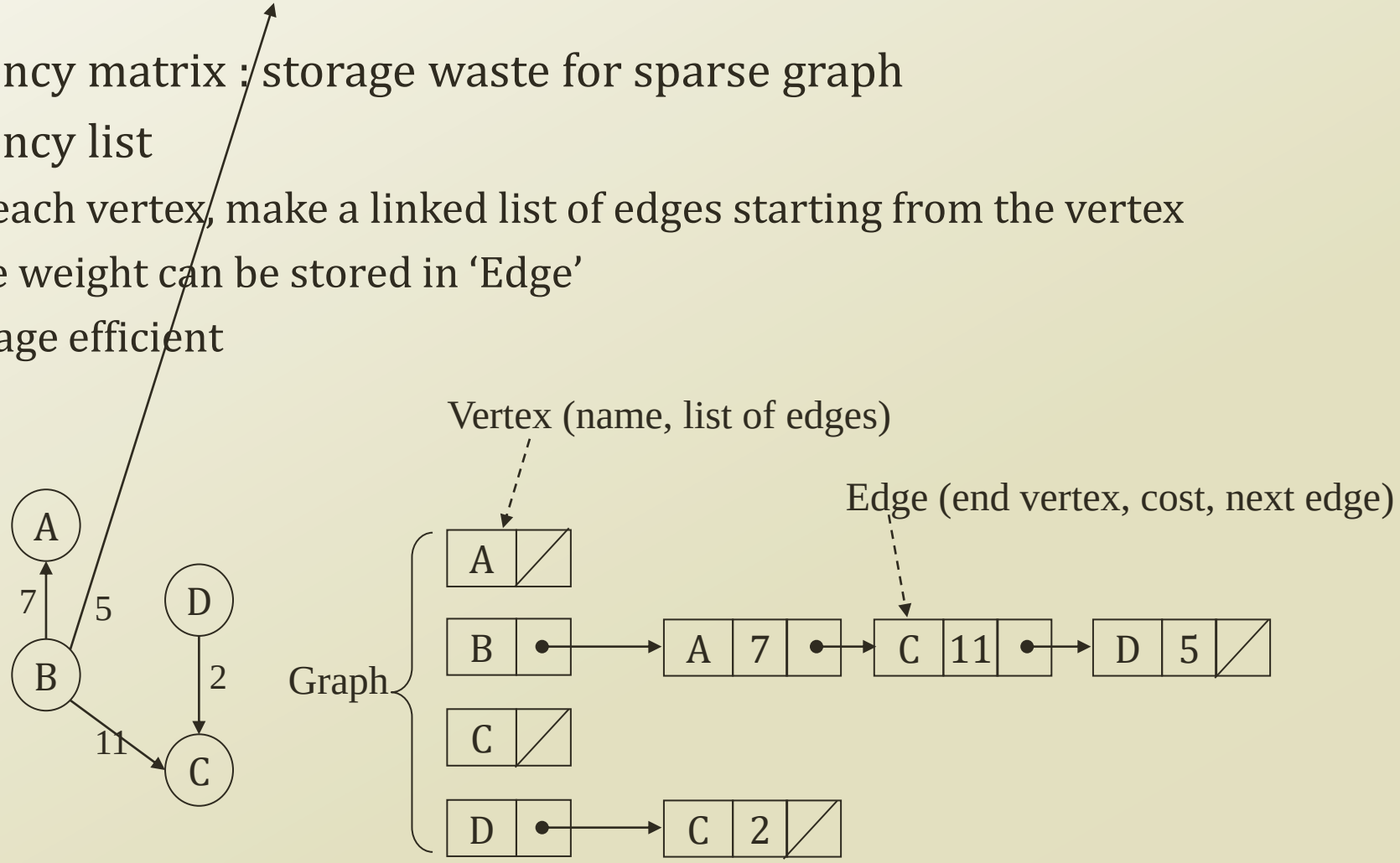
		0	1	2	3
		A	B	C	D
0	A	0	0	0	0
1	B	1	0	1	1
2	C	0	0	0	0
3	D	0	0	1	0



		0	1	2	3
		A	B	C	D
0	A	0	0	0	0
1	B	7	0	11	5
2	C	0	0	0	0
3	D	0	0	2	0

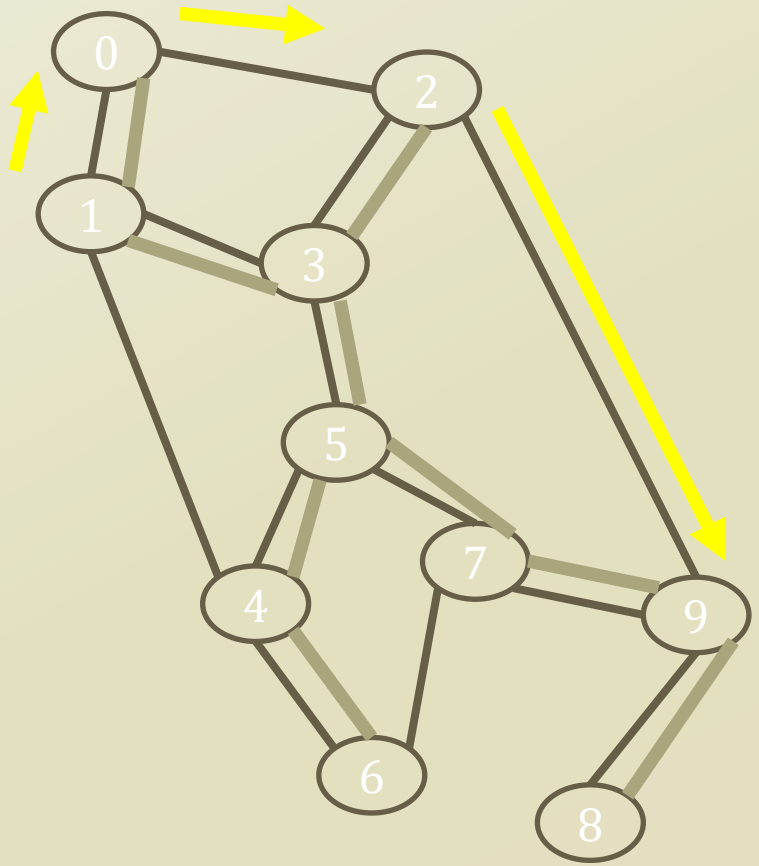
Adjacency List Representation for Sparse Graph

- Adjacency matrix : storage waste for sparse graph
- Adjacency list
  - For each vertex, make a linked list of edges starting from the vertex
  - Edge weight can be stored in 'Edge'
  - Storage efficient



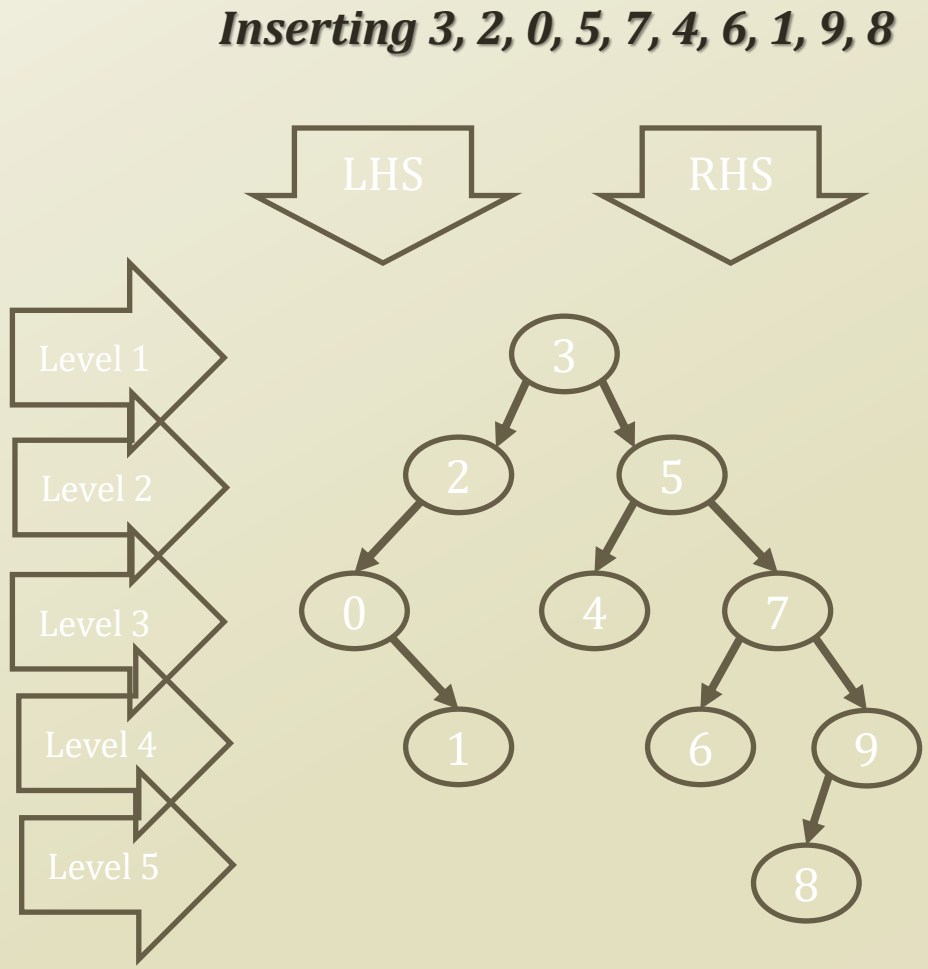
Operations of graph data structure

- Operations on graphs
  - Operation of retrieving vertexes
    - BFS traverse
    - DFS traverse
  - Operation of finding shortest paths
    - The shortest path
      - From vertex 1
      - To vertex 9
  - Operation of finding a set of path to control whole vertexes
    - The minimum spanning tree



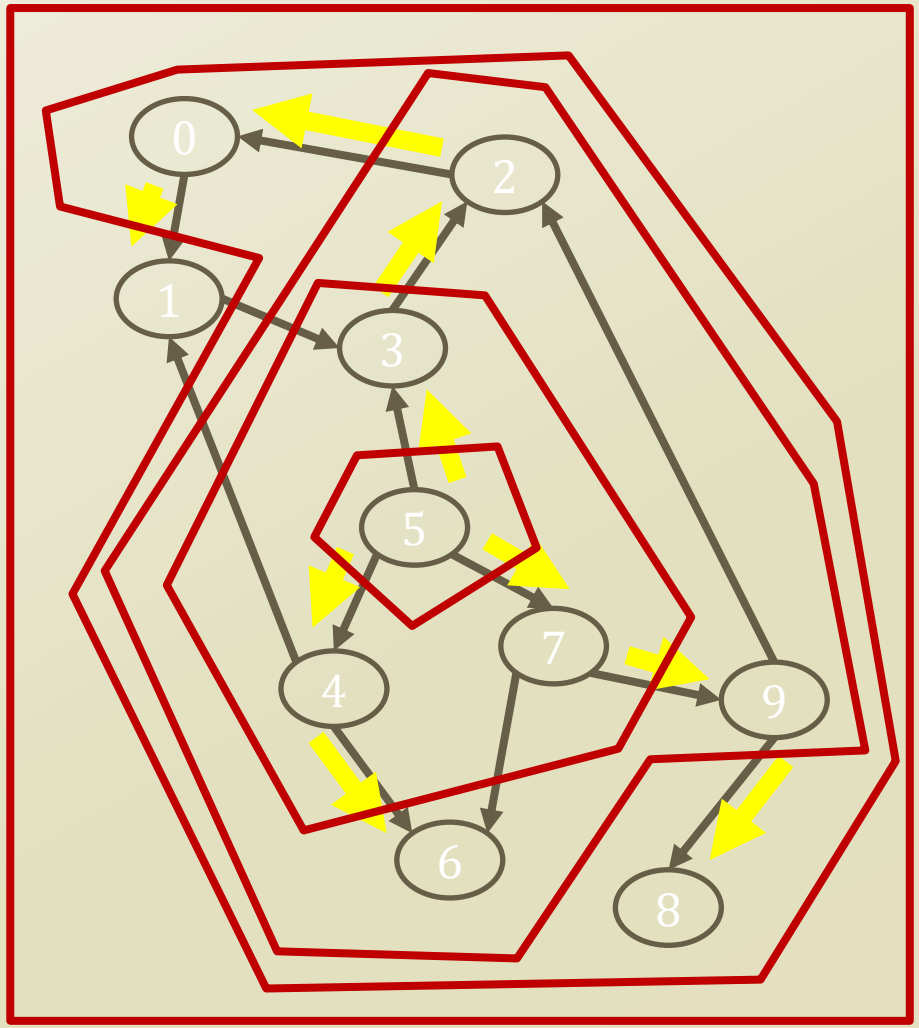
Detour: Tree traversing

- Tree
  - Complicated than a list
  - Multiple ways to show the entire dataset
    - If it were a list
      - Just show the values from the beginning to the end
    - Since this is a BST
      - You have to choose what to show at a time
        - The value in LHS
        - The value in RHS
        - The value that you have
- Hence there are multiple traversing approaches



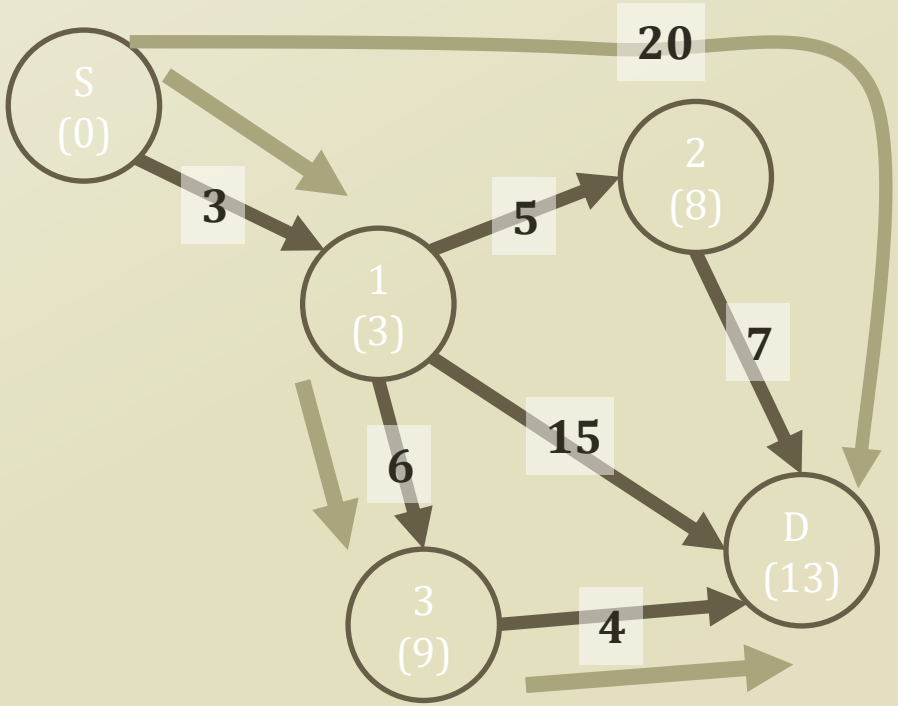
DFS vs. BFS traverse on graphs

- DFS utilizes
  - Stacks, or recursions that imitates the stack operations
  - Pre-order traverse
  - In-order traverse
  - Post-order traverse
  - In graphs, often only pre-order traverse is used
- BFS utilizes
  - Queues
  - Lever-order traverse
- Having said this,
  - Tree is a directed acyclic graph.
  - Graph may not be a DAG
  - Then….
    - You have to check the repeated visits to avoid falling into a cycle



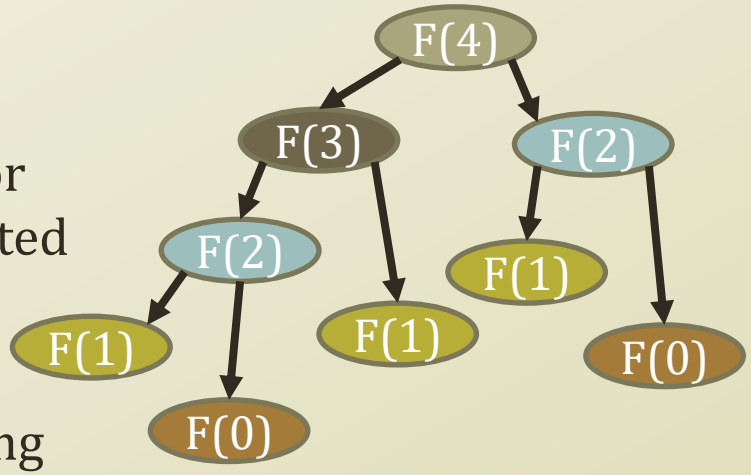
Single-Source Shortest Path Problem

- One recurring problem in graph
- Happens in
  - Path finding
  - Routing on comm. networks
  - Social networks
- We know where we are
- We want to know how long to travel to our destination
- Terminology
  - Source = where we start
  - Destination = where we arrive



Detour: Dynamic Programming

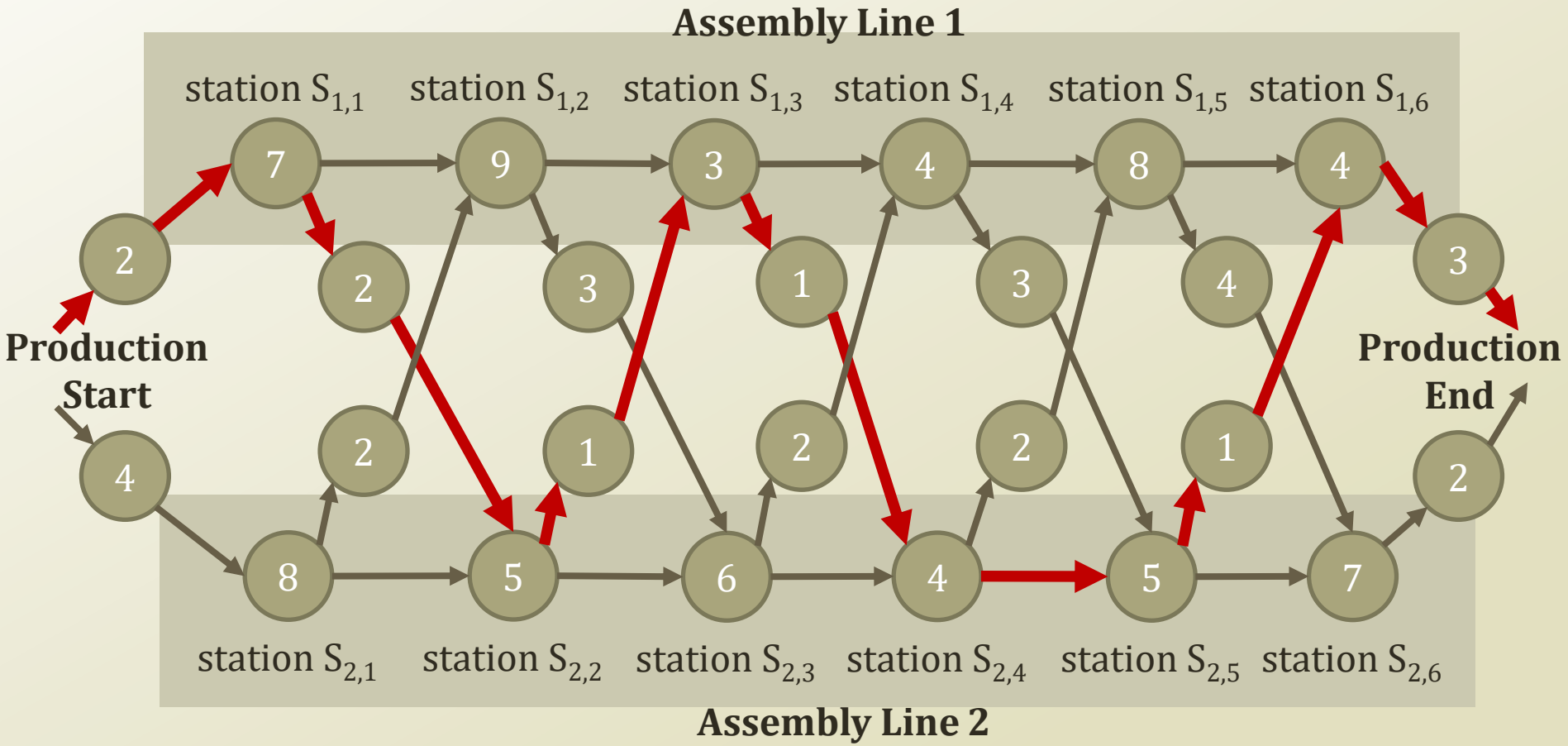
- Dynamic programming:
  - A general algorithm design technique for solving problems defined by or formulated as ***recurrences with overlapping sub-instances***
  - In this context, Programming == Planning
- Main storyline
  - Setting up a recurrence
    - Relating a solution of a larger instance to solutions of some smaller instances
    - Solve small instances once
    - Record solutions in a table
    - Extract a solution of a larger instance from the table



Memoization Table

Instance	Solution
F(0)	0
F(1)	1
F(2)	1
F(3)	2
F(4)	?

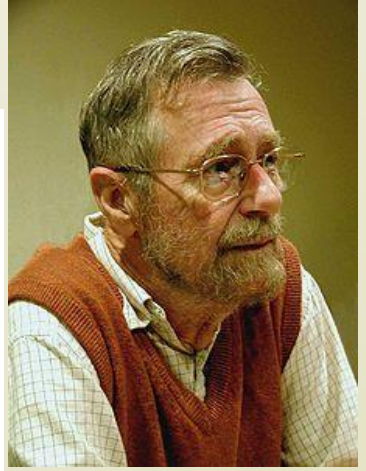
Detour: Tracing Assembly Line Scheduling in DP



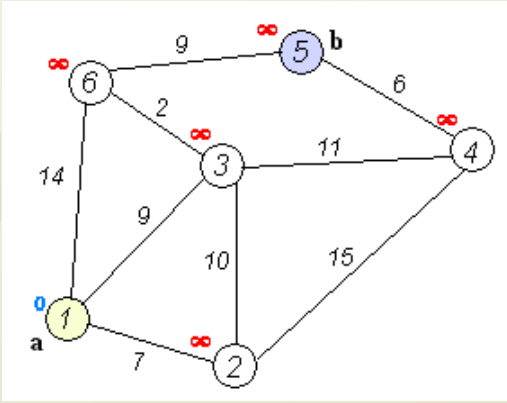
Time	1	2	3	4	5	6
L1	9	18	20	24	32	35
L2	12	16	22	25	30	37

Trace	1	2	3	4	5	6
L1		1	2	1	1	2
L2		1	2	1	2	2

Computer science is no more about computers than astronomy is about telescopes.



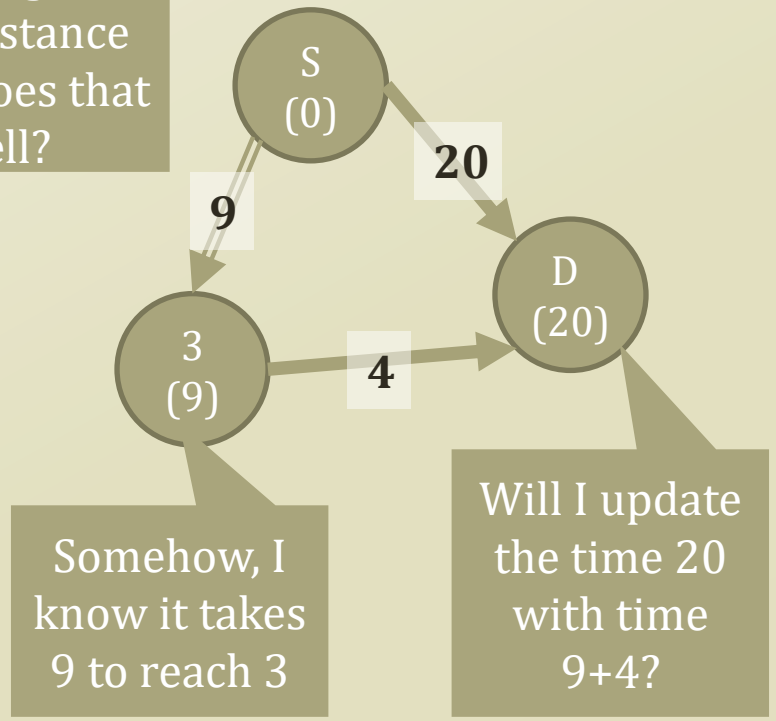
Dijkstra's algorithm



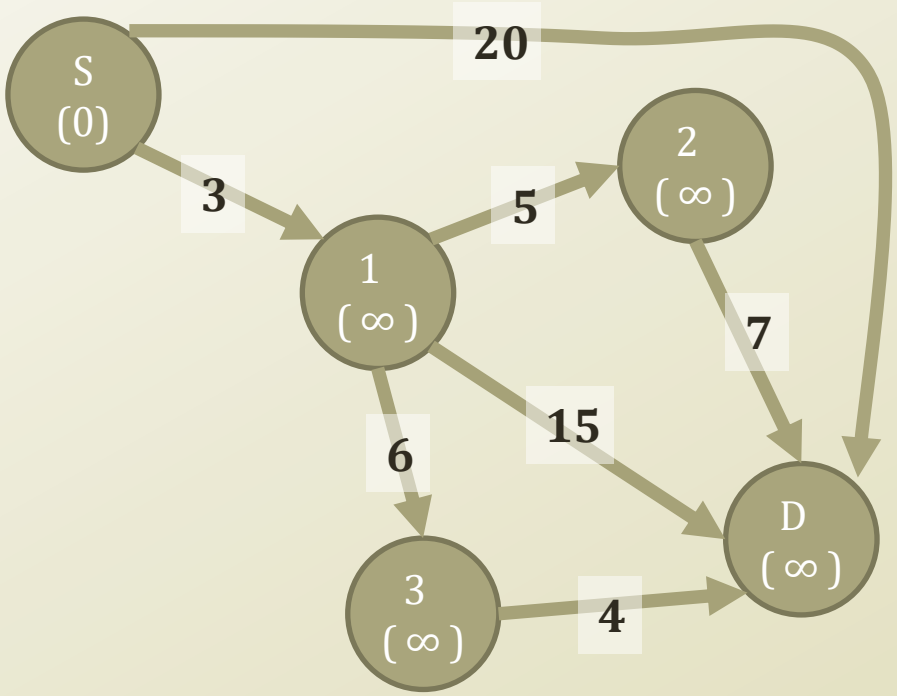
Memoization Table

- V = the set of vertexe
- W = the set of wei ts on edges
- s = the source v rtex
- Dijkstra's algorithm(V, W,s)
  - dist = {}
  - For itr in V
    - dist[v] = 99999
  - dist[s] = 0
  - While size(V) != 0
    - u = getVertexWithMinDistance(V, dist)
    - V.remove(u)
    - For neighbor in getNeighbors(u)
      - If dist[neighbor] > dist[u]+w(u,neighbor)
        - dist[neighbor] = dist[u]+w(u,neighbor)
  - Return dist

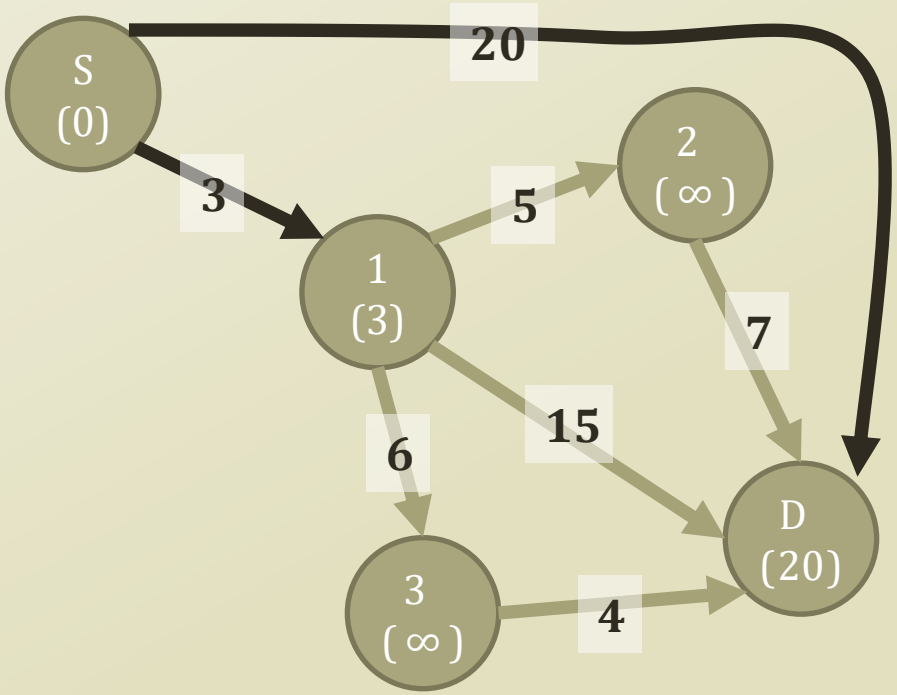
Retrieving minimum distance from a list. Does that ring a bell?



Progress of Dijkstra's algorithm (1)

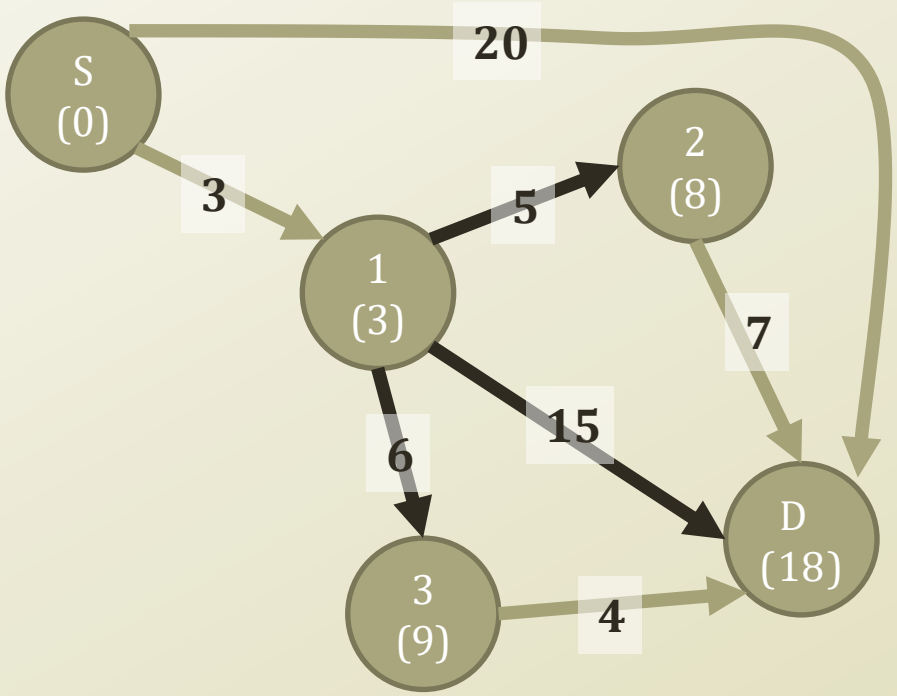


V	S	1	2	3	D
Dist	0	∞	∞	∞	∞

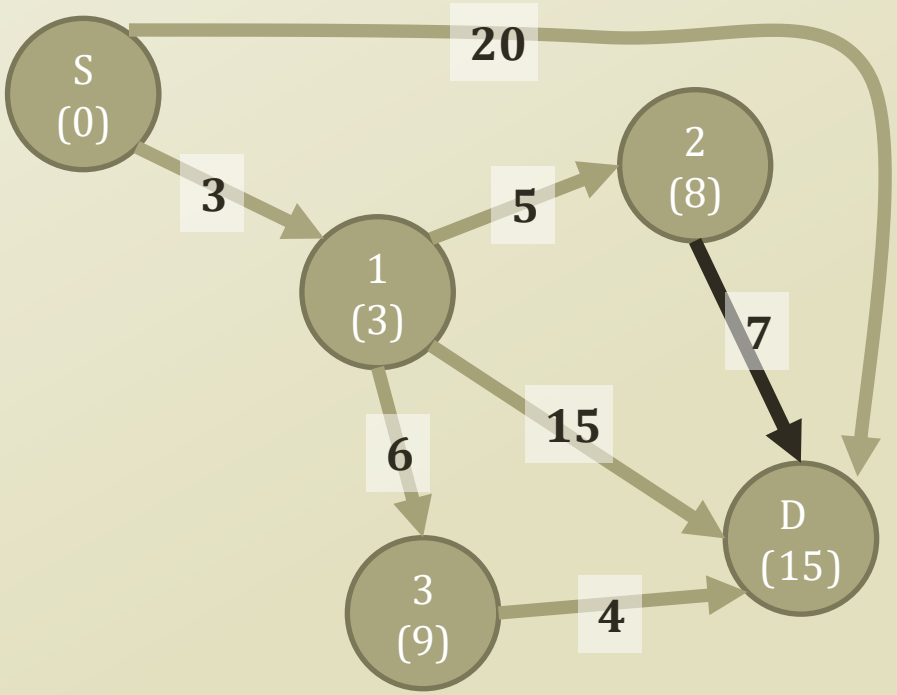


V	S	1	2	3	D
Dist	0	3	∞	∞	20

Progress of Dijkstra's algorithm (2)

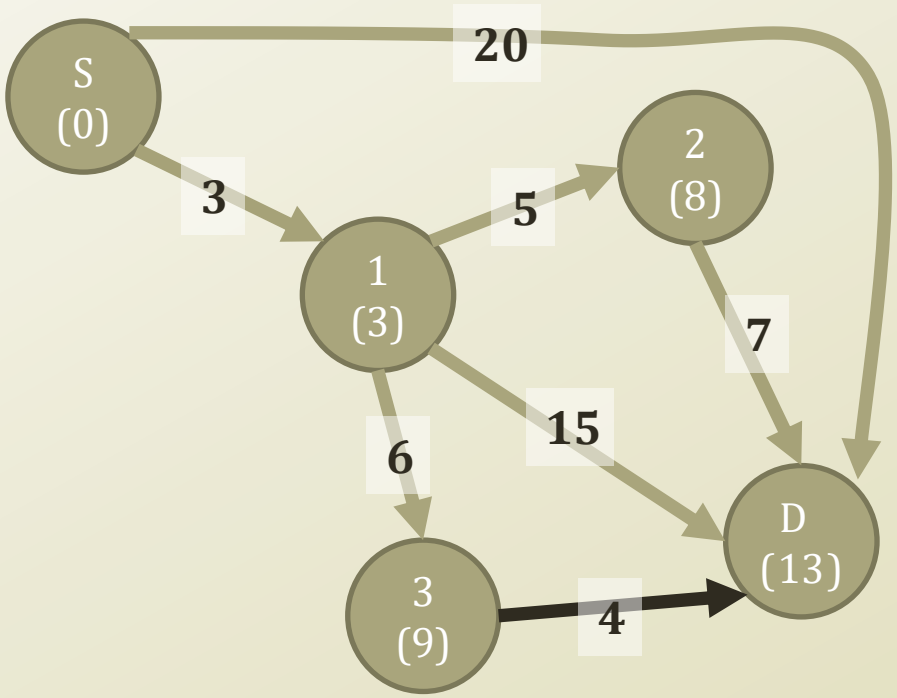


V	S	1	2	3	D
Dist	0	3	8	9	18



V	S	1	2	3	D
Dist	0	3	8	9	15

Progress of Dijkstra's algorithm (3)



V	S	1	2	3	D
Dist	0	3	8	9	13

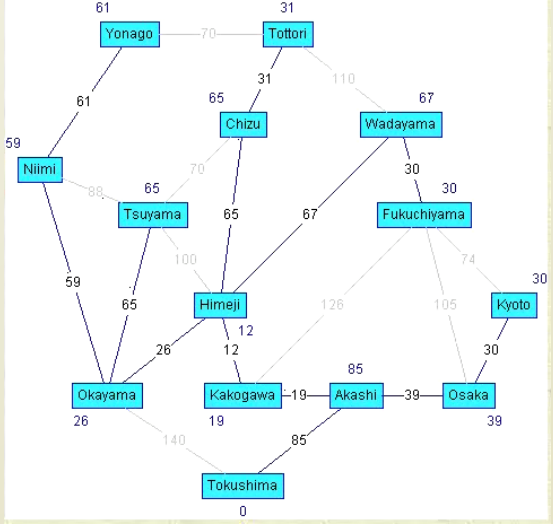
- Time complexity
  - O((|E|+|V|)log|V|)
    - We will not prove this
  - |E|
    - The number can vary
    - It can be close to
      - 1 = dense graph
      - 0 = sparse graph
    - If it is a dense graph,
      - |E| is almost equal to |V| X |V|
      - Then?
      - $O(|V|^2 \log|V|)$
      - More than a quadratic time complexity
      - Pretty expensive!

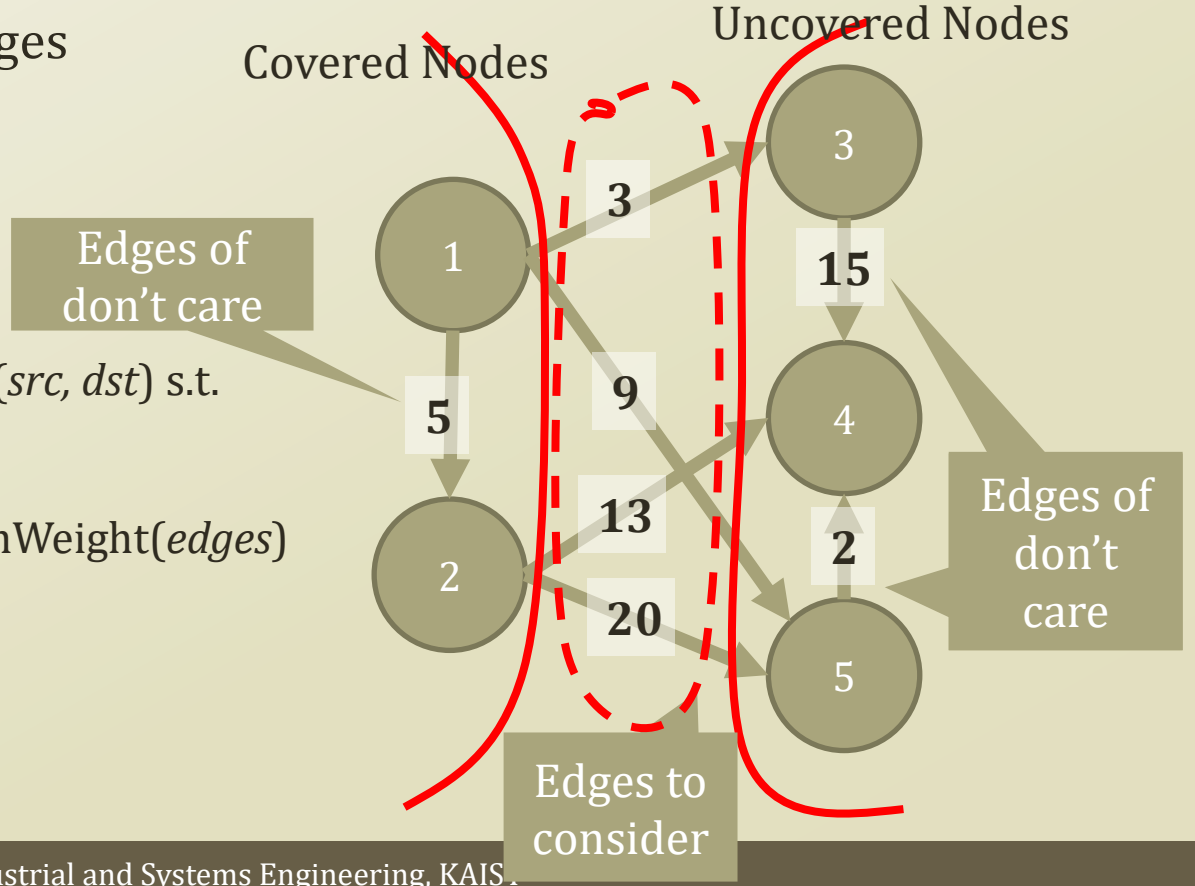
Minimum Spanning Tree Problem

- Shortest-path problem
  - Path planning from a selected source
  - Many times, algorithm for planning
- Minimum spanning tree
  - Network control problem
  - All vertex coverage with minimum cost
  - Algorithm from network design
    - Telephone network
    - Electricity grid network
    - TV cable network
    - Computer network
    - Road network
  - Evolving to the influence propagation tree
    - Social network influence
      - From one politician to all tweeter accounts

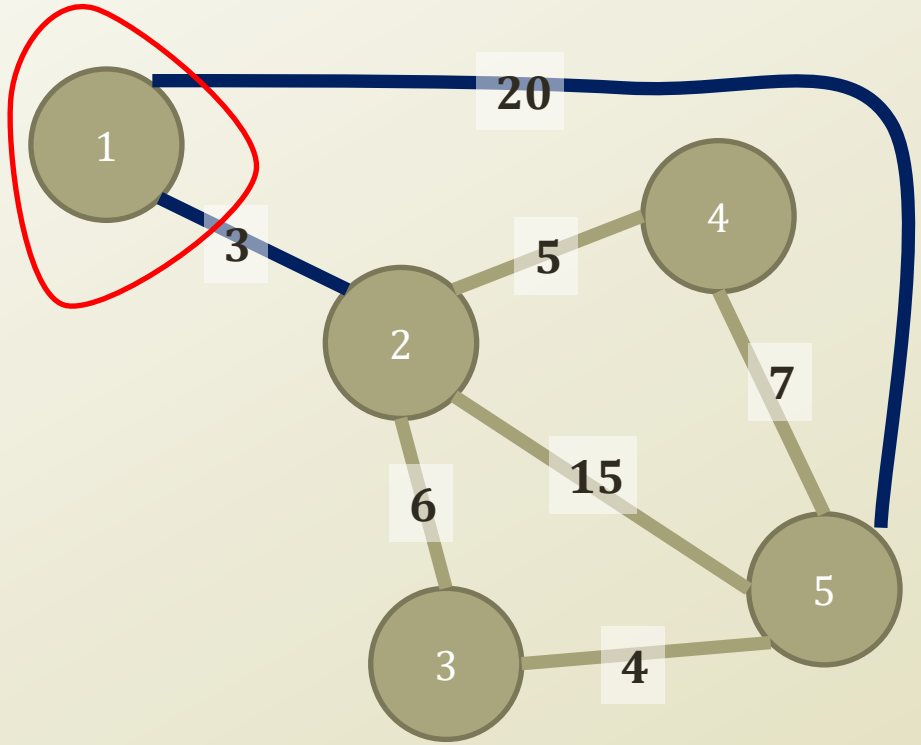
Prim's algorithm

- V = the set of vertexes
- U = the covered set of vertexes
- W = the set of weights on edges
- E = the selected set of edges
- s = the source vertex
- Prim's algorithm(V, W,s)
  - U = {s}, E={}
  - While U == V
    - *edges* = Find edges of (*src, dst*) s.t. $src \in U, dst \in V$
    - e = getEdgeWithMinimumWeight(*edges*)
    - $E = E \cup \{e\}$
    - $U = U \cup \{e.dst\}$
  - Return E and U

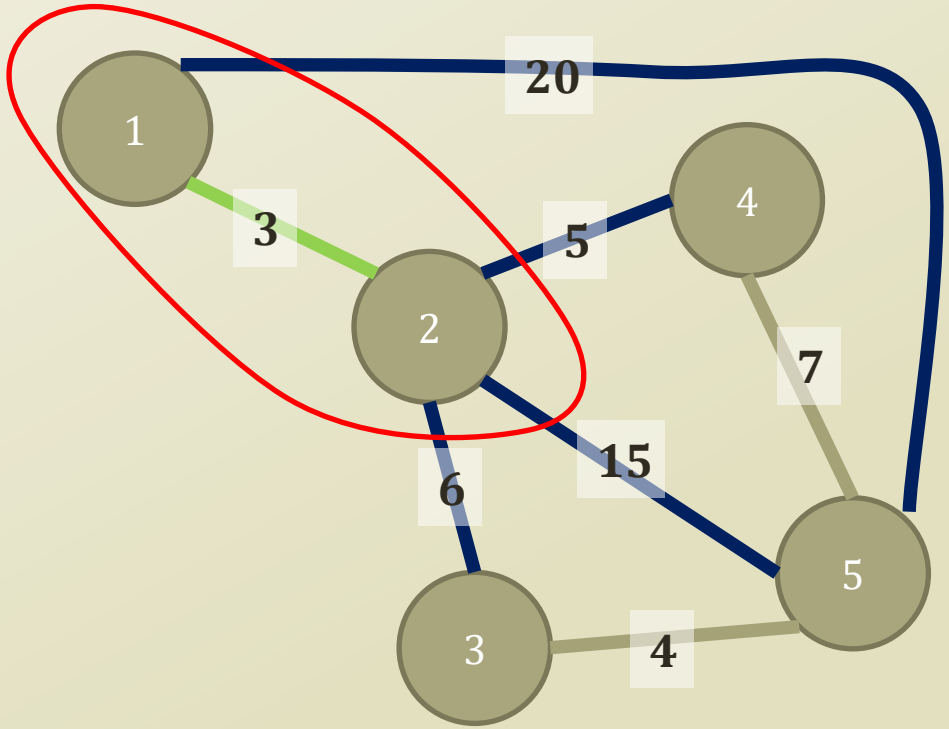




Progress of Prim's Algorithm (1)

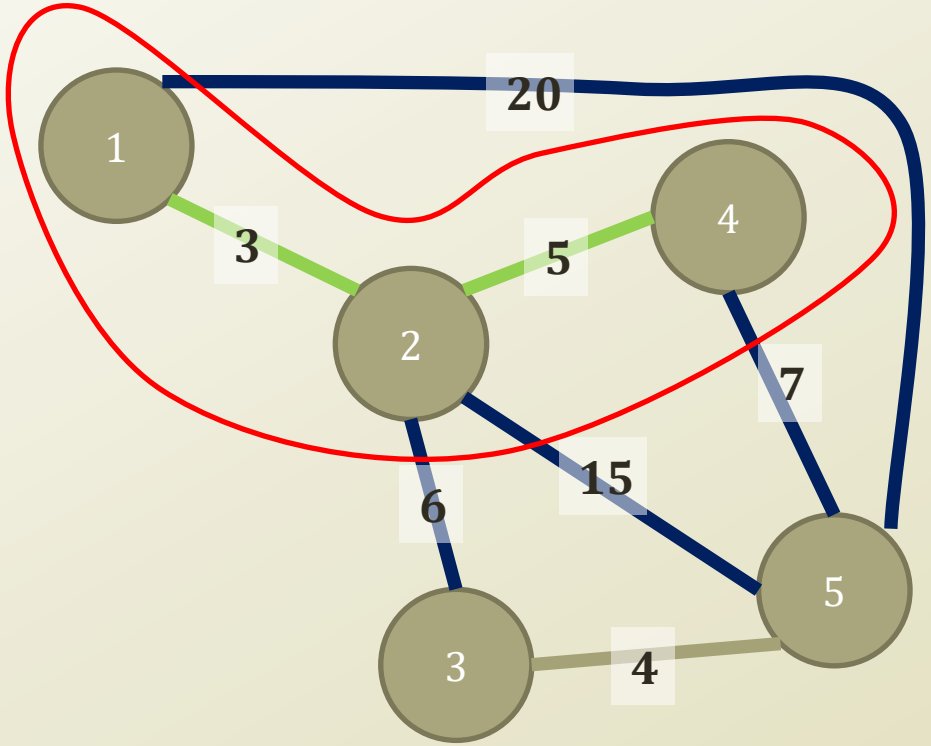


V={1,2,3,4,5}
U={1}
E={}

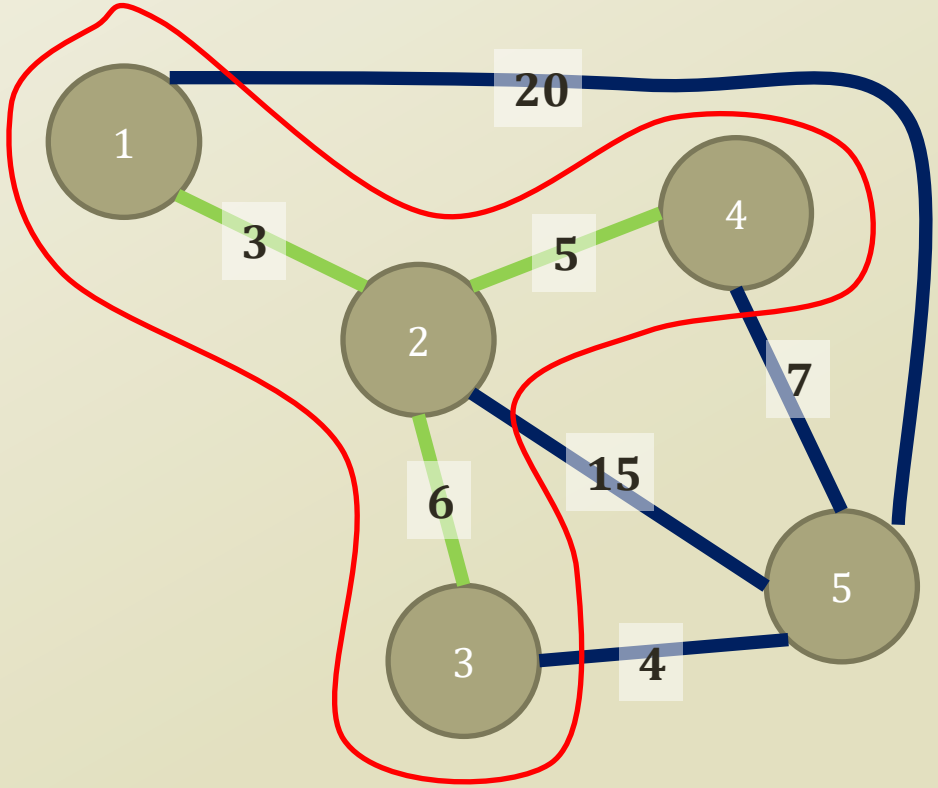


V={1,2,3,4,5}
U={1,2}
E={(1,2)}

Progress of Prim's Algorithm (2)

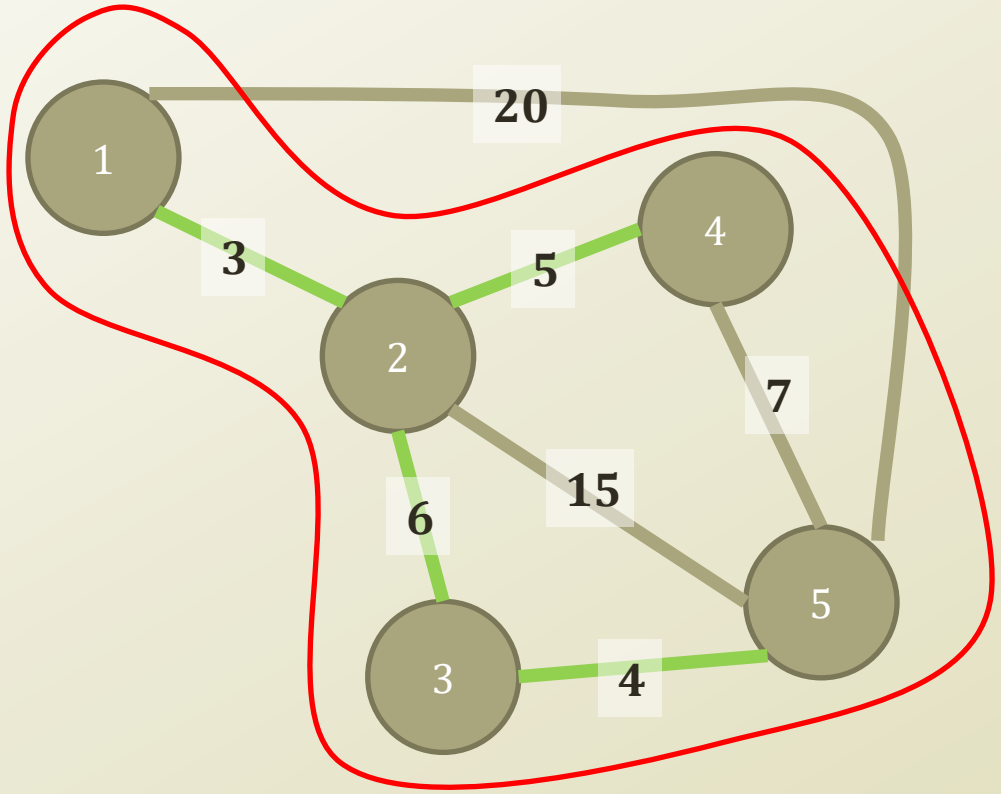


V={1,2,3,4,5}
U={1,2,4}
E={(1,2),(2,4)}



V={1,2,3,4,5}
U={1,2,3,4}
E={(1,2),(2,4),(2,3)}

Progress of Prim's Algorithm (3)



V={1,2,3,4,5}

U={1,2,3,4,5}

E={(1,2),(2,4),(2,3),(3,5)}

- Time complexity
  - O((|E|+|V|)log|V|)
    - We will not prove this
  - Same time complexity to the Dijkstra's algorithm

Further Reading

- Introductions to Algorithms by Cormen et al.
  - pp. 527-619