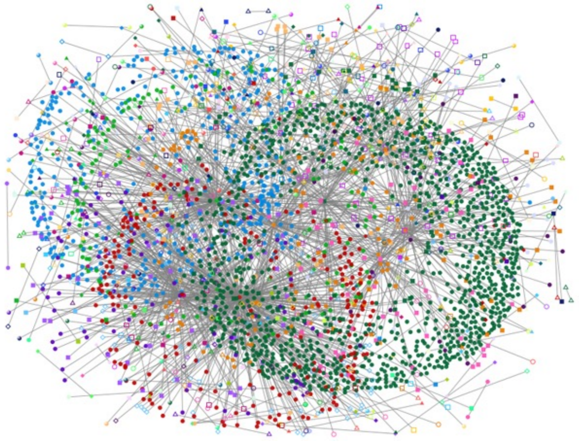
[CSED233-01] Data Structure

Graph Representation

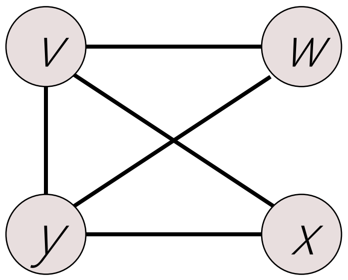
Jaesik Park



POSTECH

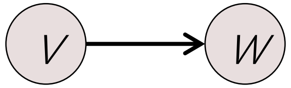
Graph: Terminology

- Graph $G = (V, E)$
  - V : a finite set of vertices/nodes,
    - $n = |V|$: # of vertices
  - E : a finite set of edges/arcs (v, w) where $v, w \in V$
    - $e = |E|$: # of edges



- Example: $G = (V, E)$
  - $V = \{v, w, x, y\}$
  - $E = \{ (v, w), (v, y), (w, y), (y, x), (x, v) \}$

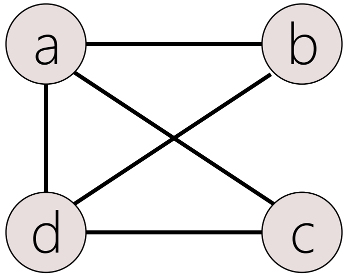
- Two vertices are adjacent if they are connected by an edge
  - v is adjacent to w (w is adjacent from v)
  - (v, w) is incident to w (from v)



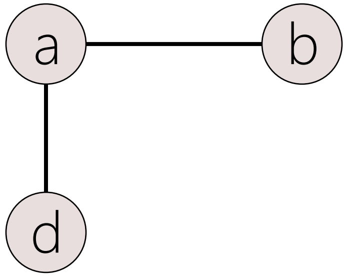
Graph: Terminology

- Types of graphs
  - Complete
    - There is an edge b/w every pair of vertices (denoted K_n)
    - e = $n(n-1)/2$
  - Sparse < Dense (# of edges)
    - Sparse: e = $O(n)$
    - Dense: $e = \Theta(n^2)$
  - Directed ↔ Undirected
    - Edge directionality
  - Weighted (weights on edges)
  - Labeled (labels on vertices)

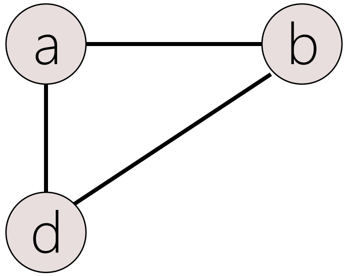
Graph: Terminology



(labeled) graph



subgraph



Induced subgraph

- $G_s = (V_s, E_s)$: a subgraph of G
  - $V_s \subseteq V$
  - $E_s \subseteq E$ such that $(v, w) \in E_s \rightarrow v, w \in V_s$
- $G_s = (V_s, E_s)$: an induced subgraph of G
  - $E_s = \{ (v, w) \in E \mid v, w \in V_s \}$

Difference?

- An induced subgraph includes all the edges that have both endpoints in the inducing set V_s , whereas an ordinary subgraph may miss some.

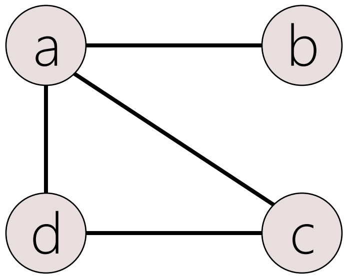
Graph: Terminology

- Path <$v_1, v_2, \ldots, v_n$>
  - A sequence of edges which connect a sequence of vertices
  - Length of path = # of edges
  - Simple – all vertices on the path are distinct: No cycle
- Cycle
  - A path (of length 3 or more) that starts & ends at the same vertex
  - How about a path <u, v, u> in *undirected* graph?
    - Not regarded as a cycle

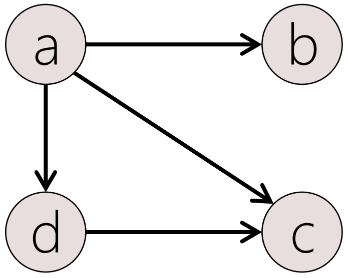
- Self-loop is an edge <v, v> from a vertex to itself
  - Length = 0
  - Generally, loop-less graphs in this course

Graph: Terminology

- Acyclic - without cycles
  - Directed acyclic graph (DAG)



cyclic



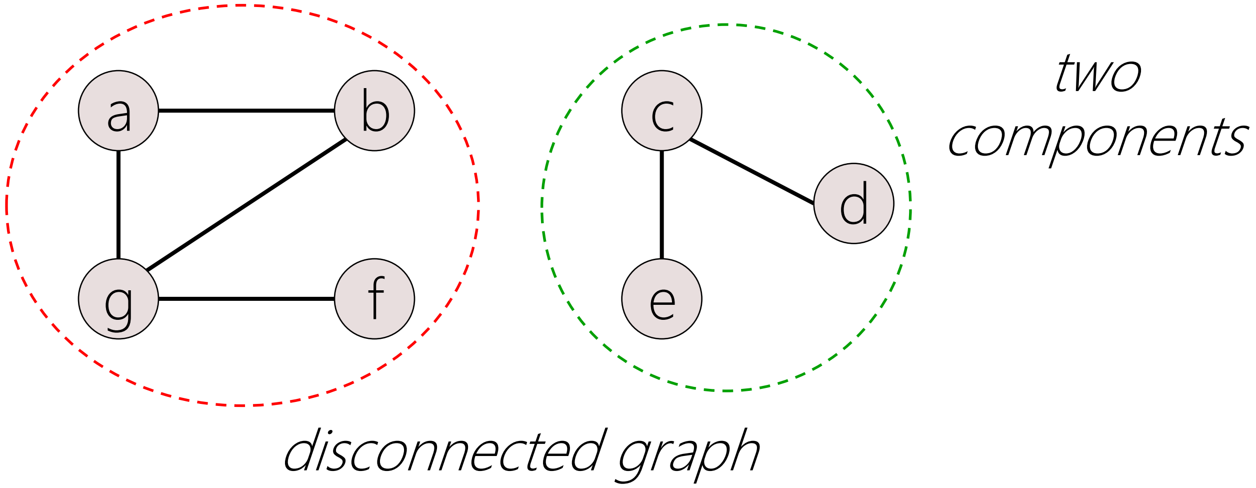
acyclic

Not strongly but weakly connected

- Connected graph G
  - In an undirected graph
    - If there is a path b/w any two vertices
  - In a directed graph
    - G is called strongly connected
    - G is weakly connected
      - if the underlying graph (without directions on the arcs) is connected

Graph: Terminology

- Connected component
  - In an undirected graph
    - A maximal subgraph that is connected
    - G is connected $\leftrightarrow$ G has exactly 1 component
  - In a directed graph
    - it is called a strongly connected component (or just strong component)



Graph: Terminology

- Tree
  - An undirected graph G, satisfying any of the following equivalent conditions:

    - G is connected & acyclic
    - G is connected & has n vertices with $n-1$ edges

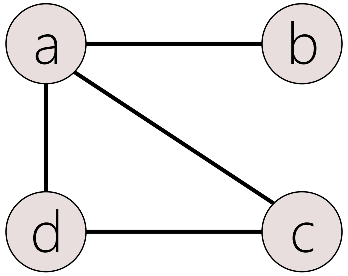
  - If any edge is added to a tree, we get a cycle
  - If any edge is removed from a tree, the graph becomes disconnected

Graph: Terminology

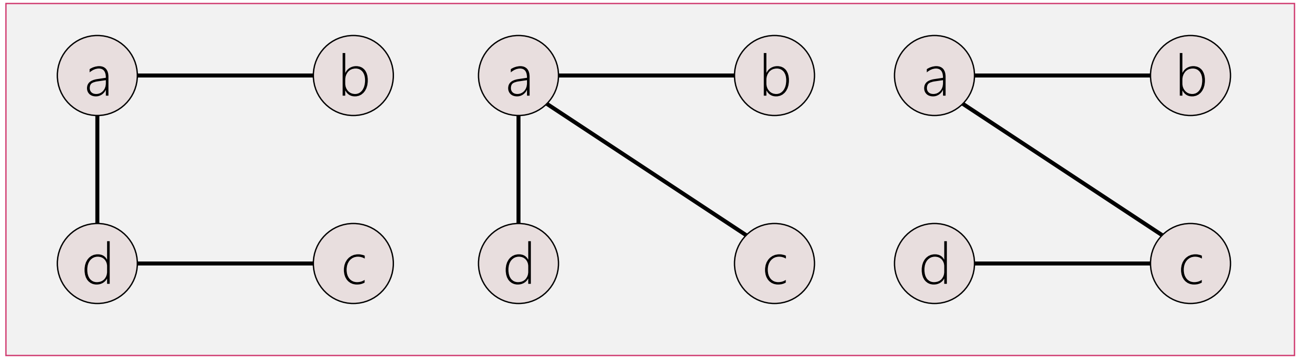
- Spanning tree T of a connected graph *G*
  - A tree T that includes all vertices of the original graph *G*

- Example:

Graph



Spanning Trees

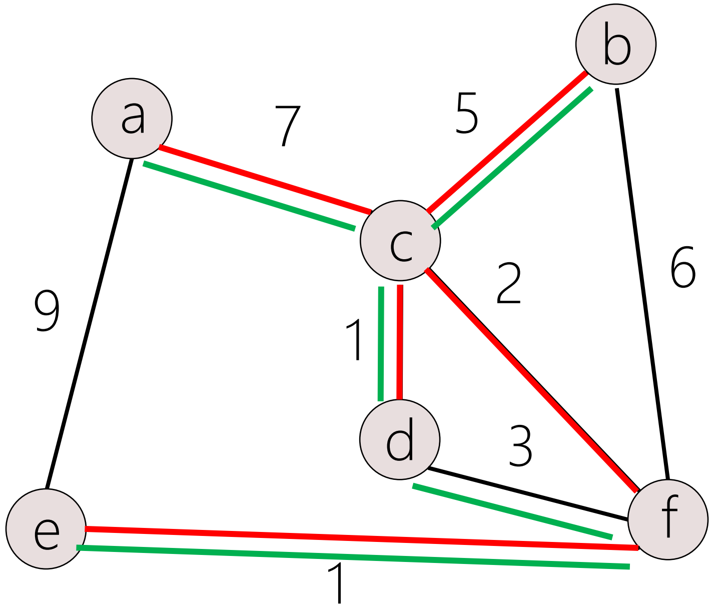


Graph: Terminology

- Minimum-cost spanning tree (MST)
  - A spanning tree whose tree cost is minimum
  - Tree cost is a sum of edge weight/cost

- Example:

Labeled, weighted, connected graph



MST-1 with cost = 16

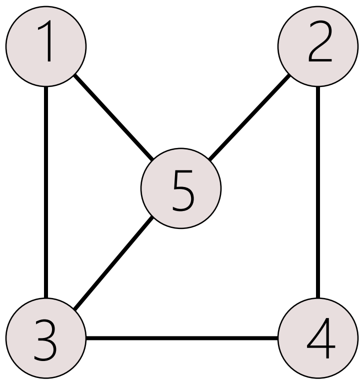
MST-2 with cost = 17

Graph Representations

- Two commonly used methods
  - Adjacency matrix
  - Adjacency lists
    - Linked adjacency lists
    - Array adjacency lists

Adjacency Matrix: Undirected Graph

- Binary *n x n* matrix (*n*: # of vertices)
  - $A(i,j) = 1$ iff (i, j) is an edge

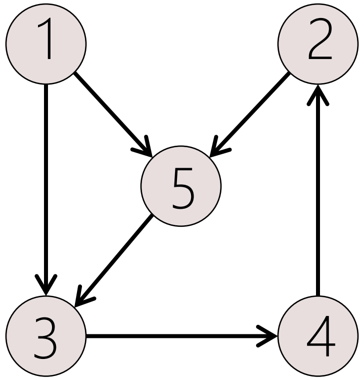


	1	2	3	4	5
1	0	0	1	0	1
2	0	0	0	1	1
3	1	0	0	1	1
4	0	1	1	0	0
5	1	1	1	0	0

- Diagonal entries = zero
- Symmetric: $A(i, j) = A(j, i)$ for all i, j

Adjacency Matrix: Directed Graph

- Binary *n x n* matrix (*n*: # of vertices)
  - $A(i,j) = 1$ iff (i, j) is an edge

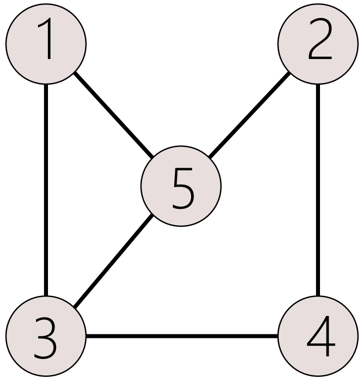


	1	2	3	4	5
1	0	0	1	0	1
2	0	0	0	0	1
3	0	0	0	1	0
4	0	1	0	0	0
5	0	0	1	0	0

- Diagonal entries = zero
- Need not be symmetric

Adjacency Lists

- An array of *n* adjacency lists
  - An adjacency list for vertex *v* = a linear list of vertices adjacent from *v*

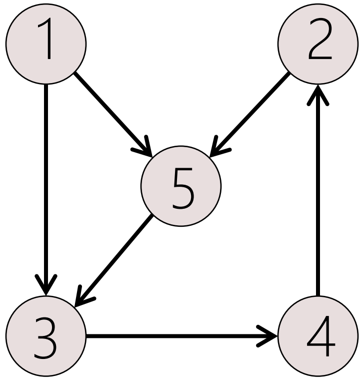


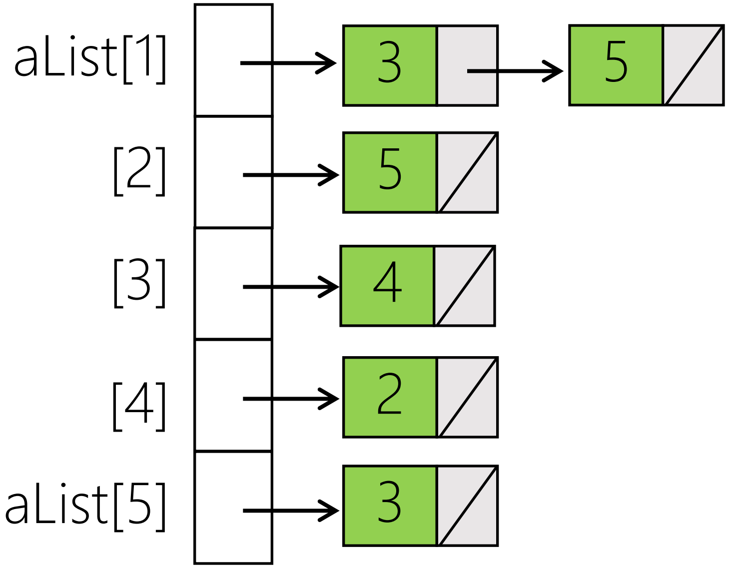
aList[1] = (3, 5)
aList[2] = (4, 5)
aList[3] = (1, 4, 5)
aList[4] = (2, 3)
aList[5] = (1, 2, 3)

- Two implementations of lists
  - Linked vs. Array

Adjacency Lists: Linked

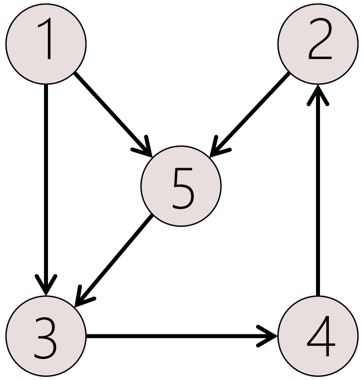
- Each adjacency list is a chain

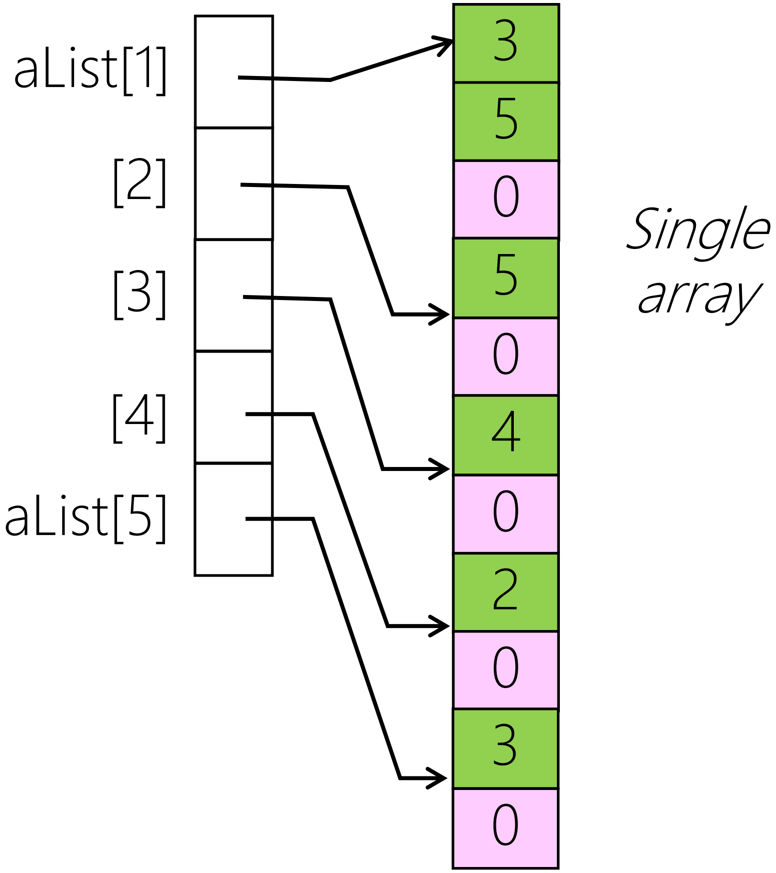




Adjacency Lists: Single Array

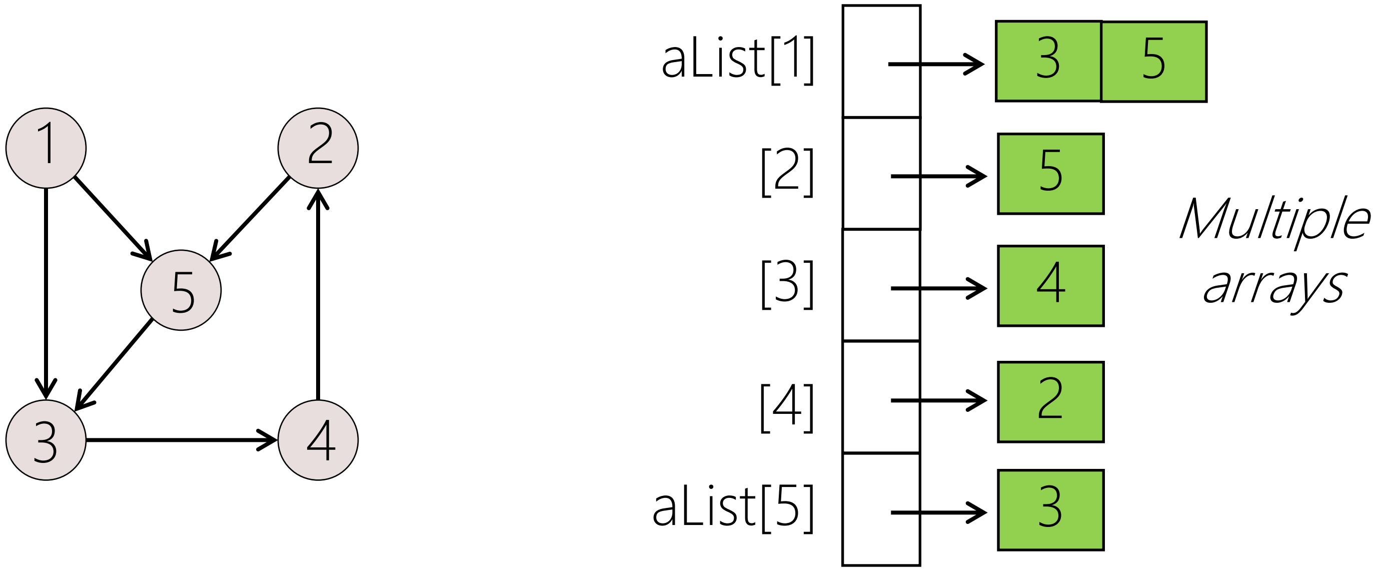
- If the graph were expected to remain fixed
  - Use a single array for all adjacency lists





Adjacency Lists: Multiple Arrays

- Each adjacency list is an array



References

- Further reading list and references
  - https://www.geeksforgeeks.org/difference-between-graph-and-tree/
  - https://www.geeksforgeeks.org/strongly-connected-components/

- Slide credit
  - Jaesik Park
  - Seung-Hwan Baek
  - Jong-Hyeok Lee