# 実践的アルゴリズム理論
# Theory of Advanced Algorithms

## グラフのデータ構造

担当:上原隆平

# Theory of Advanced Algorithms実践的アルゴリズム理論

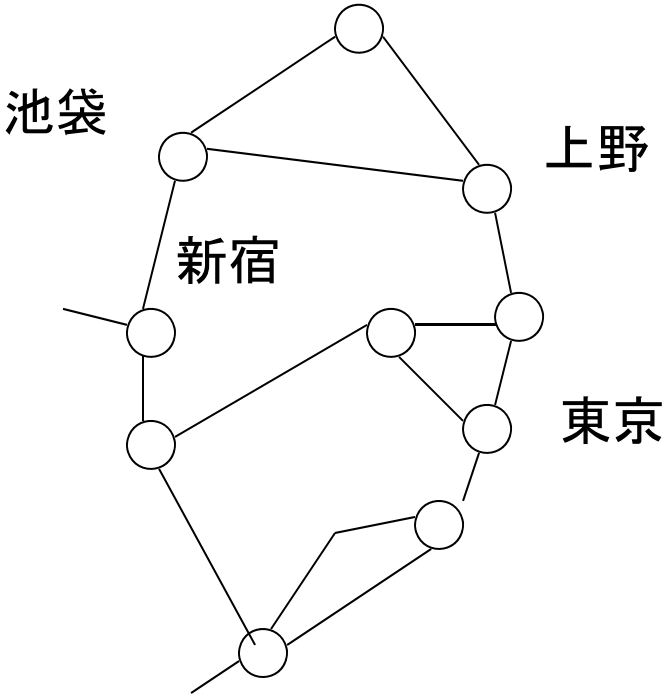
## Data Structure of Graphs

**Ryuhei Uehara**

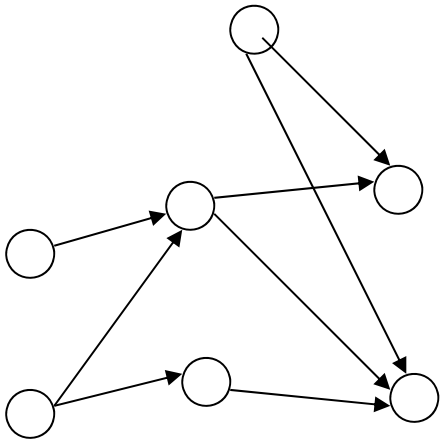
# グラフ (graph)

- 頂点を辺で結んだもの
  - 有向グラフ: 辺に方向がある
  - 無向グラフ: 辺に方向がない

例: 路線図



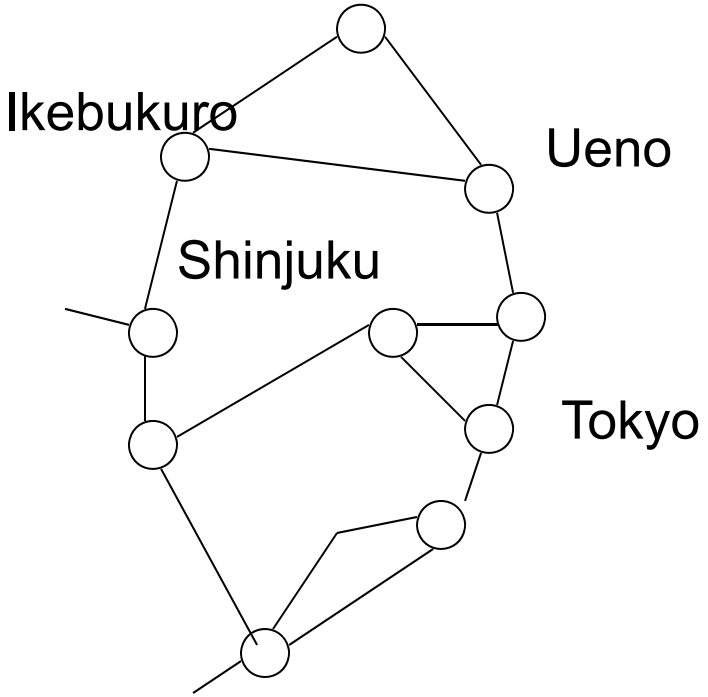

例: 講義の履修順序



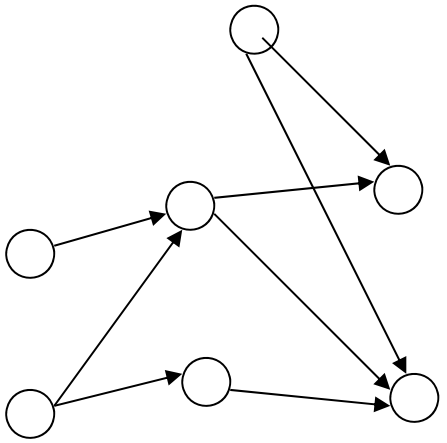
# Graph

- Vertices are joined by edges
  - Directed graph: each edge has direction
  - (Undirected) graph: each edge has no direction

Example: Railmap

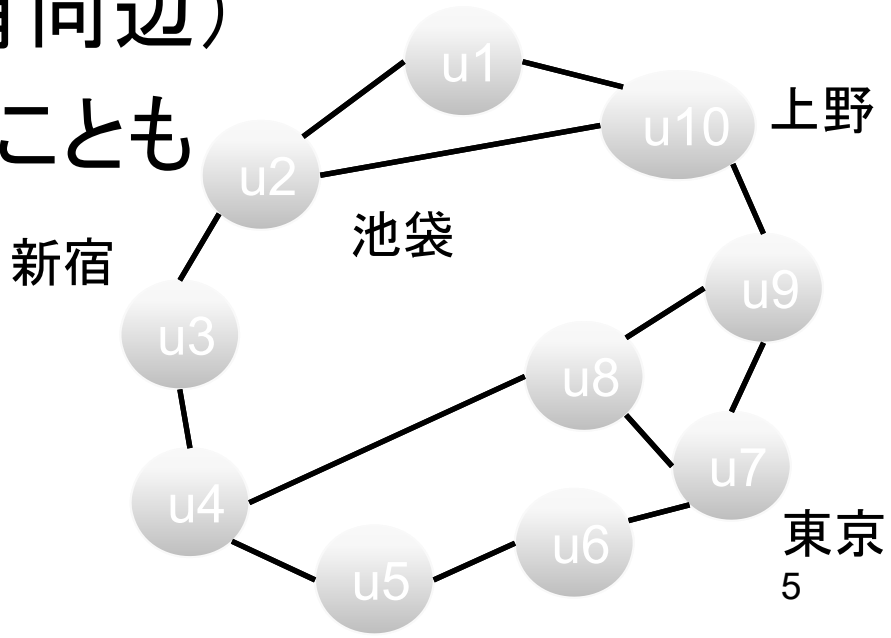



Example: Order of classes



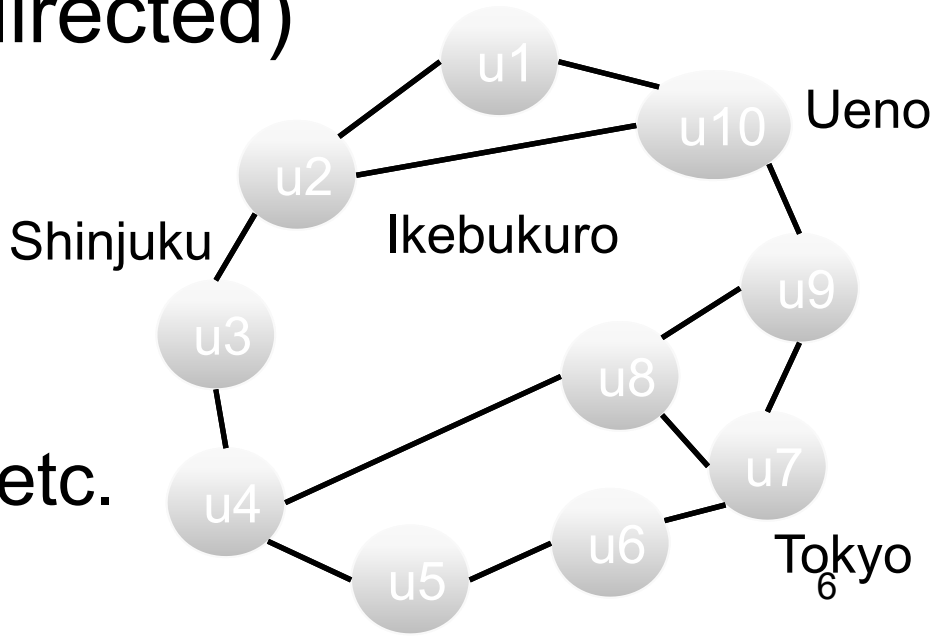
# グラフ: 表記

- グラフ: $G = (V, E)$
  - $V$: 頂点集合, $E$: 辺集合
- 頂点: $u, v, \ldots \in V$
- 辺: $e = \{u, v\} \in E$ (無向辺)
  $a = (u, v) \in E$ (有向辺)
- 頂点や辺は重みを持つことも
  - $w(u), w(e)$
  - 距離, 金額, 時間など



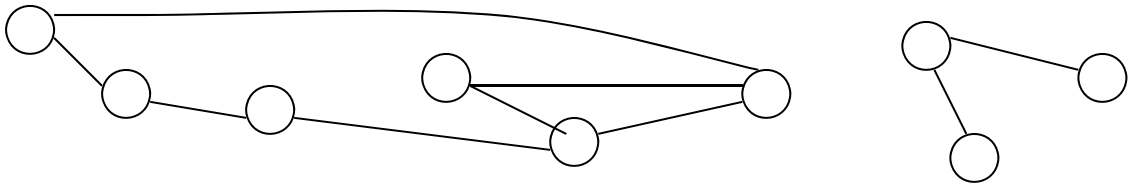
# Graph: Notation

- Graph: $G = (V, E)$
  - $V$: Vertex set, $E$: Edge set
- Vertices: $u, v, \ldots \in V$
- Edges: $e = \{u, v\} \in E$ (undirected)
  $a = (u, v) \in E$ (directed)
- Vertices/edges can have weight
  - $w(u)$, $w(e)$
  - Distance, price, time, etc.



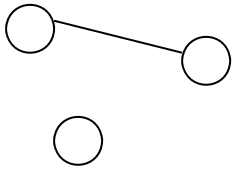
# グラフ: 基本的な用語 (1/2)

- 路 (path): 辺で隣り合う頂点を繋いだもの
  - 単純路 (simple path): 同じ頂点を複数回通らない



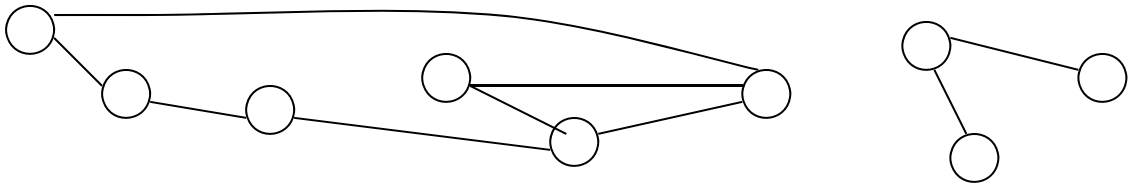


- 閉路 (cycle, closed path): $v$ から $v$ への路
- 連結グラフ (connected graph): どの2頂点間にも路が存在するグラフ



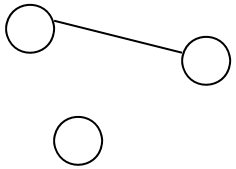
# Graph: Basic terms (1/2)

- Path: Vertices joined by edges
  - Simple path: never visits the same vertex twice



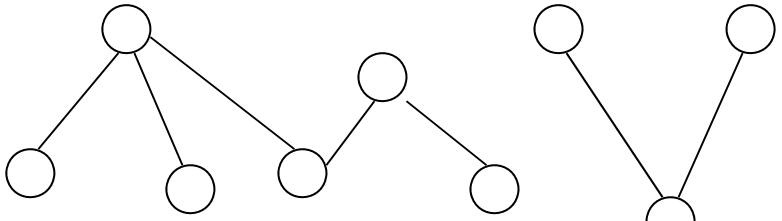


- cycle, closed path: path from *v* to *v*
- connected graph: Any pair of vertices is joined by a path

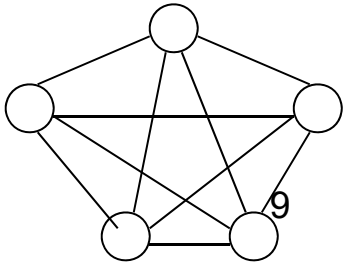


# グラフ: 基本的な用語 (2/2)

- 森 (forest): 閉路を含まないグラフ
- 木 (tree): 連結で閉路を含まないグラフ

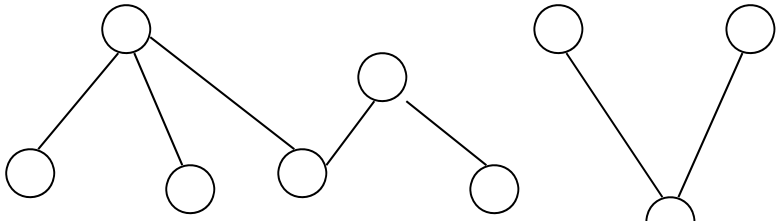


- 平面的グラフ (planar graph): 辺の交差なしで平面に描画できるグラフ
- 完全グラフ (complete graph): 全ての頂点対を辺で隣接させたグラフ
  - 完全グラフ $K_5$ は極小非平面的グラフ

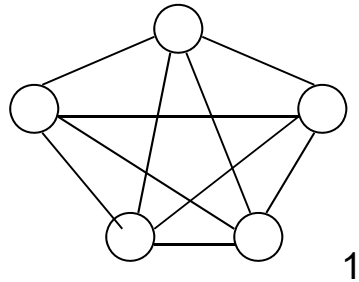


# Graph: Basic terms (2/2)

- forest: acyclic graph (no cycle)
- tree: connected acyclic graph



- planar graph: It can be drawn on a plane without crossing of edges
- complete graph: graph s. t. all pairs are joined
  - Complete graph $K_5$ is
    a minimal non-planar graph



# グラフアルゴリズムの計算量

- 頂点数 $n$, 辺数 $m \in O(n^2)$
  - 無向グラフ: $m \leqq n(n\text{-}1)/2$
  - 有向グラフ: $m \leqq n(n\text{-}1)$

- 平面グラフでは $m \in O(n)$

- グラフアルゴリズムの計算量は
  $n$ や $m$ の式で表す

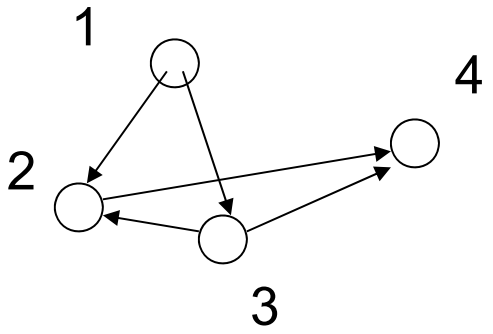
# Complexity of graph algorithms

- $n$ vertices and $m$ edges; $m \in O(n^2)$
  - undirected graph: $m \leqq n(n\text{-}1)/2$
  - directed graph: $m \leqq n(n\text{-}1)$
- On a plane graph, $m \in O(n)$
- Computational complexity of a graph algorithm is given by an expression of $n,m$

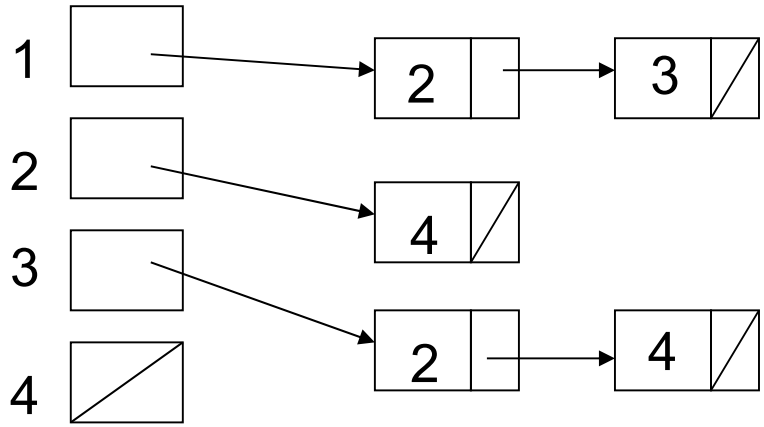
# グラフの表現方法

- 隣接行列

$$M = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$




- 隣接リスト



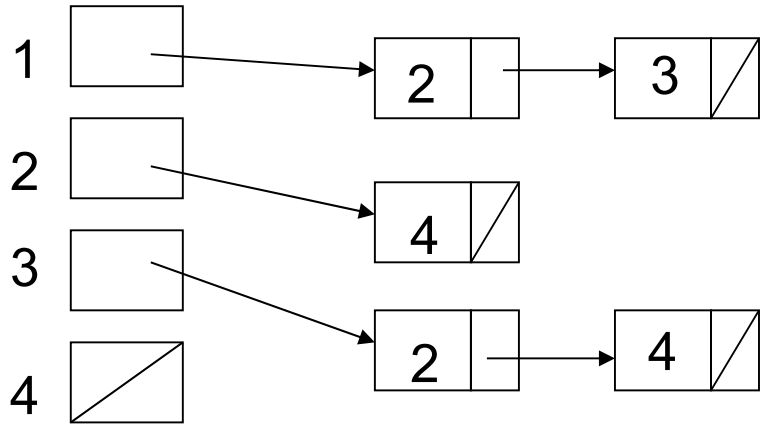

頂点　　隣接頂点のリスト

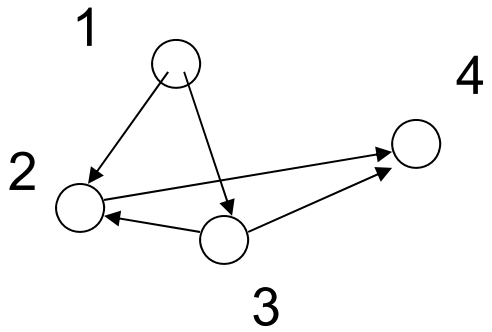
# Representation of a graph

- Adjacency matrix

$$M = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

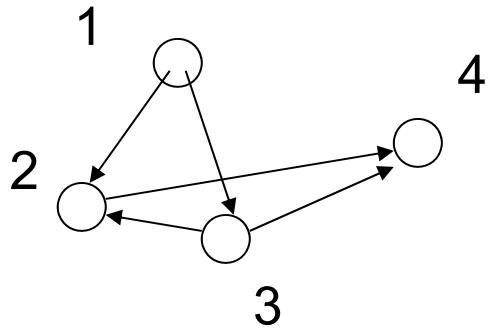
- Adjacency list



# グラフの表現方法:
# 行列表現(隣接行列)

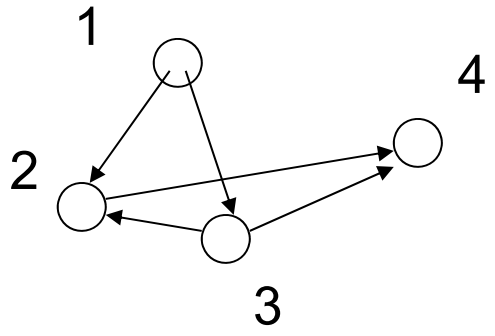
- $(u, v) \in E \Rightarrow M[u, v] = 1$
- $(u, v) \notin E \Rightarrow M[u, v] = 0$




$$M = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

# Representation of a graph: Matrix representation (adj. matrix)

- $(u, v) \in E \Rightarrow M[u, v] = 1$
- $(u, v) \notin E \Rightarrow M[u, v] = 0$

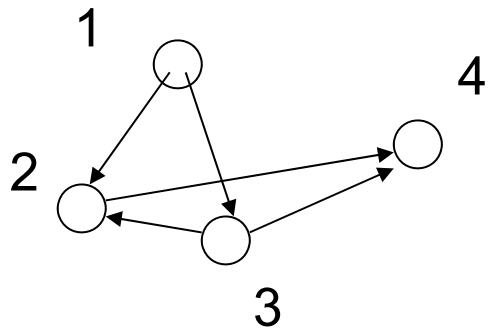


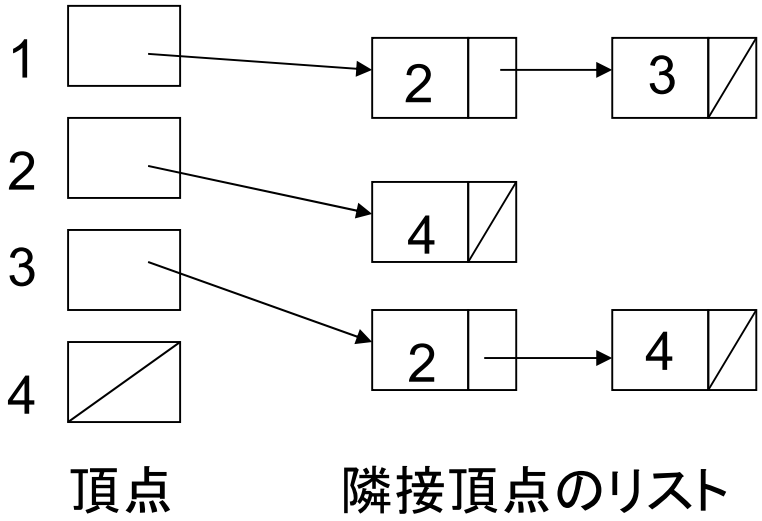

$$M = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

# グラフの表現方法: リスト表現(隣接リスト)

- $(u, v) \in E \Leftrightarrow v \in T(u)$
  - $T(u)$ は $u$ の隣接点のリスト

- 例:

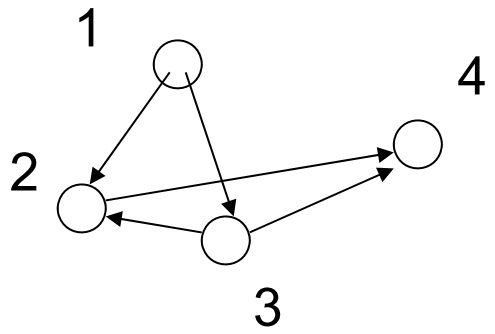


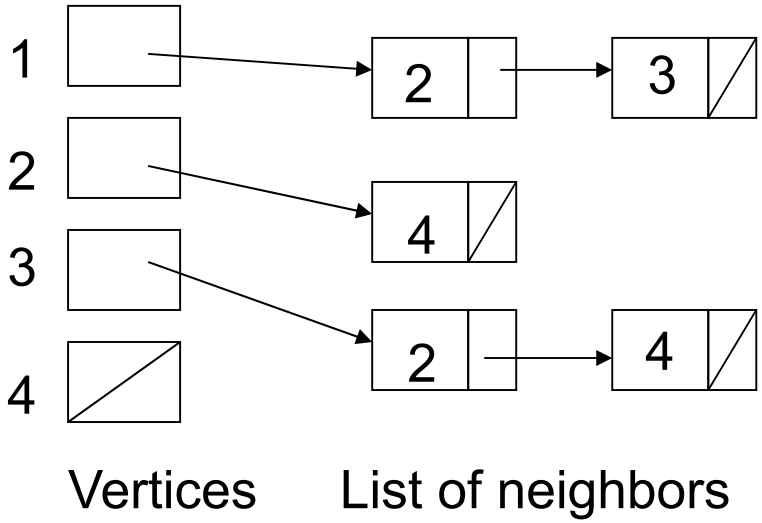



# Representation of a graph: List representation (adj. list)

- $(u, v) \in E \Leftrightarrow v \in T(u)$
  - $T(u)$ is the list of neighbors of $u$

- Example:



# 隣接行列 vs 隣接リスト

- メモリ量
  - 隣接行列: $\Theta(n^2)$
  - 隣接リスト: $\Theta(m \log n)$
- 隣接チェックにかかる時間: $(u, v) \in E$ ?
  - 隣接行列: $\Theta(1)$
  - 隣接リスト: $\Theta(n)$

**Q. グラフの更新 (e.g., 頂点・辺の追加・削除) は?**

# Adj. Matrix vs Adj. List

- Memory
  - Adj. matrix: $\Theta(n^2)$
  - Adj. list: $\Theta(m \log n)$
- Time for check if $(u, v) \in E$ ?
  - Adj. matrix: $\Theta(1)$
  - Adj. list: $\Theta(n)$

**Q. Update of graph?**
**(e.g., add/remove of vertex/edge)**