

グラフを記述するための データ構造

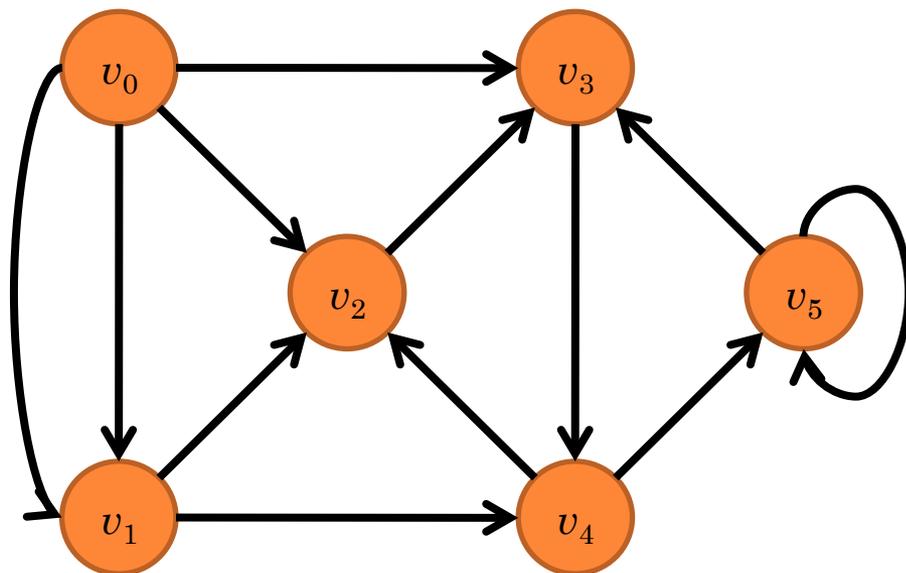
グラフを記述するには何が必要か

- 探索問題を考えよう
 - 最短経路
 - 閉路探索
 - 頂点の数が多くなると、全体の情報を一度に扱えない
- アルゴリズム・プログラムで扱えるようにするには
 - 局所的(ある頂点から出る弧、弧の両端の頂点)な視点
 - 再帰的(recursive)手法が素直な方法



頂点と弧の関連を調べる

- 頂点から弧を経由して移動するための情報
 - 頂点を始点とする弧の一覧
- 弧の両端の頂点を知るための情報
 - 弧からその両端の頂点を得る
 - 弧の終点を得て、次のステップへ進む



行列を使った記述方法

- 伝統的手法
- 隣接行列
 - 頂点の間の連結状況
- 接続行列
 - 弧と頂点の関連付け

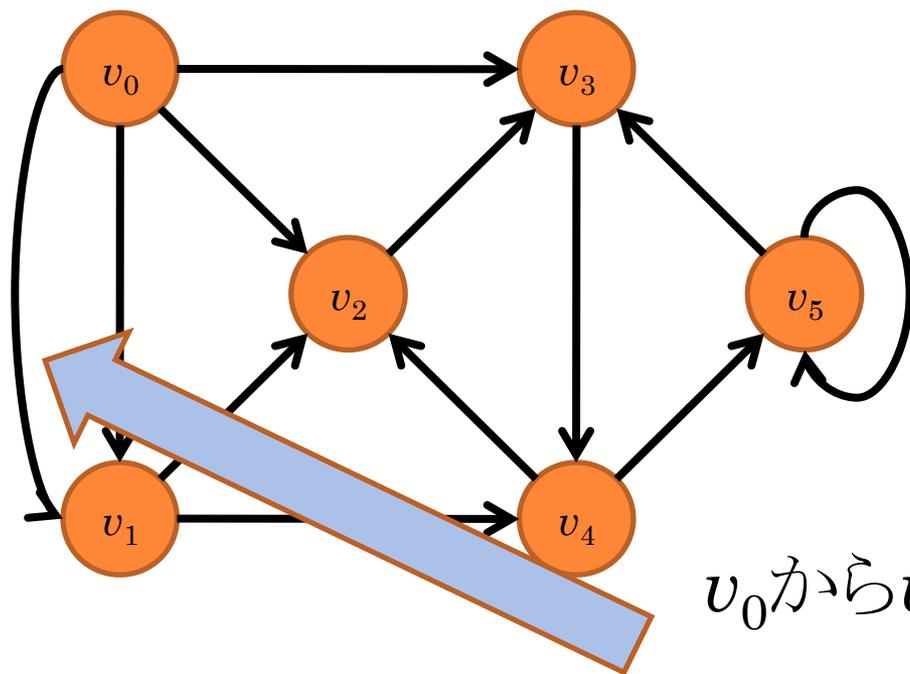
- 簡潔な表現という利点がある
- 無駄が多いという問題点
- 不足する情報あり



隣接行列 (ADJACENCY MATRIX)

- 各要素: 頂点間の連結の有無を示す: 向きに注意

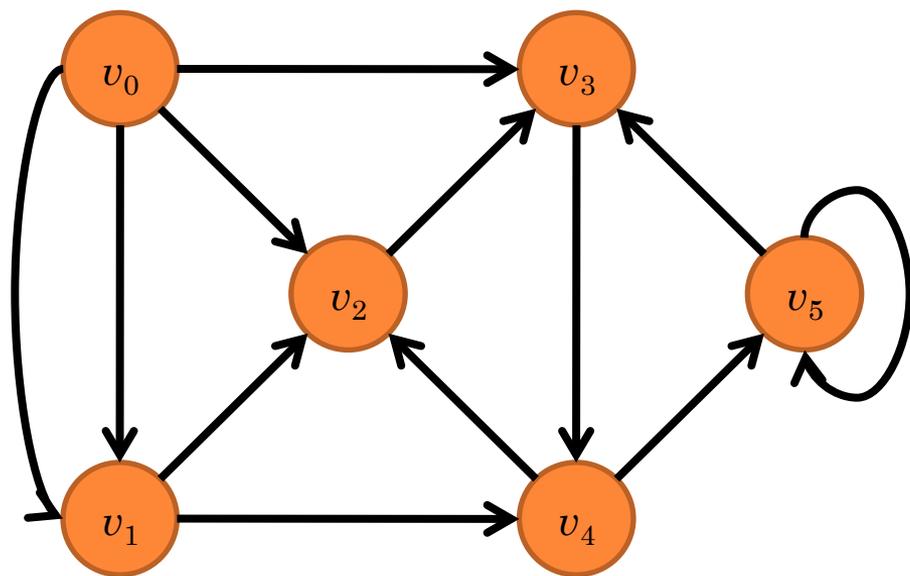
$$\partial^+ a_i = v_j, \partial^- a_i = v_k \rightarrow \Gamma_{kj} = 1$$



$$\Gamma = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

v_0 から v_1 への並列弧の表現? ●

隣接行列の活用: 道の数



$$\Gamma^2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 2 & 1 \\ 3 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

複数の道の存在

$$v_0 \rightarrow v_4$$

$$v_4 \rightarrow v_3$$



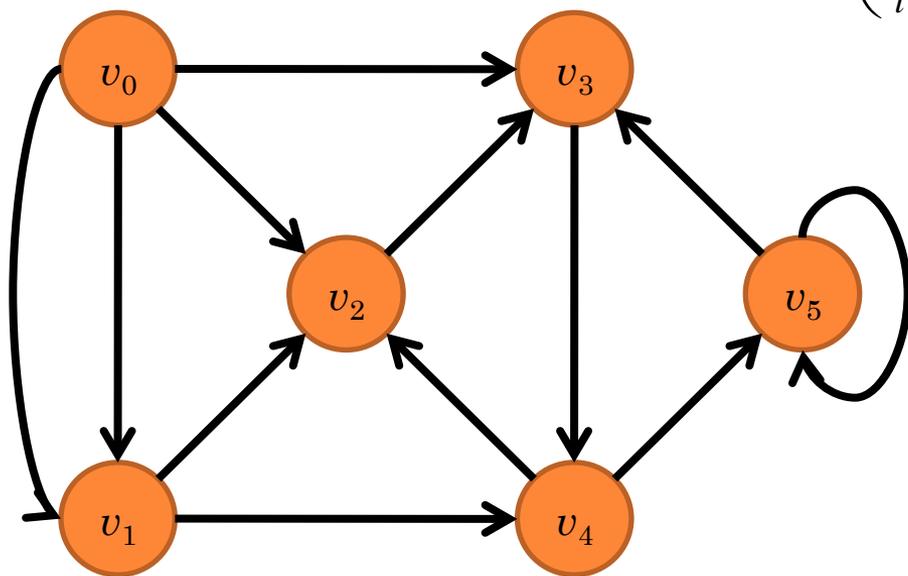
隣接行列の活用: 道の有無

- N 個の頂点からなるグラフ
 - 頂点 j から i への道の存在

$$\left(\sum_{l=1}^{N-1} \Gamma^l \right)_{ij} = \begin{cases} > 0 & \text{道が存在} \\ 0 & \text{道が無い} \end{cases}$$



隣接行列の活用: 連結性



$$\left(\sum_{l=1}^5 \Gamma^l \right) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 9 & 5 & 1 & 3 & 4 & 3 \\ 15 & 7 & 4 & 4 & 9 & 10 \\ 12 & 5 & 3 & 4 & 4 & 6 \\ 13 & 8 & 3 & 6 & 10 & 11 \end{pmatrix}$$



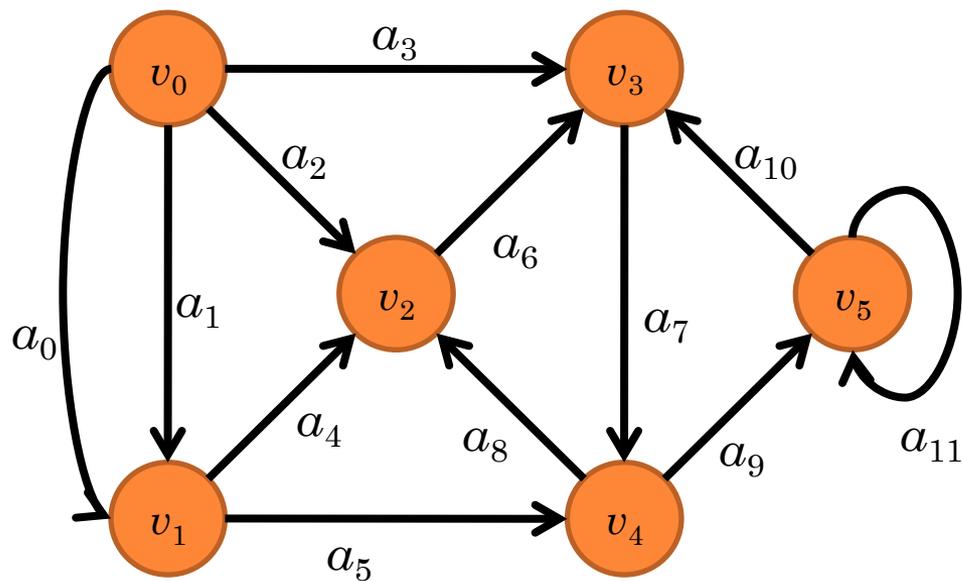
接続行列(INCIDENCE MATRIX)

- 弧と頂点の関連付け

$$D(v, a) = \begin{cases} +1 & \text{if } \partial^+ a = v \wedge \partial^- a \neq v \\ -1 & \text{if } \partial^- a = v \wedge \partial^+ a \neq v \\ 0 & \text{otherwise} \end{cases}$$



接続行列の例



孤立弧の情報が無い

$$D = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \end{pmatrix}$$



接続行列の活用: グラフ上の定常的流れ

- 弧集合上の関数: 流れの定義

$$\varphi: A \rightarrow R$$

- 定常的な流れ

- 各頂点への流れ込む量と流れ出る量が釣り合う

$$(D\varphi)_v = \sum_a D(v, a)\varphi(a) = 0$$



行列表現の利点と問題点

- 演算という観点では便利
 - 道の有無、定常的な流れ
 - 大規模なグラフの性質
- ほとんどの要素がゼロ
 - 情報量が少ない:sparse (薄い、貧弱な)
- 接続行列は孤立弧を表現出来ない
- 弧や頂点に属性が付く場合は、データ構造として対応できない



もっと柔軟な方法:リスト構造

- 柔軟なデータ構造:リスト構造
 - サイズが可変
 - 必要なデータだけを保持
 - リストの要素にもデータ構造を保持できる:属性など
- 柔軟なアルゴリズム:再帰
 - 図としてのグラフの探索を素直に実装



リスト表現の利点

- 現代的なプログラミング言語はもっと多様なデータ構造が扱えるではないか
 - 柔軟なデータ構造とアルゴリズム
- 各頂点を始点とする弧の一覧(リスト)を作成する
 - 必要な分だけの情報
 - 並列弧・孤立弧が両方とも扱える
- 弧は始点と終点の情報を知っている



例:再帰的探索

頂点 v に関する関数

$f(v)$ {

forall(v を始点とする弧 a) {

if(終了条件){return}

頂点 w は弧 a の終点

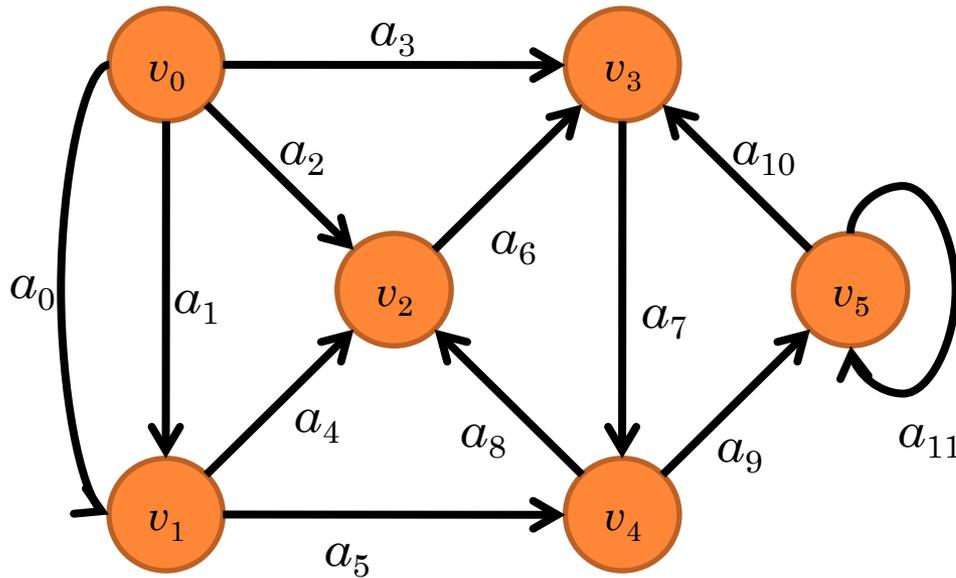
$f(w)$

}

}



グラフの探索例



$$\begin{aligned} \delta^+ v_0 &= \{a_0, a_1, a_2, a_3\} & \delta^+ v_1 &= \{a_4, a_5\} \\ \delta^+ v_2 &= \{a_6\} & \delta^+ v_3 &= \{a_7\} \\ \delta^+ v_4 &= \{a_8, a_9\} & \delta^+ v_5 &= \{a_{10}, a_{11}\} \end{aligned}$$

$\partial^+ a_0 = v_0$	$\partial^- a_0 = v_1$	$\partial^+ a_1 = v_0$	$\partial^- a_1 = v_1$
$\partial^+ a_2 = v_0$	$\partial^- a_2 = v_2$	$\partial^+ a_3 = v_0$	$\partial^- a_3 = v_3$
$\partial^+ a_4 = v_1$	$\partial^- a_4 = v_2$	$\partial^+ a_5 = v_1$	$\partial^- a_5 = v_4$
$\partial^+ a_6 = v_2$	$\partial^- a_6 = v_3$	$\partial^+ a_7 = v_3$	$\partial^- a_7 = v_4$
$\partial^+ a_8 = v_4$	$\partial^- a_8 = v_2$	$\partial^+ a_9 = v_4$	$\partial^- a_9 = v_5$
$\partial^+ a_{10} = v_5$	$\partial^- a_{10} = v_3$	$\partial^+ a_{11} = v_5$	$\partial^- a_{11} = v_5$



例: V_0 を始点とする道

$L \subseteq V$: 道が経由する頂点

道探索($v \in V, L$) {

forall($a \in \delta^+ v$) {

$w = \partial^- a$

if($w \notin L$) {

$L = L \cup \{w\}$

道探索(w, L)

$L = L \setminus \{w\}$

}

}

}



探索結果の木表記

