データ構造とアルゴリズム

07 有向グラフ

宮本 裕一郎

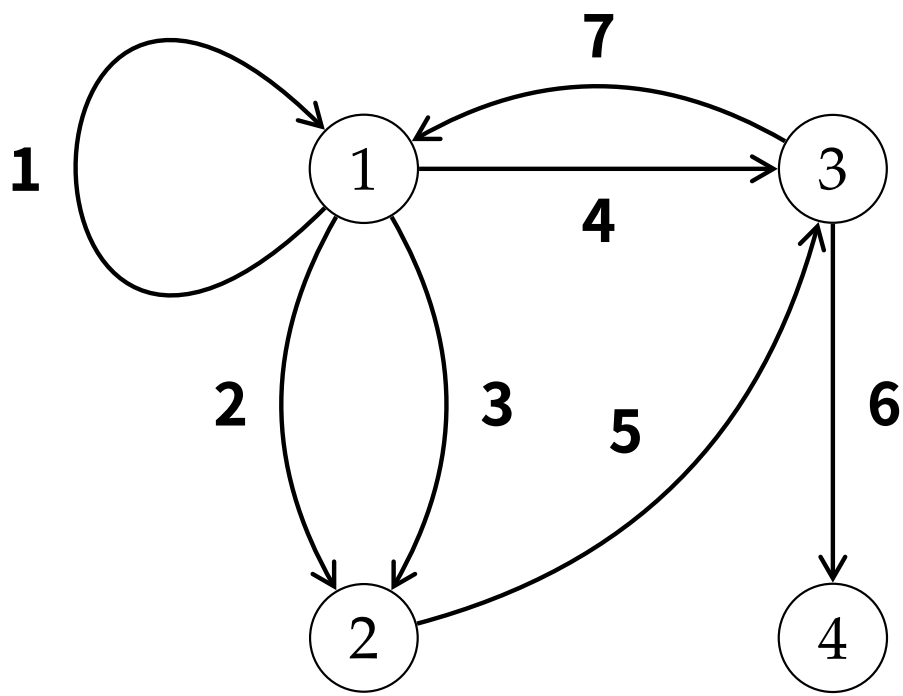
miyamoto **あっと** sophia.ac.jp

上智大学 理工学部 情報理工学科

目次

有向グラフ

枝に向きがあるグラフを有向グラフという．以下に有向グラフのイメージ図を示す．



有向グラフの定義

定義 (有向グラフ)

集合 V と集合 E と接続関数 $\Psi\colon E \to V \times V$ の 3 つ組 (V, E, Ψ) を有向グラフ（directed graph or digraph）という[1]．集合 V を頂点集合（vertices），集合 E を（有向）枝集合（(directed) edges）という．

- **有向グラフ G が与えられたとき，その頂点集合を $V(G)$，枝集合を $E(G)$ で表す．**
- **有向グラフ G において，枝 $e \in E(G)$ は $\Psi(e) = (v, w)$ であるとき頂点 v と w を結んで（join）いるという．このとき，v と w は有向グラフ G において隣接（adjacent）しているという．またこのとき，v と w は枝 e の端点（end point）であるという．**
- **有向グラフの枝には向きがあるので，枝 $e \in E(G)$ は $\Psi(e) = (v, w)$ であるとき v から出ている，そして w に入っているという．またこのとき，v を e の tail，w を e の head という．**

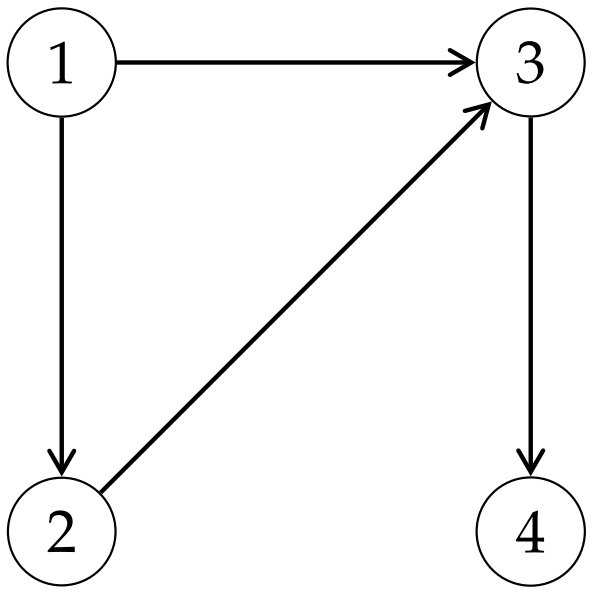
[1] ここでも，集合 V, E は有限集合とは限らない．しかし，アルゴリズムの文脈では多くの場合において有限集合である．

有向グラフの用語

- 有向グラフに対して，以前定義した「枝に向きがないグラフ」を**無向グラフ**（undirected graph）という．
- 有向枝 e の端点が同一であるとき，その枝を**自己ループ**（self loop **あるいは単に** loop）という．また，$e, e' \in E(G)$ が $e \neq e'$ であるにもかかわらず $\Psi(e) = \Psi(e')$ であるとき，e と e' は parallel であるという．
- Parallel な枝を含まない有向グラフを**単純有向グラフ**（simple directed graph **あるいは** simple digraph）という．

単純有向グラフの定義

例 (単純有向グラフ G の図)



- 単純有向グラフは簡便に以下のように定義できる．

定義 (単純有向グラフ)

単純有向グラフ G は頂点集合 V と枝集合 $E \subset \{(v,w) \mid v \in V,\ w \in V\}$ の 2 つからなる組 (V,E) である．

例 (単純有向グラフ G の式)

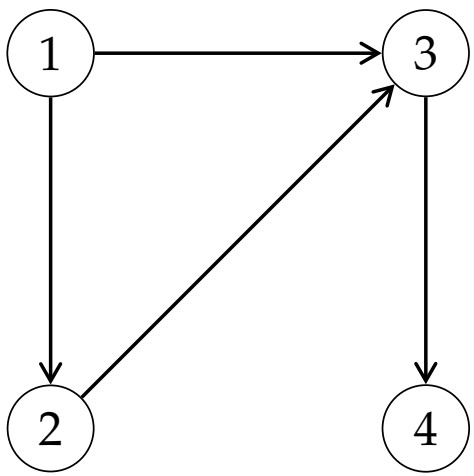
$V(G) = \{ ①, ②, ③, ④ \},\ E(G) = \{(①, ②),\ (①, ③),\ (②, ③),\ (③, ④)\}$

有向歩道

定義 (有向歩道)

有向グラフ G が与えられているとする．その頂点 $v_i \in V(G)$ と枝 $e_j \in E(G)$ の列 $(v_1, e_1, v_2, \ldots, v_k, e_k, v_{k+1})$ は，すべての $i \in \{1, \ldots, k\}$ に関して $\Psi(e_i) = (v_i, v_{i+1})$ を満たすとき（あるいは G が単純有向グラフで $e_i = (v_i, v_{i+1})$ を満たすとき），有向グラフ G 上の有向歩道（directed walk）であるという．

例 (有向グラフ G 上の歩道)



- $V(G) = \{ ①, ②, ③, ④ \}$,
  $E(G) = \{(①, ②),\ (①, ③),\ (②, ③),\ (③, ④)\}$
  とする．
- $(①,\ (①, ②),\ ②,\ (②, ③),\ ③)$ **は有向グラフ G 上の有向歩道である．**
- $(①,\ (①, ③),\ ③,\ (②, ③),\ ②)$ **は有向グラフ G 上の有向歩道ではない．**

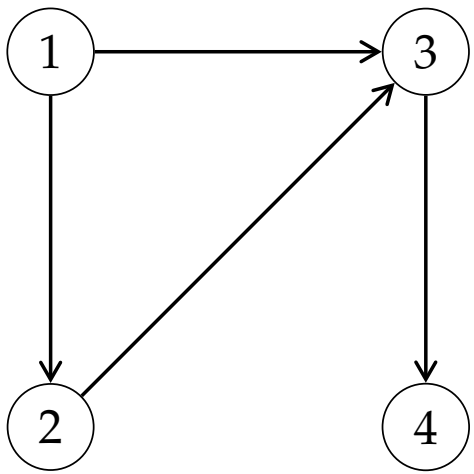
有向歩道の長さ

有向歩道に含まれる枝の数を有向歩道の長さという．正確な定義は以下の通りである．

定義 (有向歩道の長さ)

有向グラフ G 上の有向歩道 $(v_1, e_1, v_2, \ldots, v_k, e_k, v_{k+1})$ の長さは k である．ここで $v_i \in V(G)$ $(i \in \{1, \ldots, k+1\})$，$e_i \in E(G)$ $(i \in \{1, \ldots, k\})$ である．

例 (有向グラフ G 上の歩道の長さ)



有向グラフ G 上の有向歩道
$(①, (①,②), ②, (②,③), ③, (③,④), ④)$
の長さは 3 である．

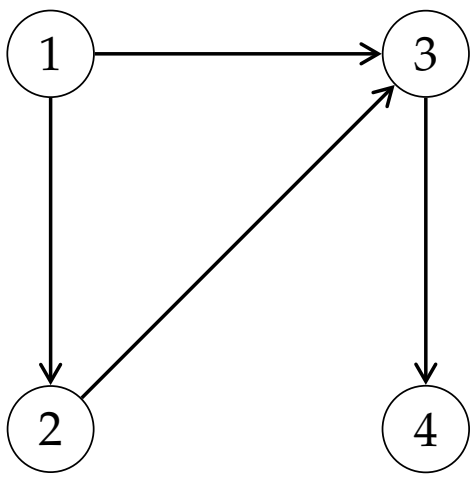
グラフ上の距離

与えられた有向グラフ上の 2 頂点間の有向歩道の長さの最小値を，その 2 頂点間の距離という．正確な定義は以下の通りである．

定義 (グラフ上の距離)

有向グラフ G において，頂点 $v \in V(G)$ から頂点 $w \in V(G)$ への有向歩道の長さの最小値を頂点 v から頂点 w への**距離（distance）**という．頂点 $v \in V(G)$ から頂点 $w \in V(G)$ への有向歩道が存在しないときには，便宜的に，頂点 v から頂点 w への距離は ∞ とする．

例 (有向グラフ G 上の距離)



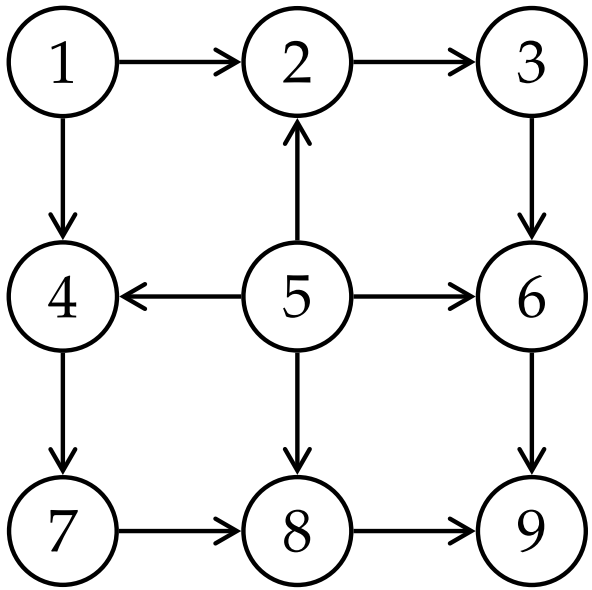
有向グラフ G における，

- 頂点①から頂点②への距離は 1，
- 頂点①から頂点③への距離は 1，
- 頂点①から頂点④への距離は 2，
- 頂点④から頂点③への距離は ∞，

である．

有向グラフ上の最短路問題

例 (入力)



$s = ①$

問題 (有向グラフ上の最短路)

入力 有向グラフ G, 始点 $s \in V(G)$

出力 始点から各頂点への距離
$d\colon V(G) \to \mathbb{Z}_{\geq 0} \cup \{\infty\}$

例 (出力)

$d(①) = 0,\ d(②) = d(④) = 1,$
$d(③) = d(⑦) = 2,$
$d(⑥) = d(⑧) = 3,\ d(⑨) = 4,$
$d(⑤) = \infty$

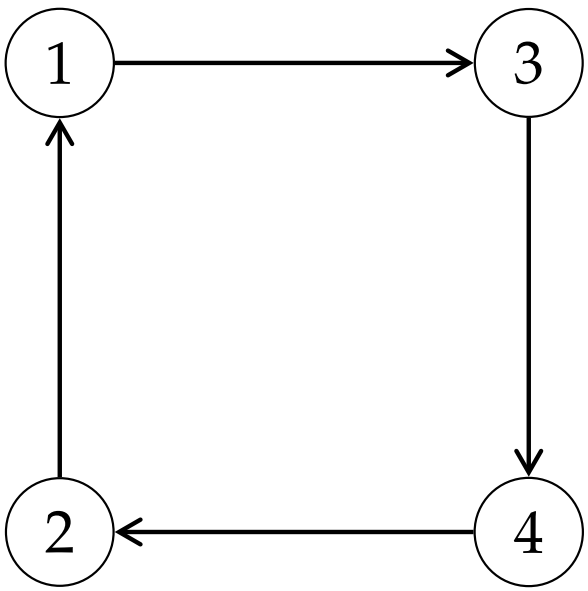
- 有向グラフ上の最短路問題は，基本的には，幅優先探索で解ける．(→ 演習問題)

閉路

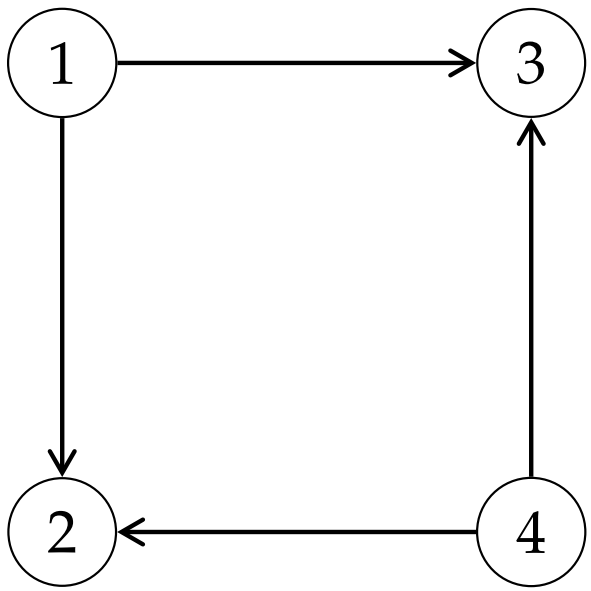
定義 (閉路)

有向グラフ C の頂点集合 $V(C)$ を $\{v_1, \ldots, v_n\}$，枝集合を $\{e_1, \ldots, e_n\}$ とする．$(v_1, e_1, v_2, \ldots, v_n, e_n, v_1)$ が有向歩道となっているならば，グラフ C を有向閉路（directed cycle or directed circuit）という．

例 (有向閉路 C)



例 (有向閉路でないグラフ G)

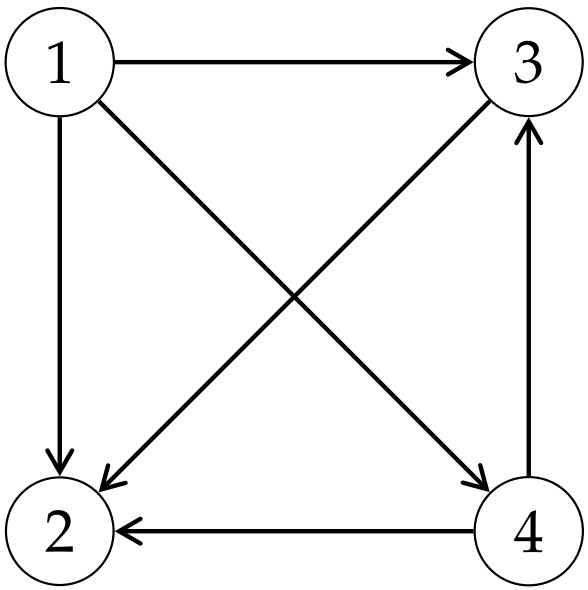


有向非閉路的グラフ

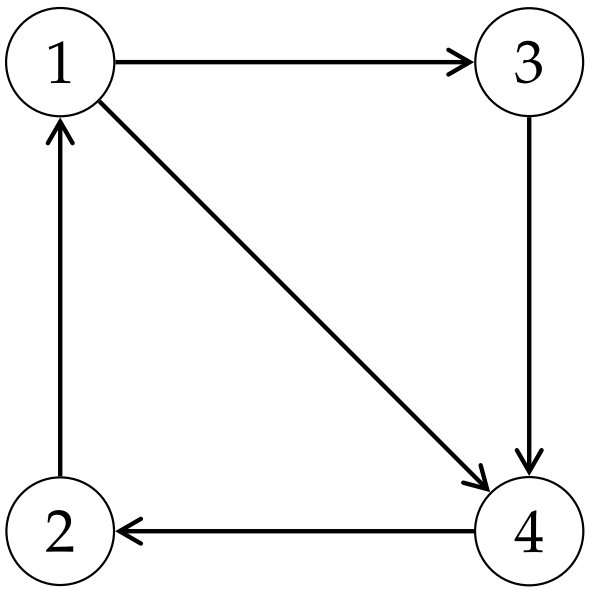
定義 (有向非閉路的グラフ)

有向閉路を部分グラフとして含まない有向グラフを有向非閉路的グラフ（directed acyclic graph 略して DAG）という．

例 (有向非閉路的グラフ G)



例 (有向非閉路的でないグラフ H)

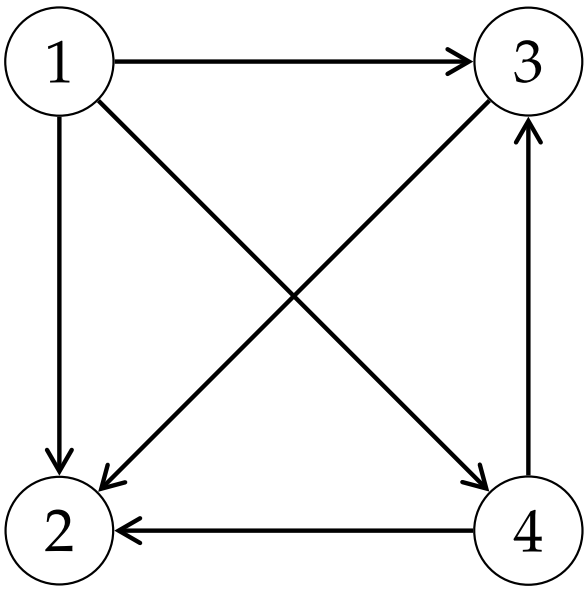


出次数と入次数

定義 (出次数と入次数)

頂点 v から出ている枝の集合を $\delta^+(v)$ で表し，その本数 $|\delta^+(v)|$ を出次数（out degree）という．同様に，頂点 v に入っている枝の集合を $\delta^-(v)$ で表し，その本数 $|\delta^-(v)|$ を入次数（in degree）という．

例 (グラフ G とその頂点の次数)



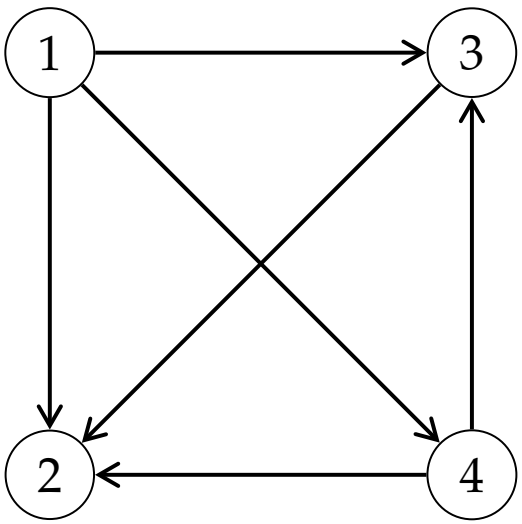
- $\delta^+(①) = \{(①,②),\ (①,③),\ (①,④)\},\ |\delta^+(①)| = 3$
- $\delta^-(①) = \{\},\ |\delta^-(①)| = 0$
- $\delta^+(②) = \{\},\ |\delta^+(②)| = 0$
- $\delta^-(②) = \{(①,②),\ (③,②),\ (④,②)\},\ |\delta^-(②)| = 3$
- $\delta^+(③) = \{(③,②)\},\ |\delta^+(③)| = 1$
- $\delta^-(③) = \{(①,③),\ (④,③)\},\ |\delta^-(①)| = 2$
- $\delta^+(④) = \{(④,②),\ (④,③)\},\ |\delta^+(①)| = 2$
- $\delta^-(④) = \{(①,④)\},\ |\delta^-(④)| = 1$

誘導部分グラフ

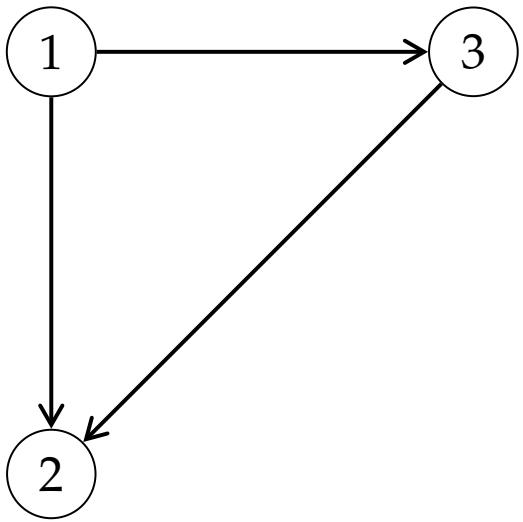
定義 (誘導部分グラフ)

有向グラフ G の部分グラフ H のうち，$E(H) = \{(v,w) \in E(G) \mid v,w \in V(H)\}$ を満たすものを誘導部分グラフ（induced subgraph）という．またこのとき，H は $V(H)$ によって誘導された部分グラフであるともいう．
グラフ G とその頂点部分集合 $S \subset V(G)$ が与えられたとき，S によって誘導される G の部分グラフを $G[S]$ で表す．

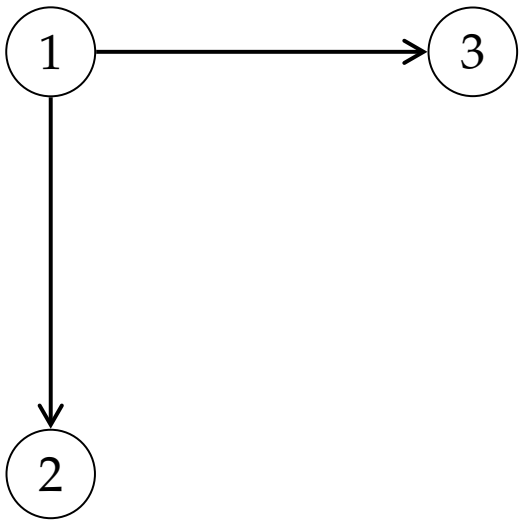
例 (グラフ G)



例 (G の誘導部分グラフ H)



例 (G の誘導部分グラフではない部分グラフ H')



Topological order

有向非閉路的グラフが与えられたとき，枝の向きが揃うような，頂点の並びを topological order **という．正確な定義は以下の通りである．**

定義 (Topological order)

有向非閉路的グラフ G **が与えられたとき，その頂点集合に対応する順序** $t\colon V(G) \to \{1,\dots,|V(G)|\}$ $(v \neq w \Longrightarrow t(v) \neq t(w))$ **が** $(v,w) \in E(G) \Longrightarrow t(v) < t(w)$ **を満たすならば，それを** topological order **という．**

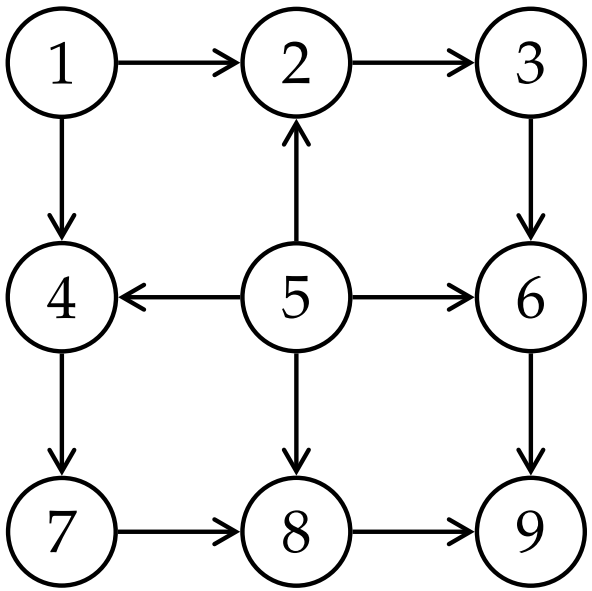
問題 (Topological order)

入力 **有向非閉路的グラフ** G

出力 *topological order* t

Topological order の例

例 (入力)



例 (出力)

$t(①) = 1,\ t(②) = 3,\ t(③) = 5,\ t(④) = 4,\ t(⑤) = 2,\ t(⑥) = 7,\ t(⑦) = 6,\ t(⑧) = 8,\ t(⑨) = 9$

- グラフの図的理解としては，頂点を直線上に並べたときに，枝の向きが揃う並べ方が Topological order である．
- 例えば，頂点が仕事であり，枝が仕事の依存関係（ある仕事を行うためにはその前に終わらせなければならない仕事があるなどの関係）を表しているならば，Topological order は仕事を行う順番となる．

Topological sort

- Topological order を出力するアルゴリズムを `TopologicalSort` として関数っぽく以下に記す．
- 参照，代入，比較，四則演算を基本演算としている．
- アルゴリズムの実行中に有向グラフ G の頂点が 1 つずつ減っていることに注意する．

`TopologicalSort`(G):

Step 1 それぞれの $i \in (1, 2, \ldots, |V(G)|)$ に関して，以下を繰り返す．

- Step 1-1 G の頂点のうち，入次数が 0 の頂点のうちの 1 つを v とする．
- Step 1-2 $t(v) = i$ とする．
- Step 1-3 G を $G[V(G) \setminus \{v\}]$ で置き換える．

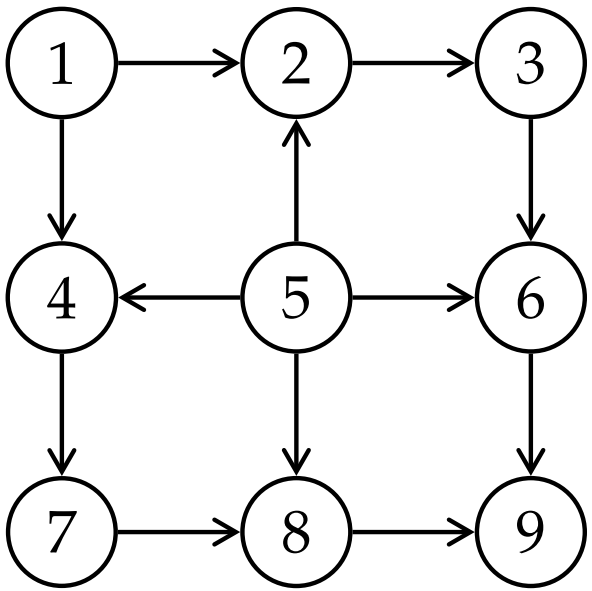
Step 2 t を出力して終了する．

その他の話題

- 強連結成分
  - すべての頂点の間に有向歩道がある有向グラフを**強連結グラフ**（strongly connected graph）という．
  - 有向グラフの極大な強連結部分グラフを**強連結成分**（strongly connected component(s)）という．
  - 参照，代入，比較，四則演算を基本演算とする場合，与えられた有向グラフ G の強連結成分をすべて出力するのは時間複雑度 $O(|V(G)| + |E(G)|)$ でできる．
- 平面グラフ
  - 枝を交差させずに平面上に描画できるグラフを**平面グラフ**（planar graph）という．これは，枝を交差させずに球面上に描画できることと同値である．
  - 平面グラフ G を平面（あるいは球面）上に描画した際の面の数は $|E(G)| - |V(G)| + 2$ である．これは Eular の多面体の公式として有名である．
- 道路網をグラフで表現したものや Social Network をグラフで表現したものの頂点数と枝数は線形比例の関係にあることが経験的に知られている．

枝が少ないグラフの頂点隣接行列

- 有向グラフも無向グラフと同様に頂点隣接行列で表現できる．例えば，下記の通りである．

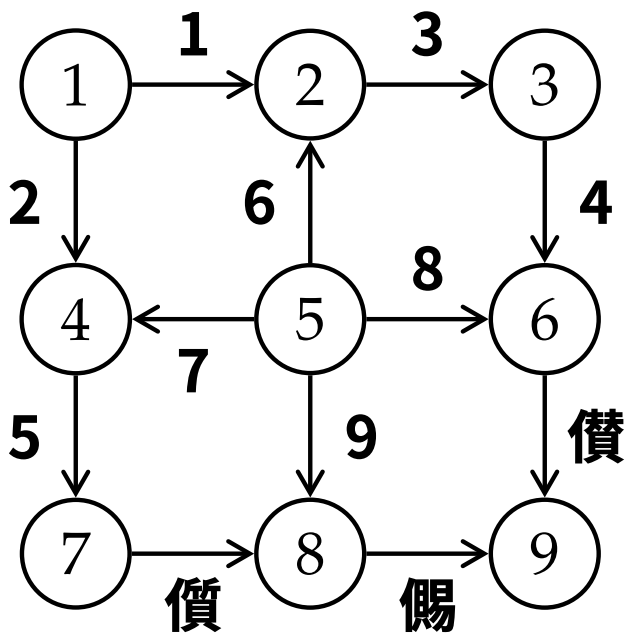


$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 &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

- 頂点隣接行列は，無向グラフにおいてもそうであるが，枝数が少ないグラフにおいては無駄が多いように感じる．

頂点隣接行列をリストっぽく表現

- イメージ的には，下記の右図のように，頂点ごとに隣接する頂点をリストで保存するほうが空間複雑度が小さくなりそうである．



①：[②, ④]
②：[③]
③：[⑥]
④：[⑦]
⑤：[②, ④, ⑥, ⑧]
⑥：[⑨]
⑦：[⑧]
⑧：[⑨]
⑨：[]

頂点隣接リスト

具体的には，以下の通り 2 つの列（あるいは配列）を用いて，頂点ごとの隣接する頂点を保存できる．これを**頂点隣接リスト**（adjacency list）という．

	①	②	③	④	⑤	⑥	⑦	⑧	⑨
p	1	3	4	5	6	僂	僭	傷	□

ただしここで□は枝の名前として決して出現しない記号とする．

	1	2	3	4	5	6	7	8	9	僂	僭	傷
h	②	④	③	⑥	⑦	②	④	⑥	⑧	⑨	⑧	⑨
n	2	□	□	□	□	7	8	9	□	□	□	□

①：[②, ④]
②：[③]
③：[⑥]
④：[⑦]
⑤：[②, ④, ⑥, ⑧]
⑥：[⑨]
⑦：[⑧]
⑧：[⑨]
⑨：[]

頂点 v が指定されたとき，その隣接する頂点をすべて出力するアルゴリズム `Neighbors` を以下に記す．

`Neighbors`(v):

Step 1　N を空集合とする．

Step 2　もし $p(v)$ が□ならば，N を出力して終了する．

Step 3　もし $p(v)$ が□でないならば，e に $p(v)$ を代入し，以下を繰り返す．

- Step 3-1　N に $h(e)$ を加える．
- Step 3-2　もし $n(e)$ が $-$ ならば，N を出力して終了する．
- Step 3-3　もし $n(e)$ が $-$ でないならば，e に $n(e)$ を代入する．

頂点隣接リストのまとめ

- 頂点隣接リストを利用すると，頂点数 n，枝数 m のグラフは空間複雑度 $\mathrm{O}(n+m)$ で保存できる．
  - 頂点隣接行列を利用すると，枝数とは関係なく空間複雑度 $\mathrm{O}(n^2)$ となる．
- 頂点隣接リストを利用すると，指定された頂点の隣接頂点は，1 つあたり時間複雑度 $\mathrm{O}(1)$ で列挙できる．
  - 頂点隣接行列を利用すると，隣接する頂点数とは関係なく合計で時間複雑度 $\mathrm{O}(n)$ となる．

なお，ここでは触れなかったが，十分に長い配列を用意してあるならば，頂点の追加，枝の追加はともに時間複雑度 $\mathrm{O}(1)$ でできる．

より一般に

- より一般に，頂点隣接リストにおいて各頂点の隣接頂点を保存したようなデータ構造を**リスト（list）**という．
- リストを利用すると，可変長列，あるいは長さの揃っていない列を効率的な空間複雑度で保存できる場合が多い．

有向グラフ上の最短路問題

問題 有向グラフ上の最短路問題を解くアルゴリズムを関数 ShortestPath として記せ．そのアルゴリズム ShortestPath の時間複雑度を頂点数 n の関数で表わせ．

解答例 有向グラフ上の最短路問題を解く（始点からの最短路長 d を出力する）アルゴリズムを ShortestPath として関数っぽく以下に記す．なお，幅優先探索との差分を赤く記す．

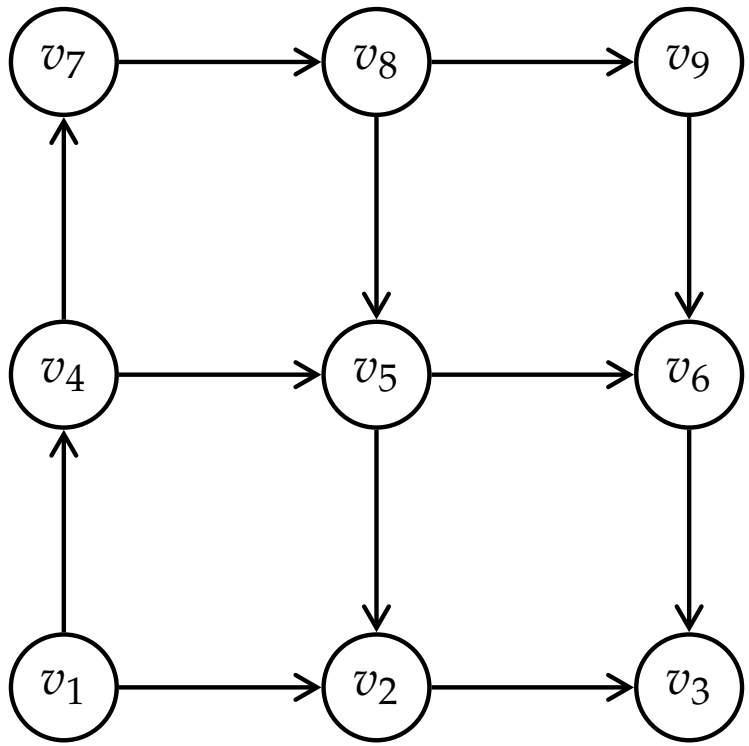
ShortestPath(G, s):

Step 1 頂点部分集合 C を $\{s\}$ とする．キュー Q を (s) とする．
Step 2 $d(s) = 0$ とする．
Step 3 キュー Q が空集合でない間，以下を繰り返す．
　Step 3-1 キュー Q から要素を 1 つ取り出し，それを v とする．
　Step 3-2 v の隣接頂点 w それぞれに関して，もし $w \notin C$ ならば，頂点部分集合 C とキュー Q に w を加え，$d(w) = d(v) + 1$ とする．
Step 4 d を出力して終了する．

- ShortestPath の時間複雑度は $O(n^2)$ である．Step 3 は $O(n)$ 回繰り返され，Step 3-2 の「すべての隣接頂点に関して……」が $O(n)$ だからである．

深さ優先探索

有向グラフ G と始点 $s \in V(G)$ を入力として，集合
$\{v \in V(G) \mid s$ から v への有向歩道がある $\}$ を出力する問題を考える．有向グラフ G が以下のグラフであり，始点 $s = v_4$ であり，深さ優先探索で問題の解を出力することを考える．このとき，それぞれの頂点が解に加えられる順番を以下のグラフの頂点の左上に記入せよ．ただし，解に加えられない頂点に関しては空欄のままで良い．なお，正解は唯一ではない．



（2015 年度期末試験問題より）

連結な有向グラフ G と始点 $s \in V(G)$ を入力とし，始点から各頂点への距離 $d\colon V(G) \to \mathbb{Z}_{\geq 0}$ を出力する問題を「連結有向グラフ上の最短路問題」と定義する．連結有向グラフ上の最短路問題を解くアルゴリズムを記せ．参考までに，スタックを用いた深さ優先探索を `StackDFS` として以下に記す．

`StackDFS`(G, s)

Step 1 C を $\{s\}$ とする．スタック S を (s) とする．

Step 2 スタック S が空集合でない限り，以下の Step 2-1 と Step 2-2 を繰り返す．

- Step 2-1 スタック S から要素を 1 つ取り出し，それを v とする．
- Step 2-2 v の隣接頂点 $w \in V(G)$ それぞれに関して，$w \notin C$ ならば，C とスタック S に w を加える．

Step 3 C を出力して終了する．

（2016 年度期末試験問題より）

頂点集合 $\{1,2,3,4,5,6,7,8\}$**，**
有向枝集合 $\{(2,8),\ (3,4),\ (4,1),\ (4,5),\ (6,8),\ (7,4),\ (8,3),\ (8,7)\}$ **からなる有向非閉路的グラフ（**Directed Acyclic Graph**）の頂点集合を** topological order **で並べた列は，例えば**

( ) **である．**

また，このグラフの頂点集合を topological order **で並べる並べ方は** **通りある．なお，参考までに，頂点集合** $\{1,2,3\}$**，有向枝集合** $\{(1,3),\ (2,1)\}$ **からなる有向非閉路的グラフの頂点集合を** topological order **で並べた列は** $(2,1,3)$ **である．**

（2016 **年度期末試験問題より）**

さらなる勉強のために

- **有向グラフの定義や用語は** [Korte and Vygen, 2012] **を参考にした．**

参考文献

[Korte and Vygen, 2012] Korte, B. and Vygen, J. (2012).
Combinatorial Optimization: Theory and Algorithms, volume 21 of *Algorithms and Combinatorics*.
Springer-Verlag, 5th edition.