アルゴリズムとデータ構造

第12回　グラフの探索（2）

今日の内容

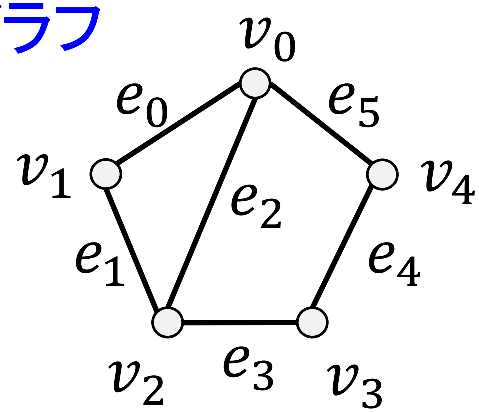
- 頂点の次数
- さまざまな グラフ
  - 完全グラフ
  - 路 (パス、道)、有向路
  - 閉路 (サイクル)、有向閉路
  - 二部グラフ
  - 完全二部グラフ
- 連結成分、連結成分分解
- 強連結成分、強連結成分分解

グラフ

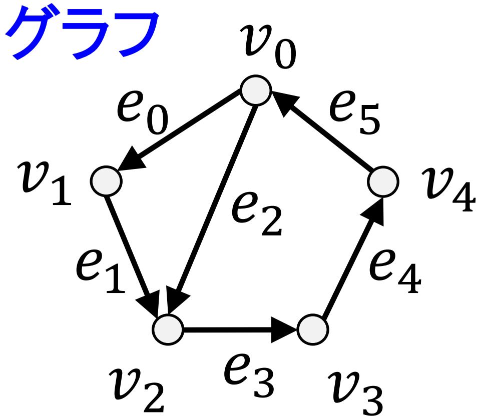
前回の復習

頂点(vertex) の集合 V と **辺** (edge) の集合$E \subseteq V \times V$ の **組** $(\boldsymbol{V}, \boldsymbol{E})$

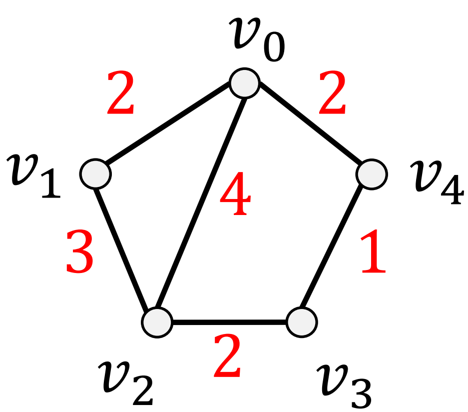
無向グラフ



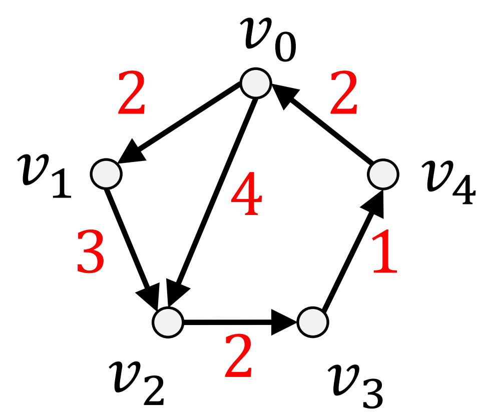
有向グラフ



無向ネットワーク
（重み付き無向グラフ）



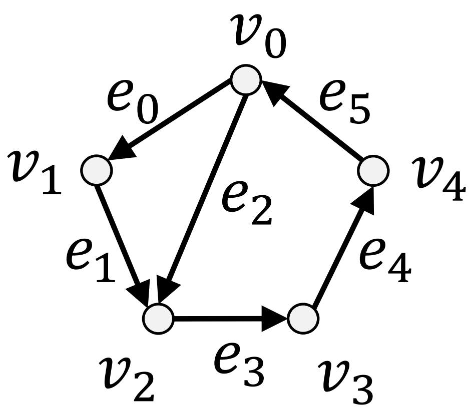
有向ネットワーク
（重み付き有向グラフ）



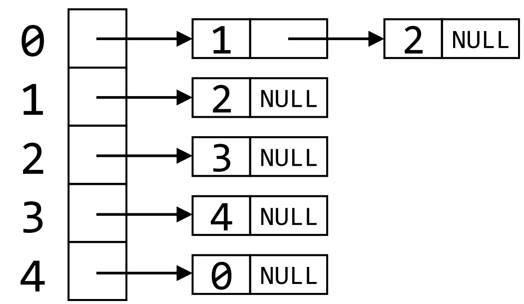
グラフの表現方法

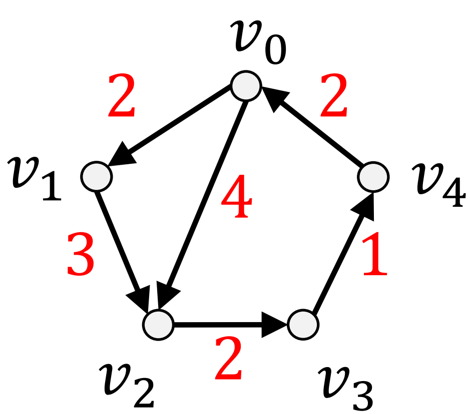
前回の復習

隣接行列　　隣接リスト

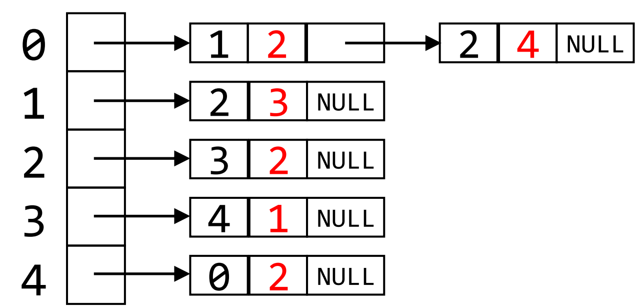


$$\begin{array}{cc} & \begin{array}{ccccc} v_0 & v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{array}{c} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} & \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$





$$\begin{array}{cc} & \begin{array}{ccccc} v_0 & v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{array}{c} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} & \begin{bmatrix} 0 & 2 & 4 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 2 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

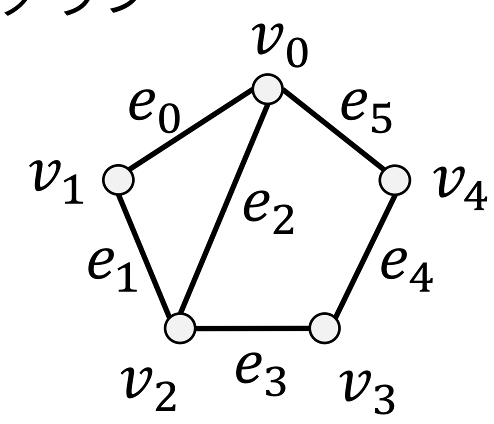


※ 無向グラフの場合は，各辺を両方向の2辺からなる有向グラフとみなして表現する

無向グラフの頂点の次数 (degree)

- 無向グラフ $G = (V, E)$、頂点 $v \in V$
- v の**次数**: v に接続する辺の本数
  - $\deg_G(v)$ や $\deg(v)$ と表す

無向グラフ



$$\deg_G(v_0) = 3$$
$$\deg_G(v_1) = 2$$
$$\deg_G(v_2) = 3$$
$$\deg_G(v_3) = 2$$
$$\deg_G(v_4) = 2$$

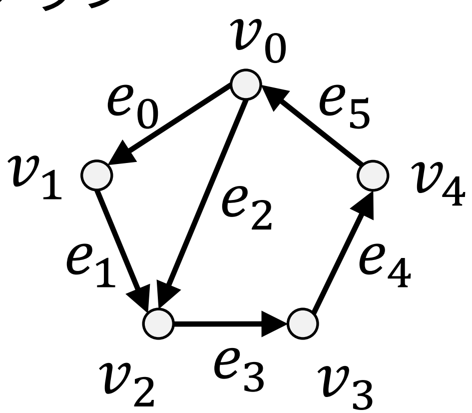
有向グラフの頂点の入次数、出次数 (indegree, outdegree)

- 有向グラフ $G=(V,A)$、頂点 $v \in V$
- v の**入次数**: v を終点とする有向辺の本数
  - $\deg_G^-(v)$ や $\deg^-(v)$ と表す
- v の**出次数**: v を始点とする有向辺の本数
  - $\deg_G^+(v)$ や $\deg^+(v)$ と表す

v に入ってくる有向辺の本数

v から出ていく有向辺の本数

有向グラフ



$\deg_G^-(v_0) = 1$ $\deg_G^+(v_0) = 2$

$\deg_G^-(v_1) = 1$ $\deg_G^+(v_1) = 1$

$\deg_G^-(v_2) = 2$ $\deg_G^+(v_2) = 1$

$\deg_G^-(v_3) = 1$ $\deg_G^+(v_3) = 1$

$\deg_G^-(v_4) = 1$ $\deg_G^+(v_4) = 1$

豆知識: 有向グラフの辺集合は A と表されることが多い

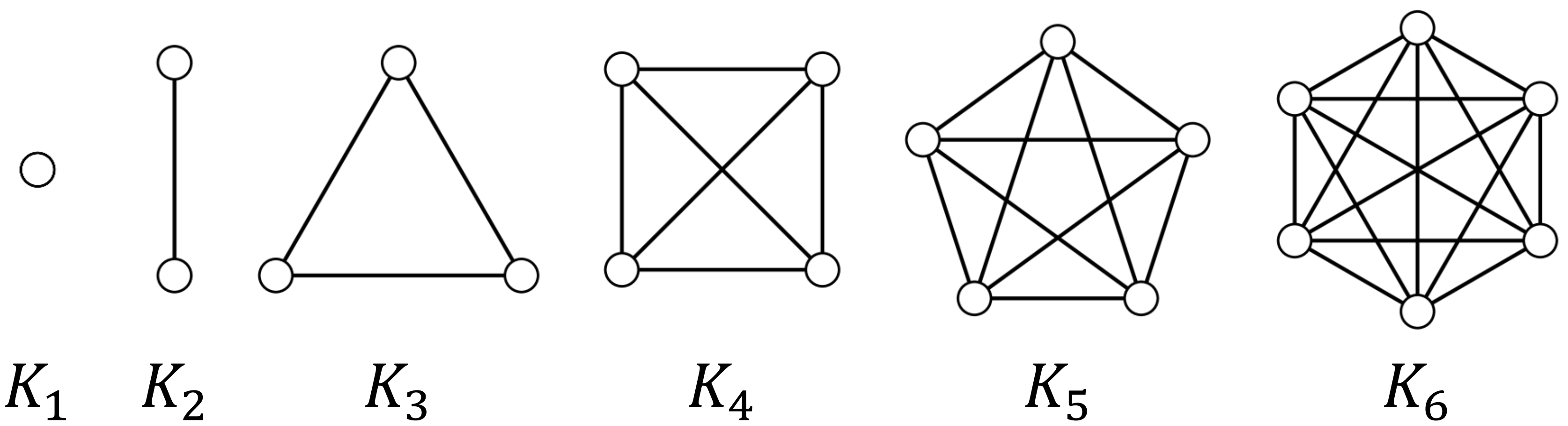
さまざまな グラフ

- 完全グラフ
- 路 (パス、道)、有向路
- 閉路 (サイクル)、有向閉路
- 二部グラフ
- 完全二部グラフ

完全グラフ (complete graph)

- 無向グラフ $G = (V, E)$
- 定義: V の任意の 2 頂点が辺で結ばれている

- K_n: 頂点数 n の完全グラフ $(n \geq 1)$

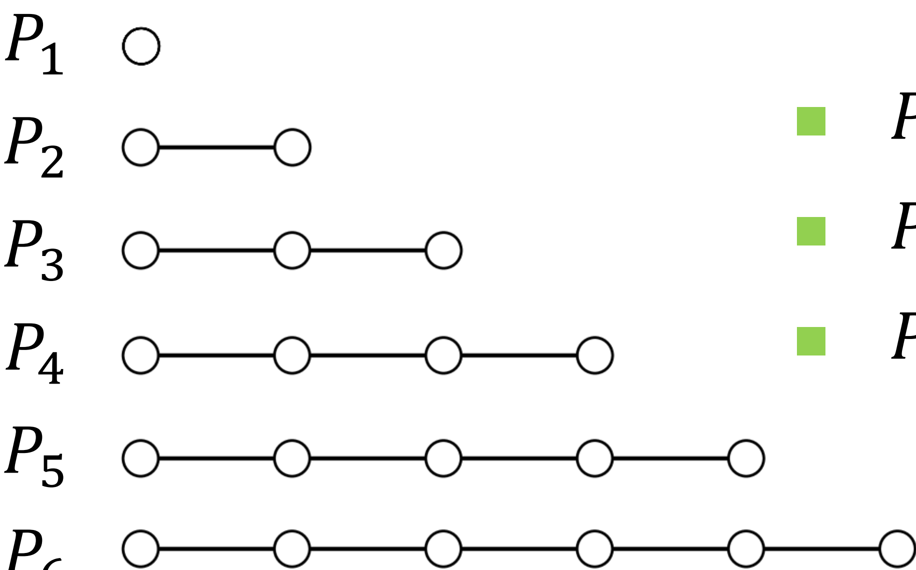


豆知識: 有向グラフでも同様に定義できるが、その場合には「有向」を付け、有向完全グラフと呼ぶことが多い

路 (パス、道) (path)

- 無向グラフ $G = (V, E)$
- 定義：$V = \{v_1, v_2, \dots, v_n\}$ として、E は
  $i = 1, 2, \dots, n-1$ の 辺 (v_i, v_{i+1}) のみからなる

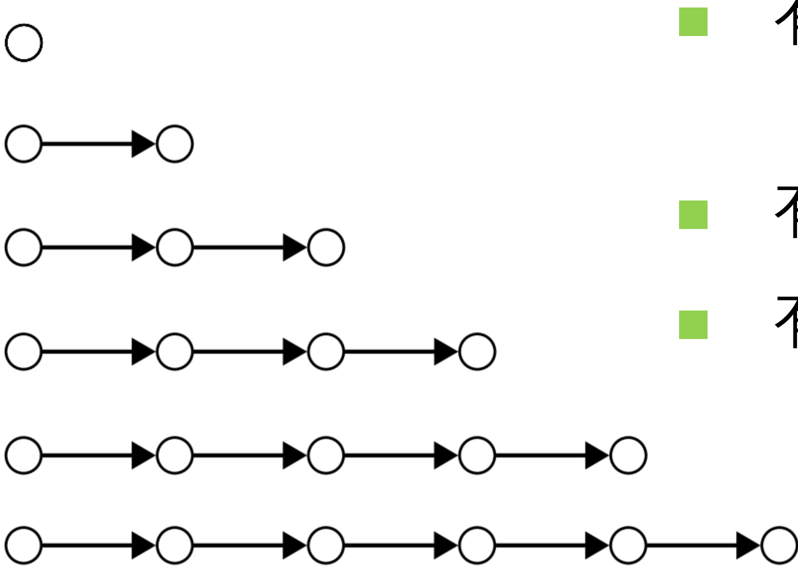
- P_n： 頂点数 n のパス $(n \geq 1)$



- P_n の端点：次数1の頂点
- P_n の長さ：辺の本数 $n-1$
- P_n は、端点と端点を結ぶ
  長さ $n-1$ のパス

有向路 (有向パス、有向道) (directed path)

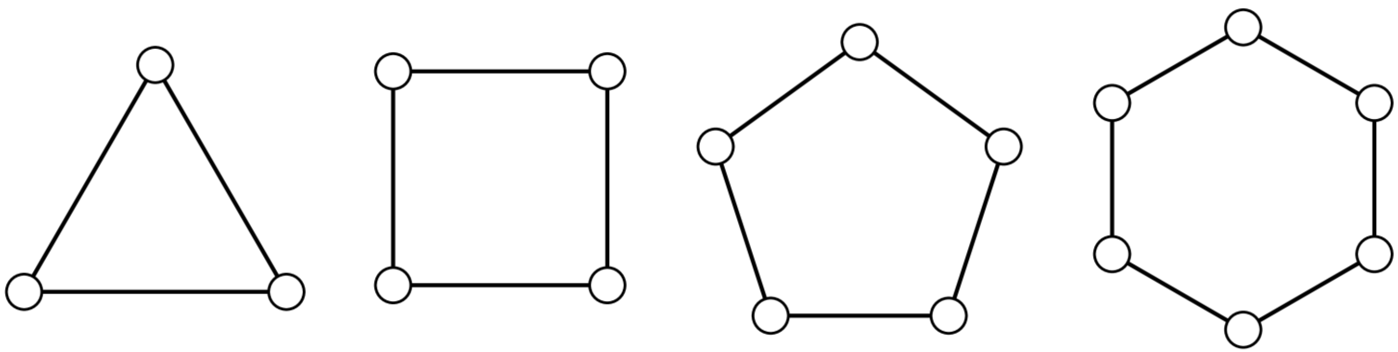
- 有向グラフ $G = (V, A)$
- 定義: $V = \{ v_1, v_2, \ldots, v_n \}$ として、E は
  $i = 1, 2, \ldots, n-1$ の 有向辺 (v_i, v_{i+1}) のみからなる



- 有向路の端点:
  入次数 or 出次数が1の頂点
- 有向路の長さ: 辺の本数
- 有向路は、入次数1の端点から
  出次数1の端点へ結ぶ

閉路 (サイクル) (cycle)

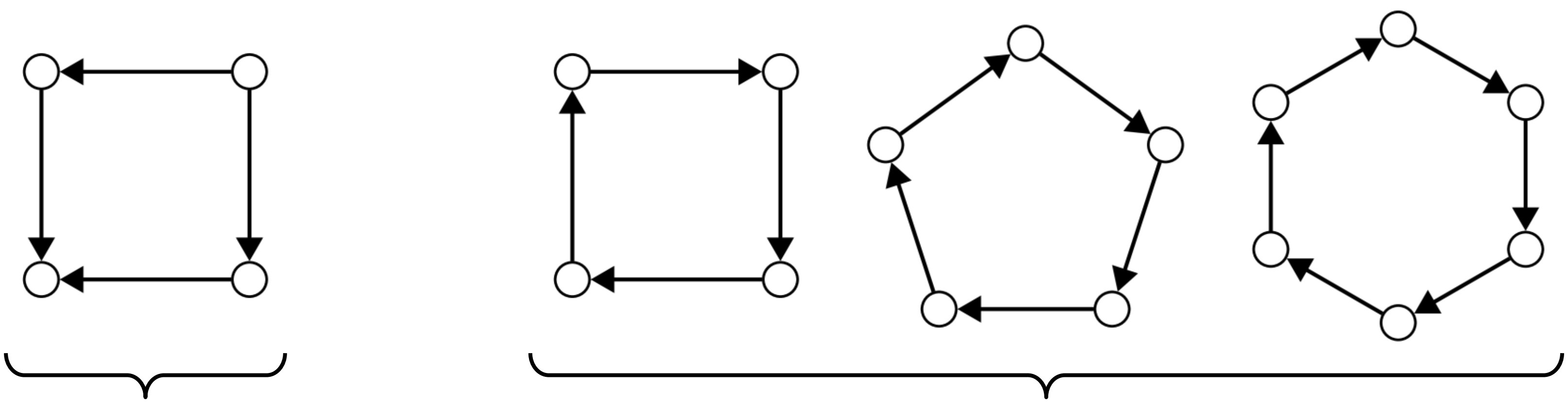
- 無向グラフ $G = (V, E)$
- 定義: $V = \{ v_1, v_2, \ldots, v_n \}$ として、E は辺 (v_n, v_1) と $i = 1, 2, \ldots, n-1$ の 辺 (v_i, v_{i+1}) のみからなる
- C_n: 頂点数 n の閉路 $(n \geq 3)$



- C_n の長さ: 辺の本数

有向閉路 (有向サイクル) (directed cycle)

- 有向グラフ $G = (V, E)$
- 定義: $V = \{ v_1, v_2, \ldots, v_n \}$ として、E は
  有向辺 (v_n, v_1) と
  $i = 1, 2, \ldots, n-1$ の 有向辺 (v_i, v_{i+1}) のみからなる

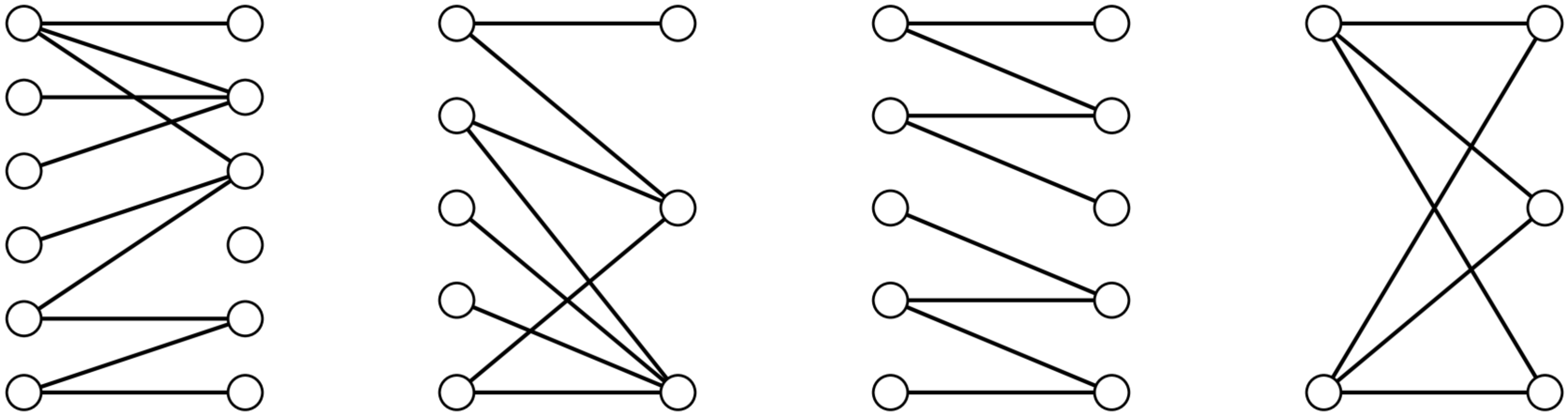


有向閉路ではない

有向閉路

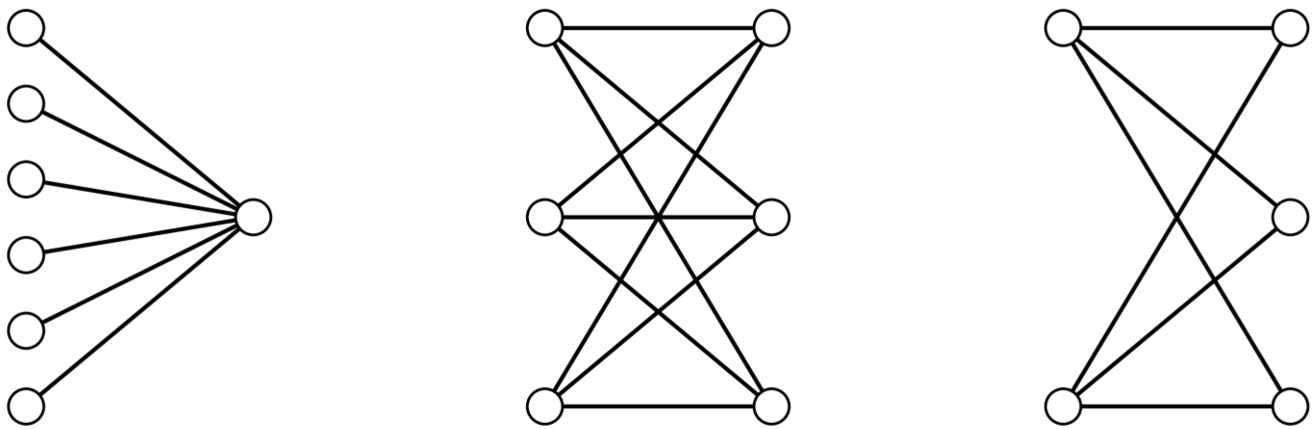
二部グラフ (bipartite graph)

- 無向グラフ $G = (V, E)$
- 定義：$V = V_1 \cup V_2$ かつ $E \subseteq V_1 \times V_2$
  つまり、V を V_1 と V_2 の2つに分割でき、
  どの辺も V_1 の頂点と V_2 の頂点を結ぶ



完全二部グラフ (complete bipartite graph)

- 無向グラフ $G = (V, E)$
- 定義：$V = V_1 \cup V_2$ かつ $E = V_1 \times V_2$
  つまり、V を V_1 と V_2 の2つに分割でき、
  V_1 の任意の頂点と V_2 の任意の頂点を結ぶ辺からなる
- $K_{m,n}$：$|V_1| = m, |V_2| = n$ の完全グラフ $(m, n \geq 1)$



$K_{6,1}$ $K_{3,3}$ $K_{2,3}$

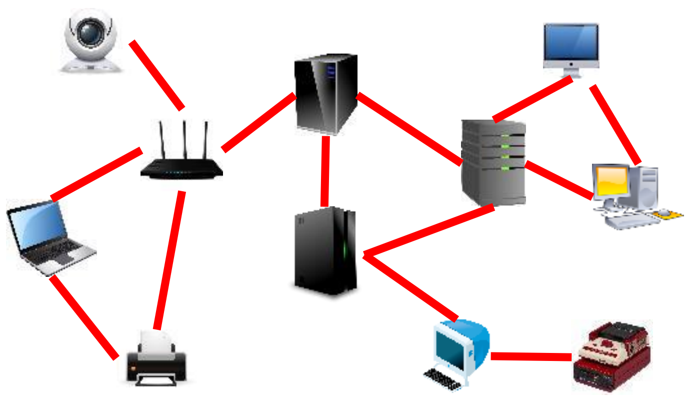
休憩

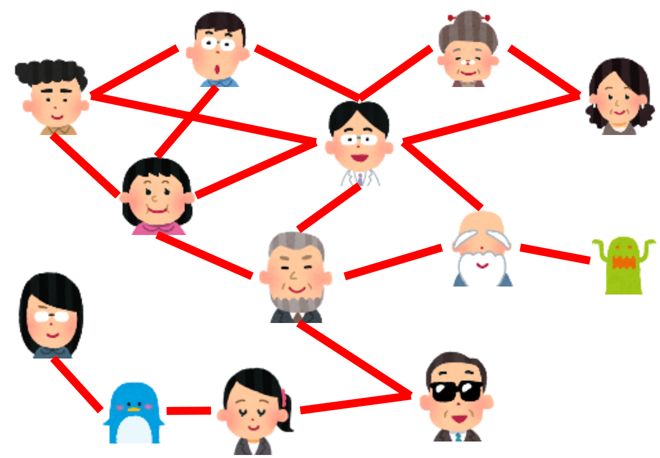
- ここで、少し休憩しましょう。
- 深呼吸したり、肩の力を抜いてから、
次のビデオに進んでください。

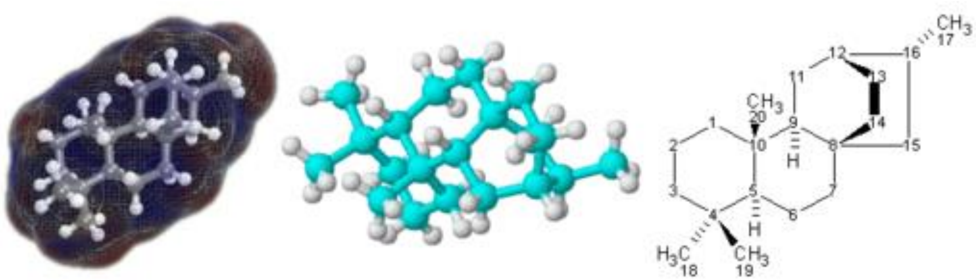
なんでグラフ？

前回の復習

モノとモノのつながりを簡潔に記述し，解析できる！





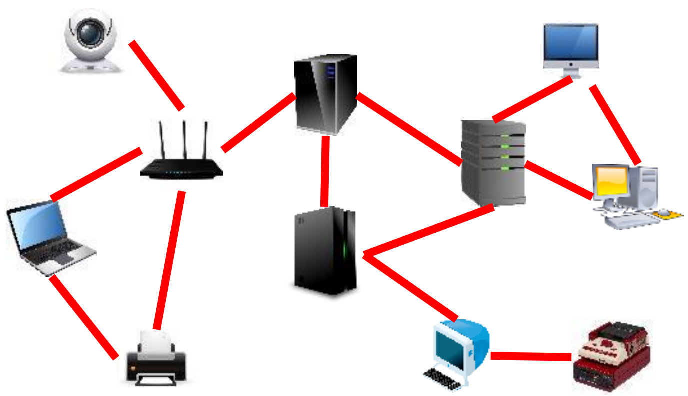


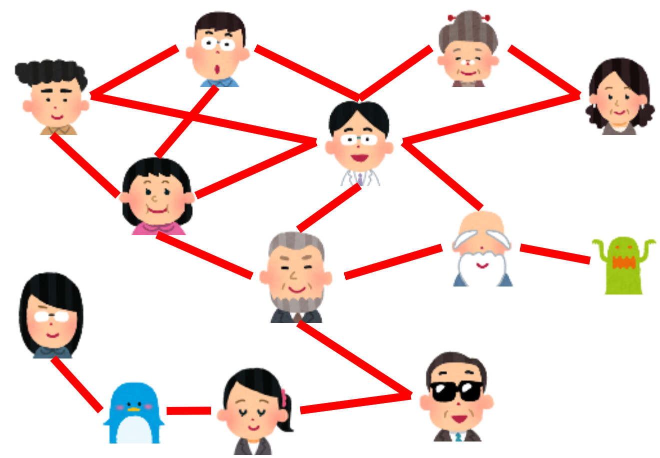
世の中のすべてはグラフ！

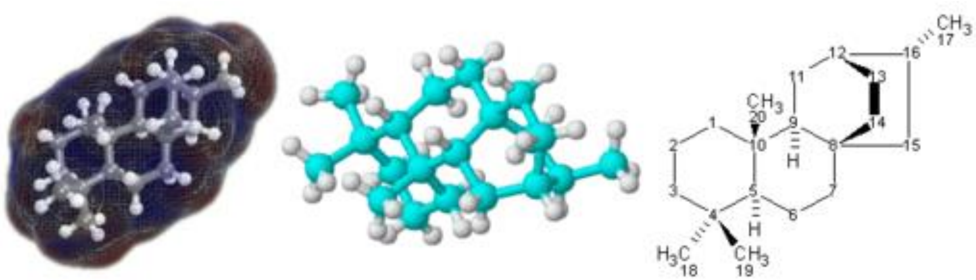
なんでグラフ？

有向グラフや無向グラフの
「つながり」って、何でしょうか？

モノとモノのつながりを簡潔に記述し，解析できる！



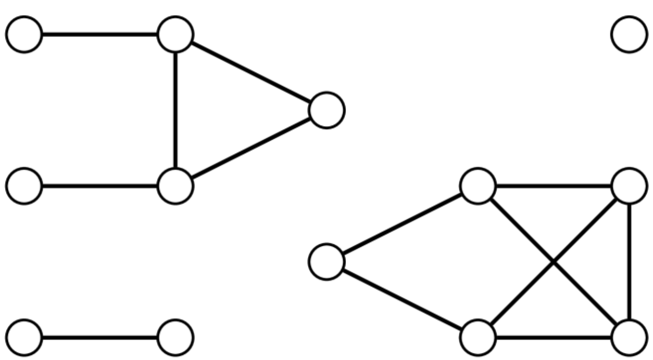




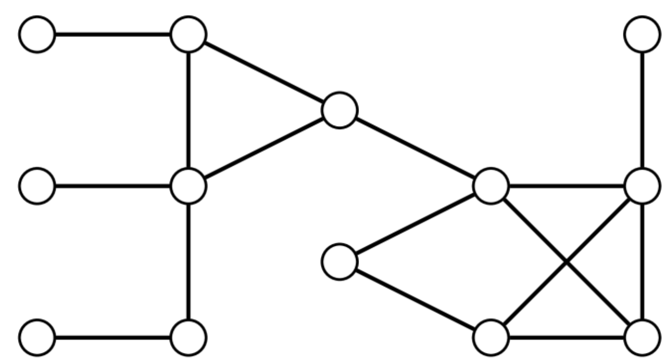
世の中のすべてはグラフ！

連結 / 非連結 (connected/disconnected)

- 無向グラフ $G = (V, E)$
- 定義：G が**連結**である
  $\Leftrightarrow$ V の任意の 2 つの頂点の間にパスが存在する
- G が連結でないとき、G は**非連結**であるという



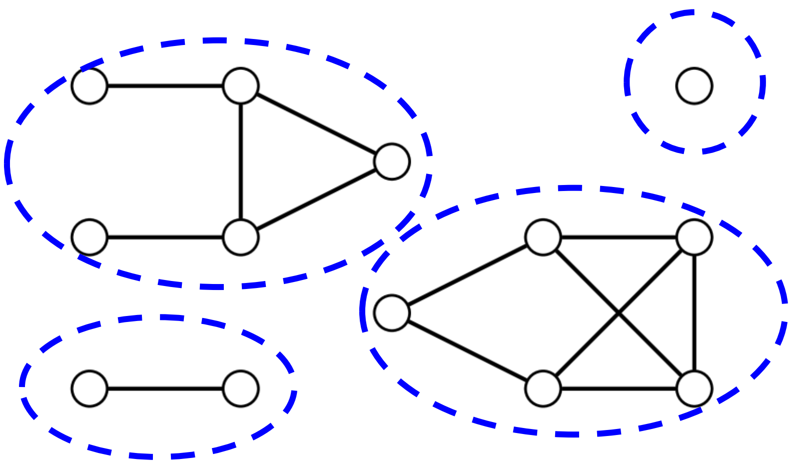
非連結



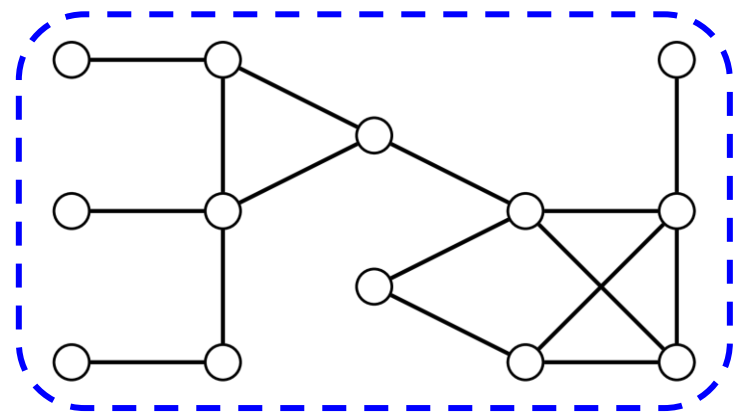
連結

連結成分 (connected component)

- 無向グラフ $G = (V, E)$
- G を**連結成分** $G_1 = (V_1, E_1), \ldots, G_k = (V_k, E_k)$ に分割する
  - 2 つの頂点 v_i, v_j が同じ連結成分に属す
    $\Leftrightarrow$ v_i, v_j を結ぶパスが存在



連結成分は 4 個

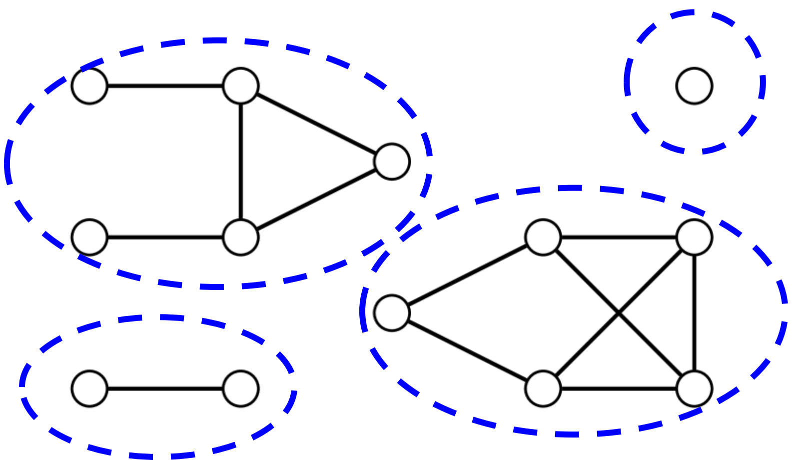


連結成分は 1 個

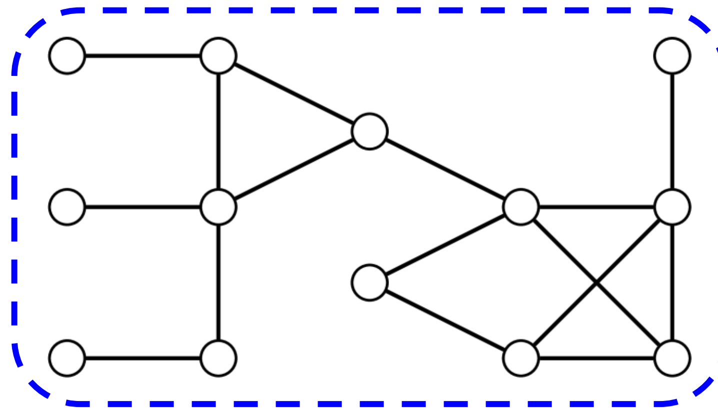
連結成分 (connecte

「頂点 v が属す連結成分」
= v から辺をたどって到達できる範囲
これは、v から深さ優先探索をすれば分かる

- 無向グラフ $G = (V, E)$
- G を**連結成分** $G_1 = (V_1, E_1), \ldots, G_k = (V_k, E_k)$ に分割する
  - 2つの頂点 v_i, v_j が同じ連結成分に属す
    $\Leftrightarrow$ v_i, v_j を結ぶパスが存在



連結成分は 4 個

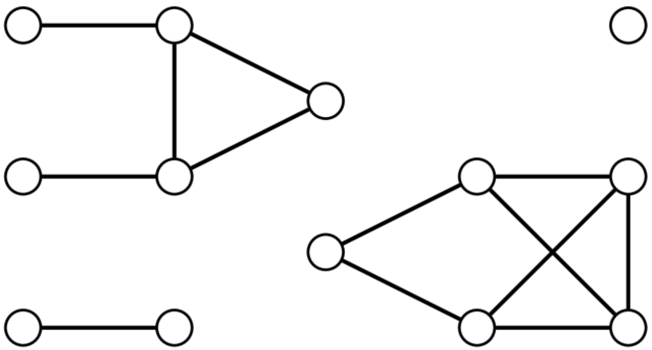


連結成分は 1 個

注意：頂点1つでも、連結成分

連結成分分解

「頂点 v が属す連結成分」
　= v から辺をたどって到達できる範囲
これは、v から深さ優先探索をすれば分かる



もう少し詳しく

1. 最初に、すべての頂点を “未訪問” にする
2. 未訪問の頂点 v を pick up して、
   v から深さ優先探索をする
   （この時、訪問した頂点には同じ ID を振る）

前回の復習
（DFS）

```
Procedure Visit(v: 頂点)
  状態[v] ← 訪問済;
  for each (vの隣接頂点u) do
      if (状態[u] = 未訪問) then
          Visit(u);      //再帰呼び出し
      end if
  end for
```

注意：
無向辺 (u, v) を、2つの有向辺 (u, v) と (v, u) とみなして実行する

連結成分分解

1. 最初に、すべての頂点を "未訪問" にする
2. 未訪問の頂点 v を pick up して、
   v から深さ優先探索をする
   （この時、訪問した頂点には同じ ID を振る）

Procedure CC (G: グラフ)
for each ($v \in V$) do 状態[v] ← 未訪問;
c ← 1; // ID として色 c を使う
for each ($v \in V$) do
    if (状態[v] = 未訪問) then
        Visit(v, c); // v から訪問できる全頂点を色 c で塗る
        $c \leftarrow c + 1$; // 次の連結成分は別の色 $c + 1$ にする
    end if
end for

Procedure Visit(v: 頂点, c: 色)
状態[v] ← c;
for each (vの隣接頂点u) do
    if (状態[u] = 未訪問) then
        Visit(u, c); //再帰呼び出し
    end if
end for

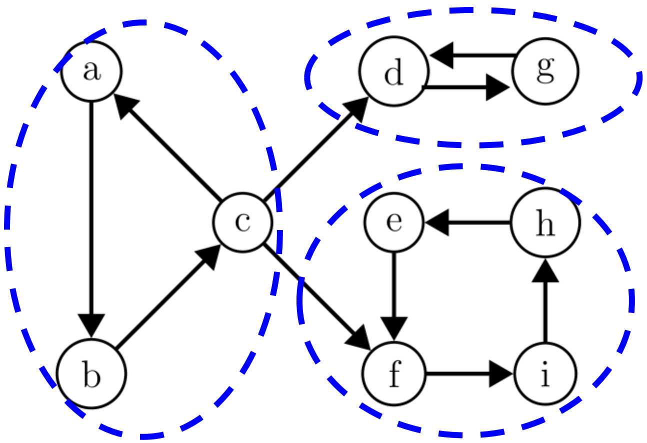
ID として
色 c を使う

強連結成分 (strongly connected component)

- 有向グラフ $G = (V, A)$
- G を**強連結成分** $G_1 = (V_1, A_1), \ldots, G_k = (V_k, A_k)$ に分割
  - 2つの頂点 v_i, v_j が同じ強連結成分に属す
    $\Leftrightarrow$ v_i から v_j への有向パスが存在
    $\wedge$ v_j から v_i への有向パスが存在

v_i から v_j へ到達可能

v_j から v_i へ到達可能



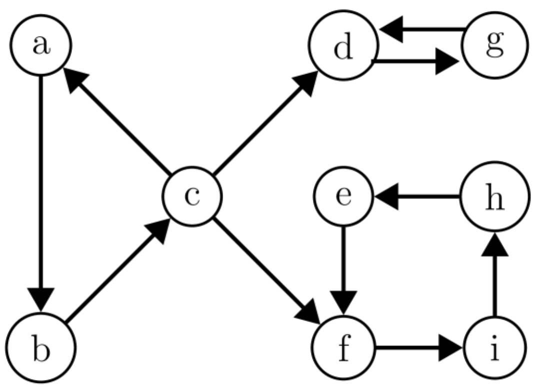
注意： 頂点1つでも、強連結成分

- 強連結成分： 頂点集合で示すと{a, b, c}, {d, g}, {e, f, h, i} の3個

強連結成分分解

Step 1: 有向グラフ $G = (V, A)$ に対し、DFS を実行
この時、各頂点 v に最後に訪問した時間の順番を
f[v] として記憶する

Step 2: G の有向辺を逆向きにしたグラフ G' に対し、
f[v] が大きい未探索の頂点から順に DFS を実行

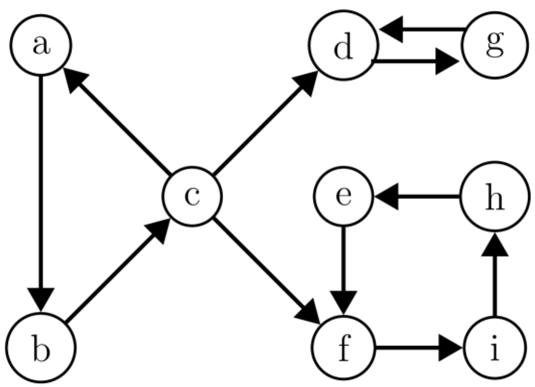


メモ： この例での
頂点の隣接リストは、
アルファベット順とする

強連結成分分解

Step 1: 有向グラフ $G = (V, A)$ に対し、DFS を実行
この時、各頂点 v に最後に訪問した時間の順番を
f[v] として記憶する

Step 2: G の有向辺を逆向きにしたグラフ G’ に対し、
f[v] が大きい未探索の頂点から順に DFS を実行



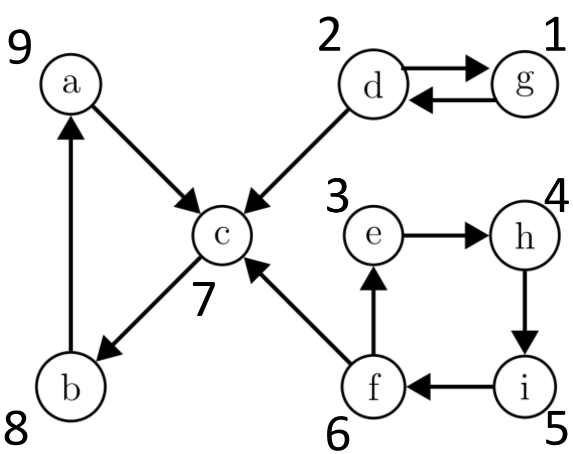
Step 1

- 頂点 a から DFS を始めてみる
- a → b → c → d → g ときて、行き止まり
- g から戻るタイミングで、f[g] ← 1
- d に戻って、ここも行き止まりなので f[d] ← 2
- c に戻って、f → i → h → e ときて、
  行き止まりなので、f[e] ← 3
- h に戻って、f[h] ← 4
- 以降、順に戻って、
  f[i] ← 5, f[f] ← 6, f[c] ← 7, f[b] ← 8, f[a] ← 9

強連結成分分解

Step 1: 有向グラフ $G = (V, A)$ に対し、DFS を実行
この時、各頂点 v に最後に訪問した時間の順番を
f[v] として記憶する

Step 2: G の有向辺を逆向きにしたグラフ G’ に対し、
f[v] が大きい未探索の頂点から順に DFS を実行

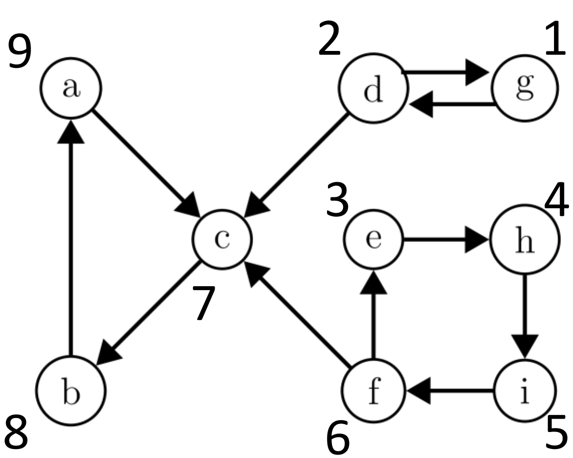


Step 2 に進む前に、G の有向辺を逆向きにする
(各頂点 v の横の数字は、f[v] の値)

強連結成分分解

Step 1: 有向グラフ $G = (V, A)$ に対し、DFS を実行
この時、各頂点 v に最後に訪問した時間の順番を
f[v] として記憶する

Step 2: G の有向辺を逆向きにしたグラフ G’ に対し、
f[v] が大きい未探索の頂点から順に DFS を実行



Step 2

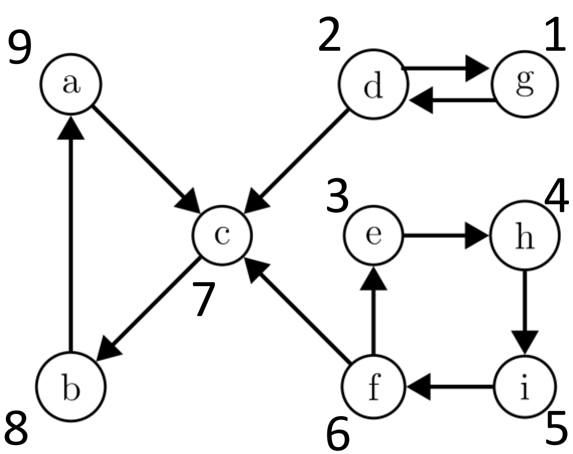
- f[ ] の値が最大の未探索頂点は a
- 頂点 a から DFS：
  a → c → b と進み、ここで再帰から戻ってくる
  この回に探索済みにした {a, b, c} が強連結成分
- f[ ] の値が最大の未探索頂点は f
- 頂点 f から DFS：
  f → e → h → i と進み、ここで再帰から戻ってくる
  {e, f, h, i} が強連結成分
- 同様に、頂点 d から DFS で {d, g} が強連結成分

強連結成分分解

発展

Step 1: 有向グラフ $G = (V, A)$ に対し、DFS を実行
この時、各頂点 v に最後に訪問した時間の順番を
f[v] として記憶する

Step 2: G の有向辺を逆向きにしたグラフ G’ に対し、
f[v] が大きい未探索の頂点から順に DFS を実行



u から v、v から u 両方の
有向パスがある → 強連結

- 「Step 2 で、頂点 u から開始して、
グラフ G’ 上で頂点 v に行けた」
これは、何を意味する？
  - もとに戻すと、
グラフ G 上で、v から u に行ける
  - Step 2 を頂点 u から開始したということは、
f[u] > f[v] だった
… 証明は省略するが、Step 1 の探索で、
u が訪問されてから v が訪問されること
を意味する

今日のまとめ

- 頂点の次数
- さまざまな グラフ
  - 完全グラフ
  - 路 (パス、道)、有向路
  - 閉路 (サイクル)、有向閉路
  - 二部グラフ
  - 完全二部グラフ
- 連結成分、連結成分分解
- 強連結成分、強連結成分分解