教養としての
アルゴリズムとデータ構造
「グラフ」

小林浩二

kojikoba@mi.u-tokyo.ac.jp

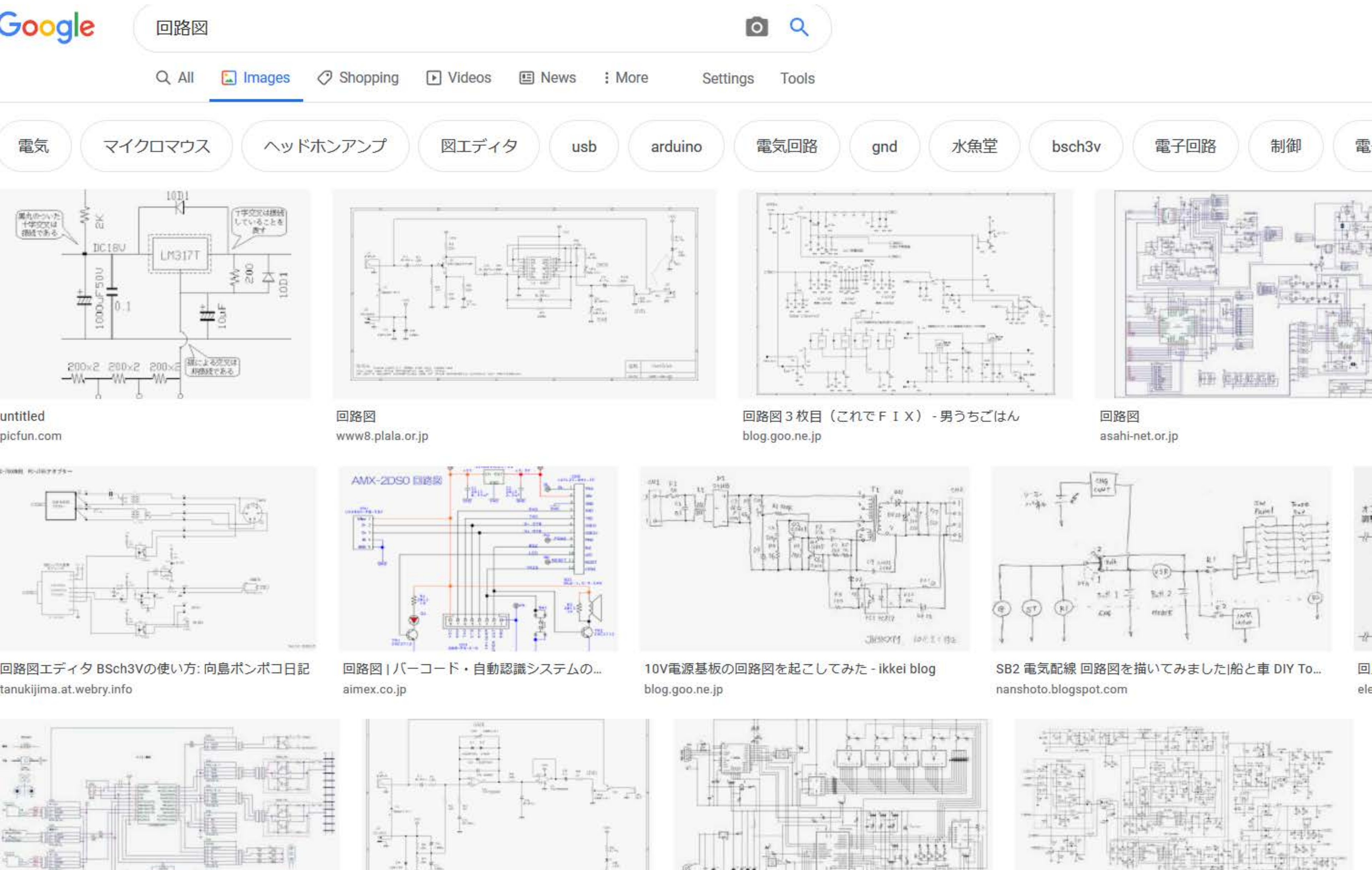
グラフとは

- グラフ（構造）とは、ネットワーク状にデータを保持するデータ構造のこと
  - 木構造の一般化
    - 木と違ってグラフでは一周回って戻ってこられる様な点と枝が存在し得る
  - ネットワーク状のデータの例：
    - 交通網（交通機関の路線図）
    - WebサイトやSNSの接続（フォロー）関係
    - 電子回路
    - 論文の共著関係
      - 著者が点、共著の論文がある著者同士に枝がある
    - 化合物の構造式

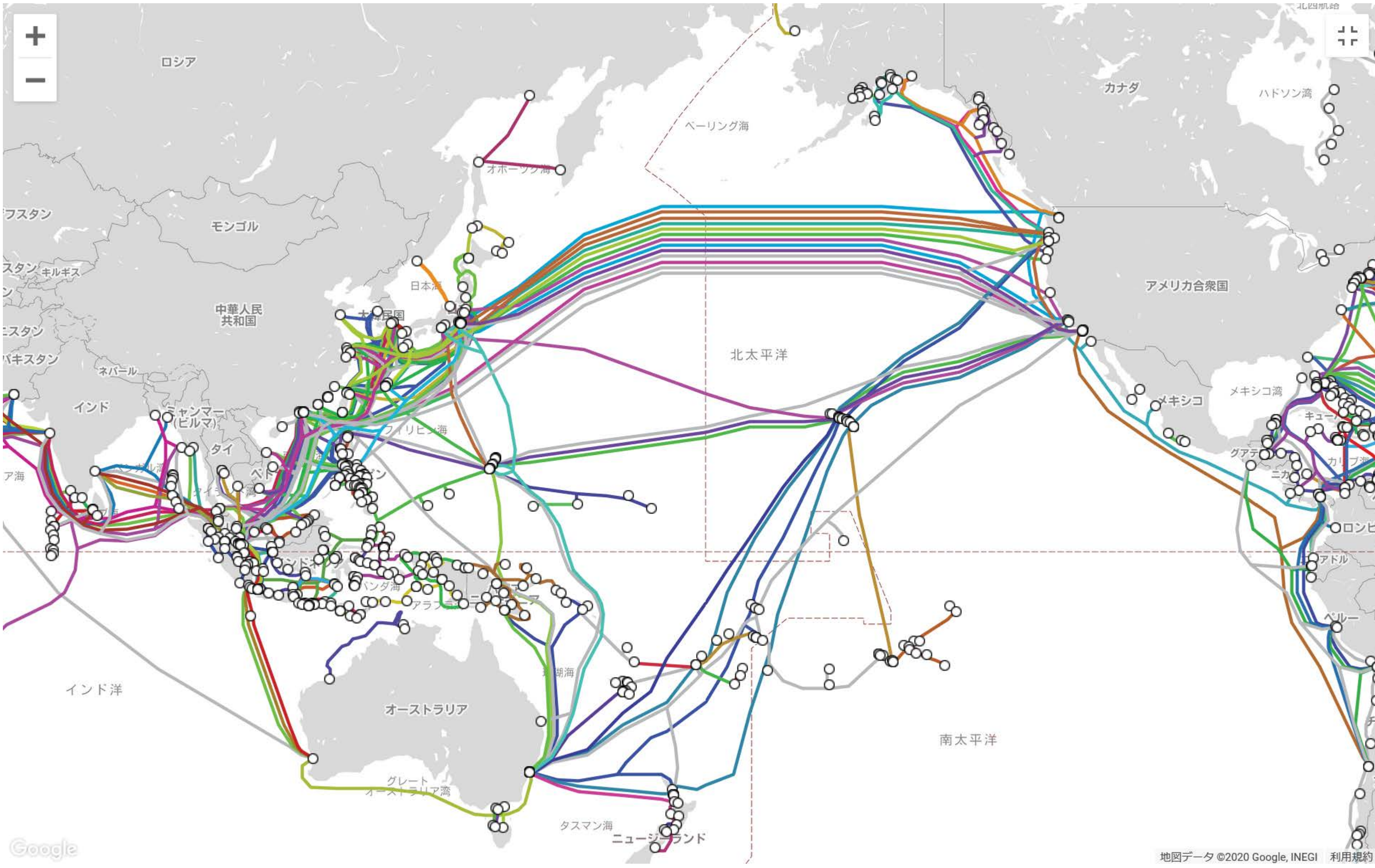
交通網

(c) 東京地下鉄株式会社のWebサイトより

回路図


Google
回路図
All
Images
Shopping
Videos
News
More
Settings
Tools
電気
マイクロマウス
ヘッドホンアンプ
図エディタ
usb
arduino
電気回路
gnd
水魚堂
bsch3v
電子回路
制御
untitled
picfun.com
回路図
www8.plala.or.jp
回路図３枚目（これでＦＩＸ） - 男うちごはん
blog.goo.ne.jp
回路図
asahi-net.or.jp
回路図エディタ BSch3Vの使い方: 向島ポンポコ日記
tanukijima.at.webry.info
回路図 | バーコード・自動認識システムの...
aimex.co.jp
10V電源基板の回路図を起こしてみた - ikkei blog
blog.goo.ne.jp
SB2 電気配線 回路図を描いてみました|船と車 DIY To...
nanshoto.blogspot.com

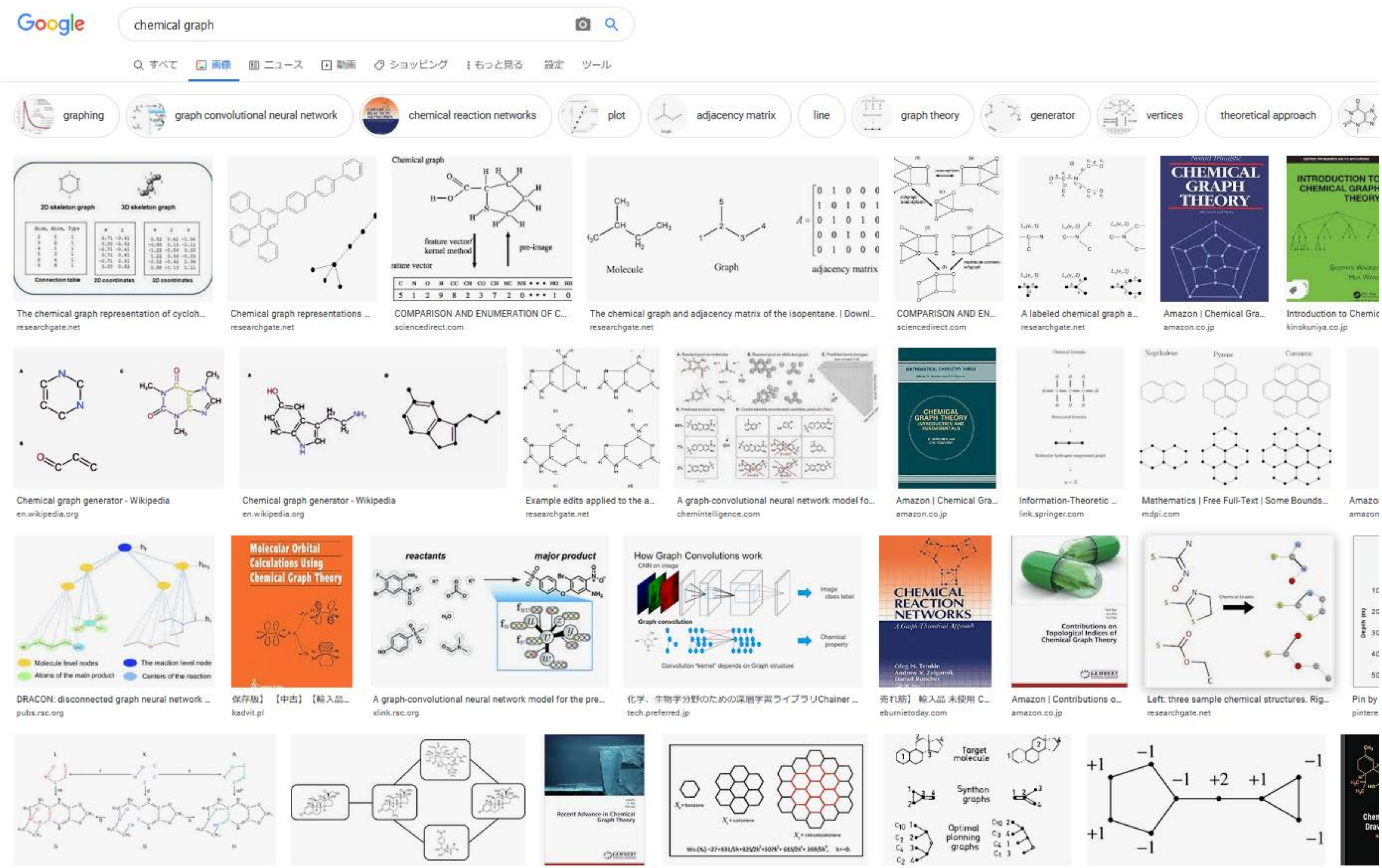
海底ケーブル(の位置)



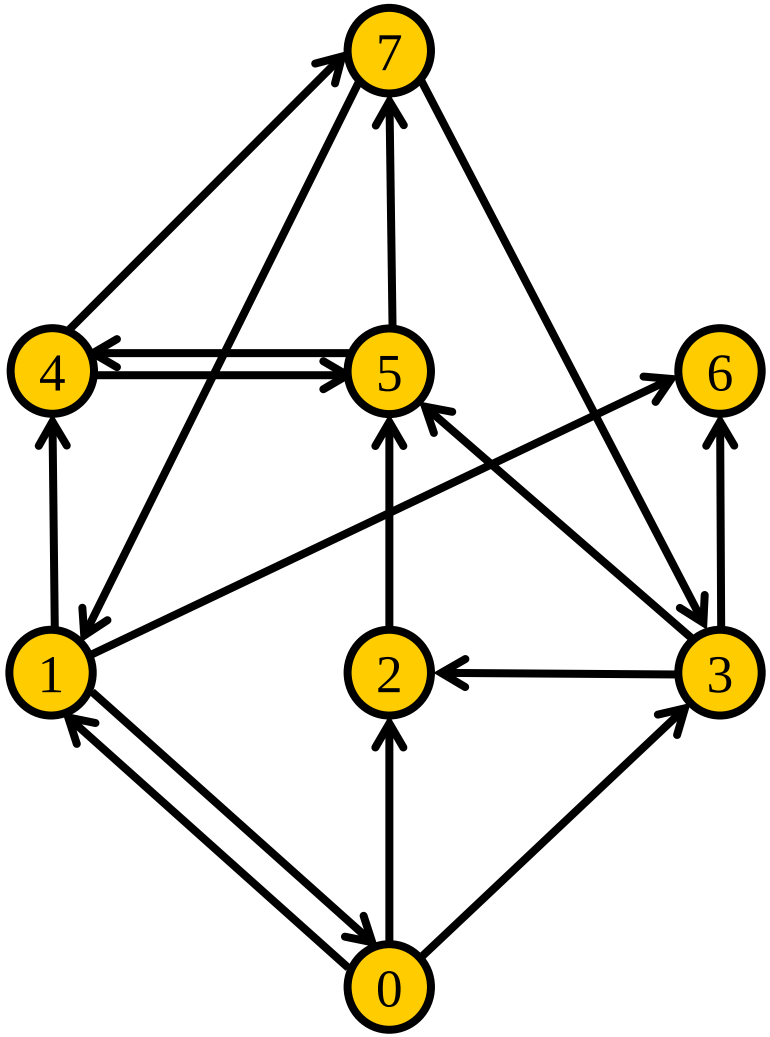
ロシア
カナダ
ハドソン湾
ベーリング海
オホーツク海
モンゴル
中華人民共和国
日本海
アメリカ合衆国
北太平洋
メキシコ
メキシコ湾
インド
ネパール
パキスタン
キルギス
ミャンマー (ビルマ)
タイ
フィリピン海
ペルー
インド洋
オーストラリア
南太平洋
グレート・オーストラリア湾
タスマン海
ニュージーランド
Google
地図データ ©2020 Google, INEGI
利用規約

化合物の構造式

- Chemical Graphs



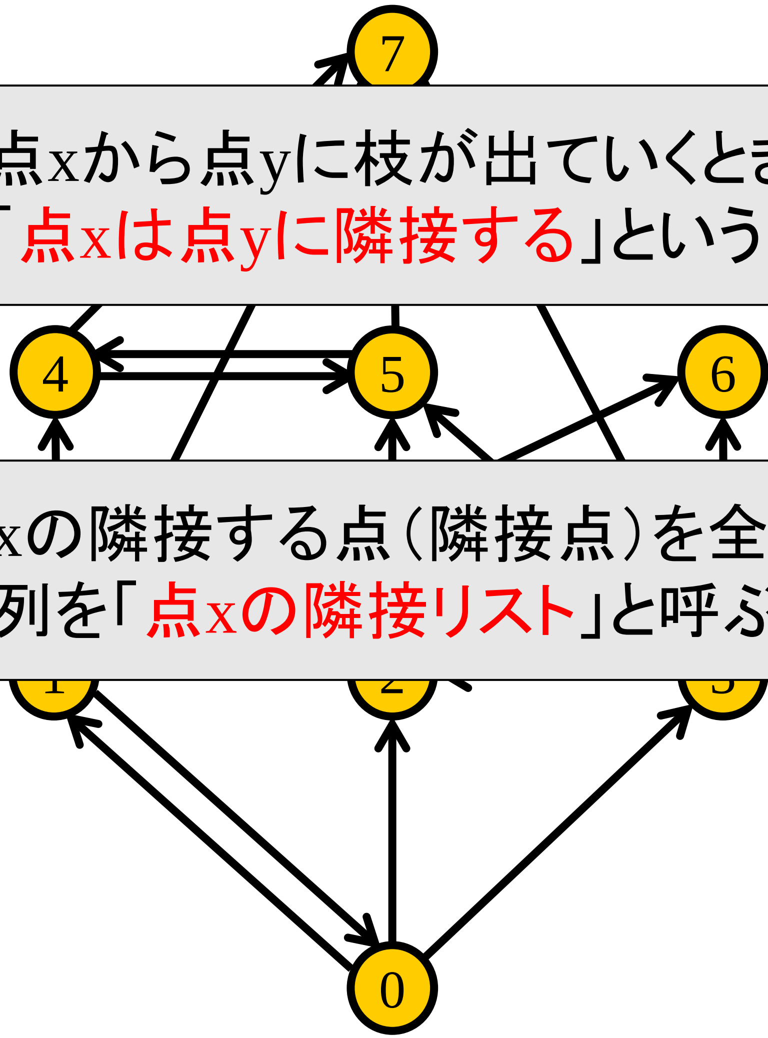
グラフの例



グラフの表現方法

- 最も単純な方法：
  - 大きさ2の配列の集まりで表す
    - 非常に利便性が低い（計算量的によろしくない）
- ネットワークを活かした方法：
  - 隣接リスト
    - 多次元配列で実現
      - Pythonでは多重リスト（2次元）
  - 隣接行列
    - 多次元配列、ハッシュで実現
      - Pythonでは多重リスト（2次元）、辞書、集合（セット）

隣接リストの例



点xから点yに枝が出ていくとき、
「点xは点yに隣接する」という

点xの隣接する点(隣接点)を全て集めた
配列を「点xの隣接リスト」と呼ぶ

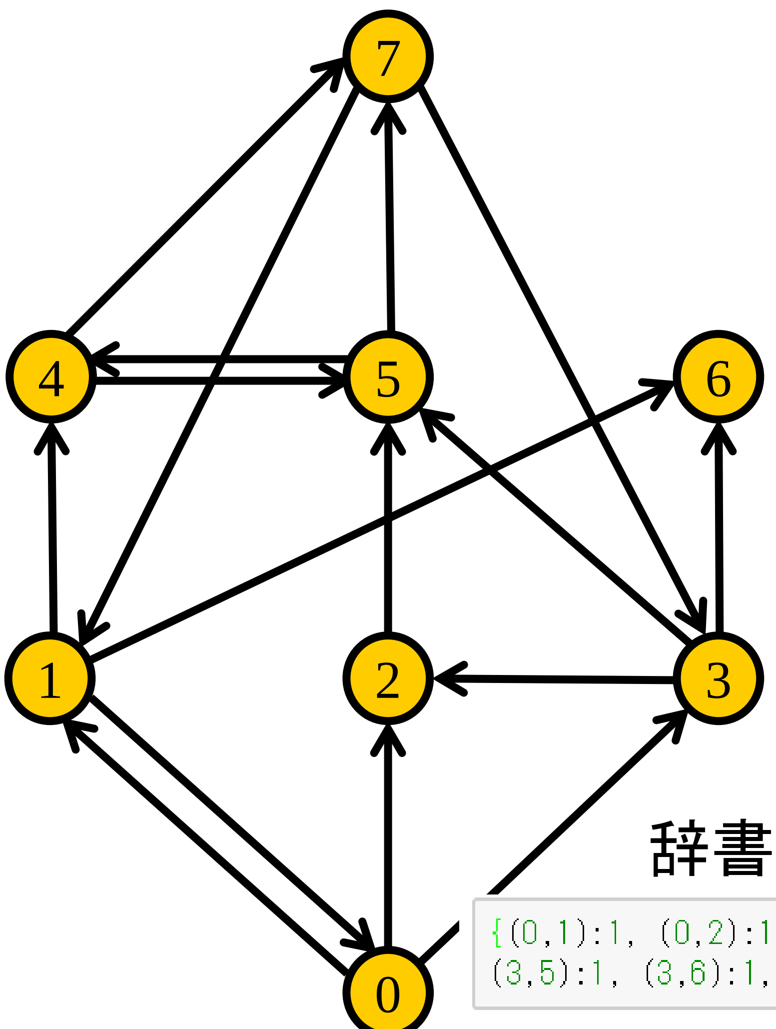
多重配列

0	1	2	3
1	0	4	6
2	5		
3	2	5	6
4	5	7	
5	4	7	
6			
7	1	3	

内側のリストの値が昇順に並んでいる必要はないが、
二分探索などを使うことを考えるとその方が便利

```
[[1,2,3], [0, 4, 6], [5], [2, 5, 6], [5, 7], [4, 7], [], [1, 3]]
```

隣接行列の例



隣接行列

枝の伸びている先

枝の伸びている元

$$
\begin{array}{c}
 \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7
\end{array}
\begin{array}{c}
\begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
\begin{pmatrix}
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0
\end{pmatrix}
\end{array}
$$

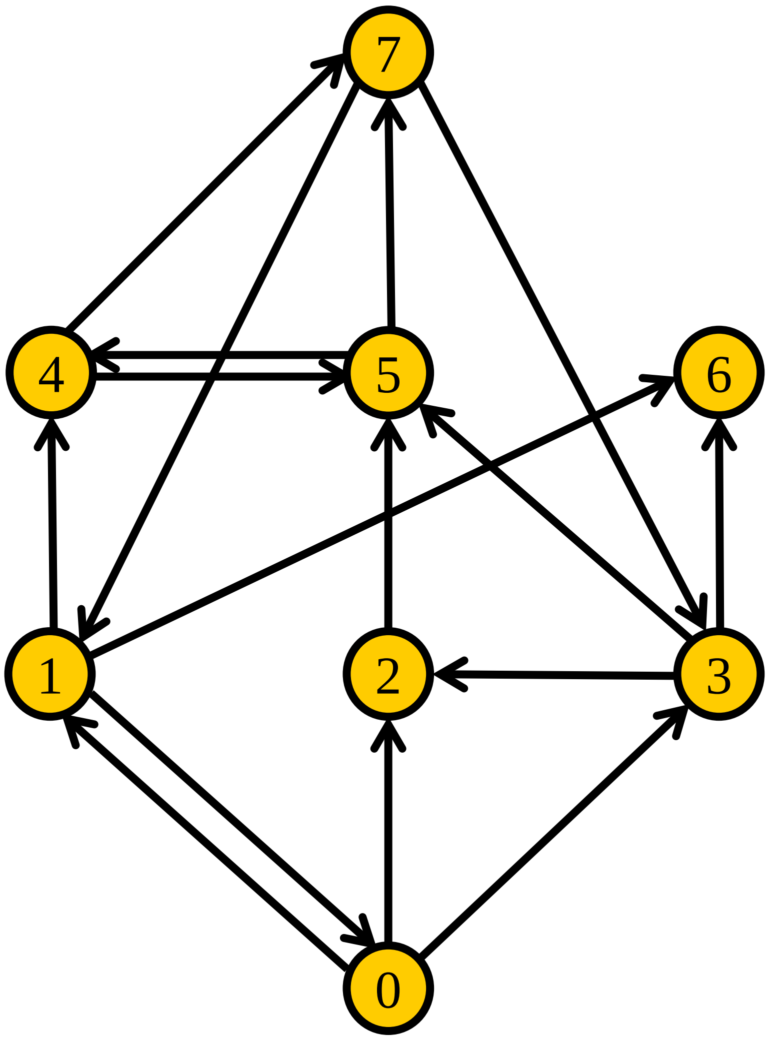
- 点vから点uへの枝がある場合、v行u列を1とする（逆でも良い）

辞書

```
{(0,1):1, (0,2):1, (0,3):1, (1,0):1, (1,4):1, (1,6):1, (2,5):1, (3,2):1,
(3,5):1, (3,6):1, (4,5):1, (4,7):1, (5,4):1, (5,7):1, (7,1):1, (7,3):1 }
```

- タプル(0,1)を文字列'0,1'などとしても良い

隣接行列の例



多重リスト

```
[[0, 1, 1, 1, 0, 0, 0, 0],
 [1, 0, 0, 0, 1, 0, 1, 0],
 [0, 0, 0, 0, 0, 1, 0, 0],
 [0, 0, 1, 0, 0, 1, 1, 0],
 [0, 0, 0, 0, 0, 1, 0, 1],
 [0, 0, 0, 0, 1, 0, 0, 1],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 1, 0, 1, 0, 0, 0, 0]]
```

- 多重リストを使って表しても良い

（頂点数が多い場合は記憶領域を大量に使うことに注意）

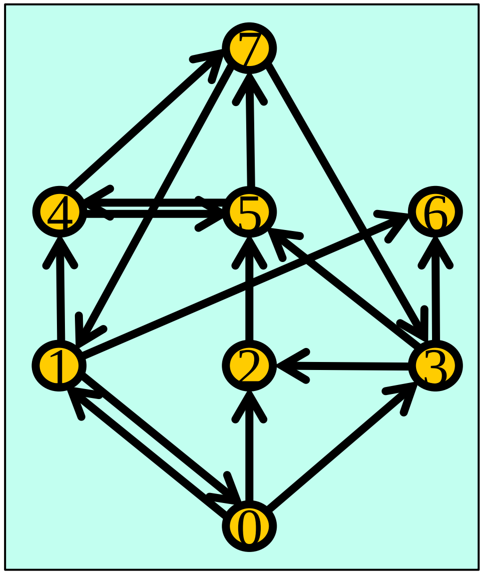
有向グラフ / 無向グラフ

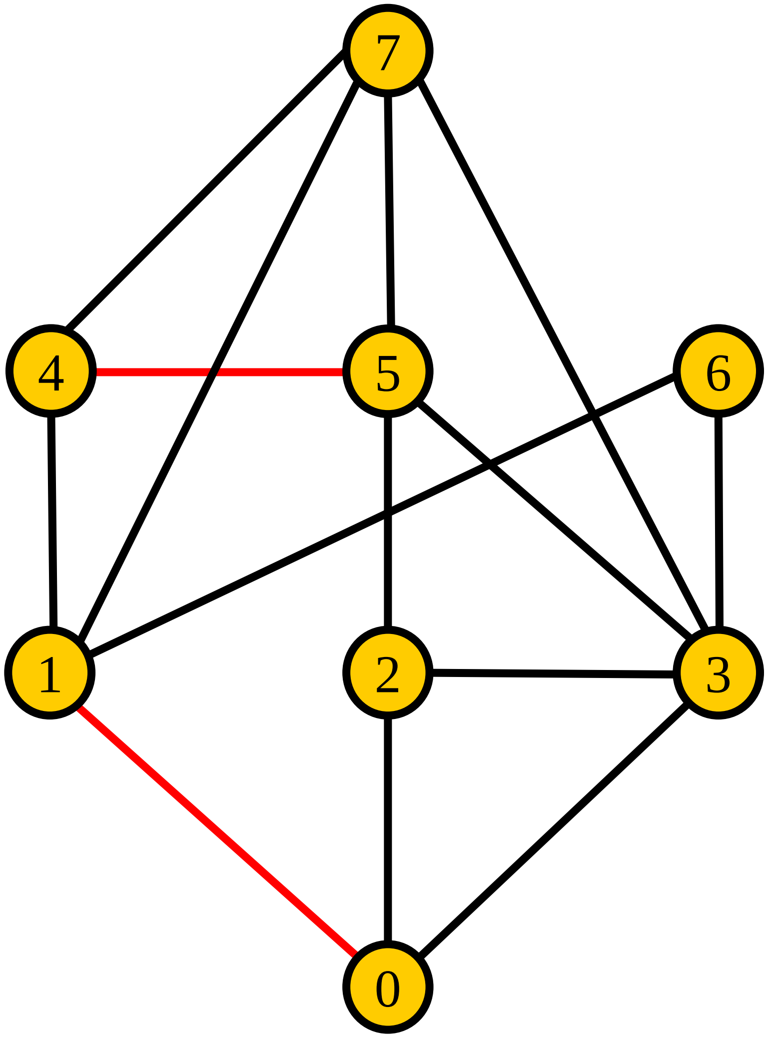
- 有向グラフ：枝に向きのあるグラフ
- 無向グラフ：枝に向きのないグラフ

- 例：
  - 交通網：一方通行あり（なし）→有向（無向）
  - SNSのフォロー関係：有向（全て相互なら、無向）
  - 電気回路：有向
  - 論文の共著関係：無向
    - 著者が点、共著の論文がある著者同士に枝がある

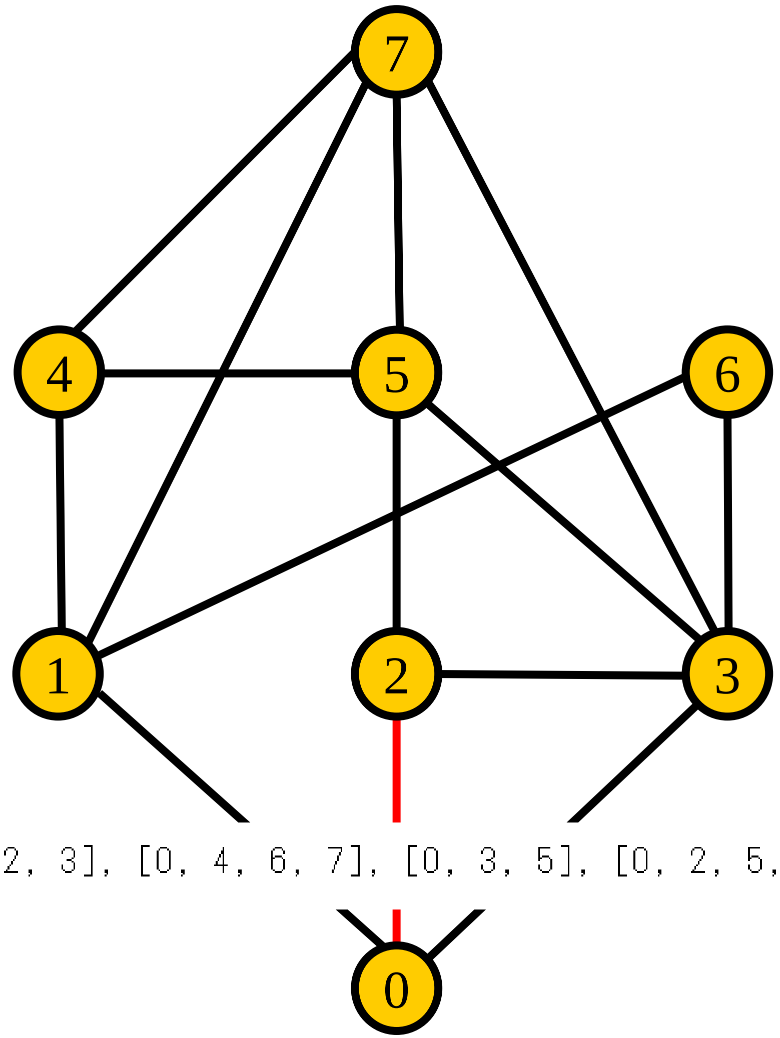
無向グラフの例

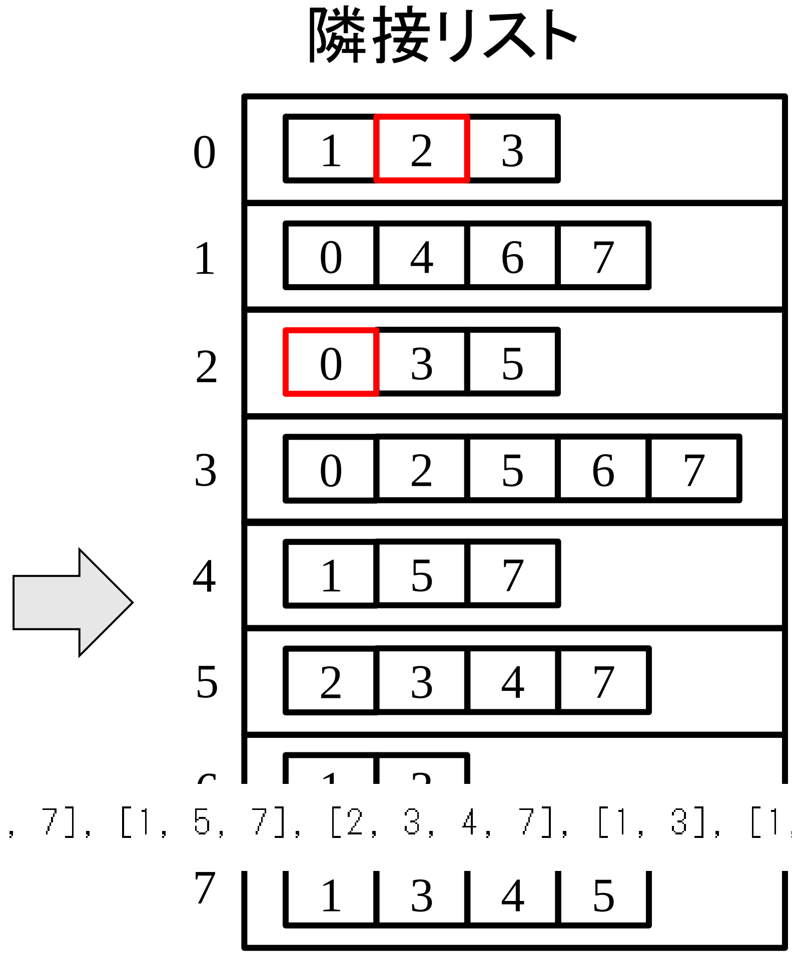
- 先の有向グラフの例で点uから点v、もしくは点vから点uへの枝があれば枝がある様な無向グラフ
  - 無向グラフでは2点間に枝は高々1本になる
    - 有向グラフでは点0と1、点4と5は両方向の枝があった





無向グラフの隣接リストの例



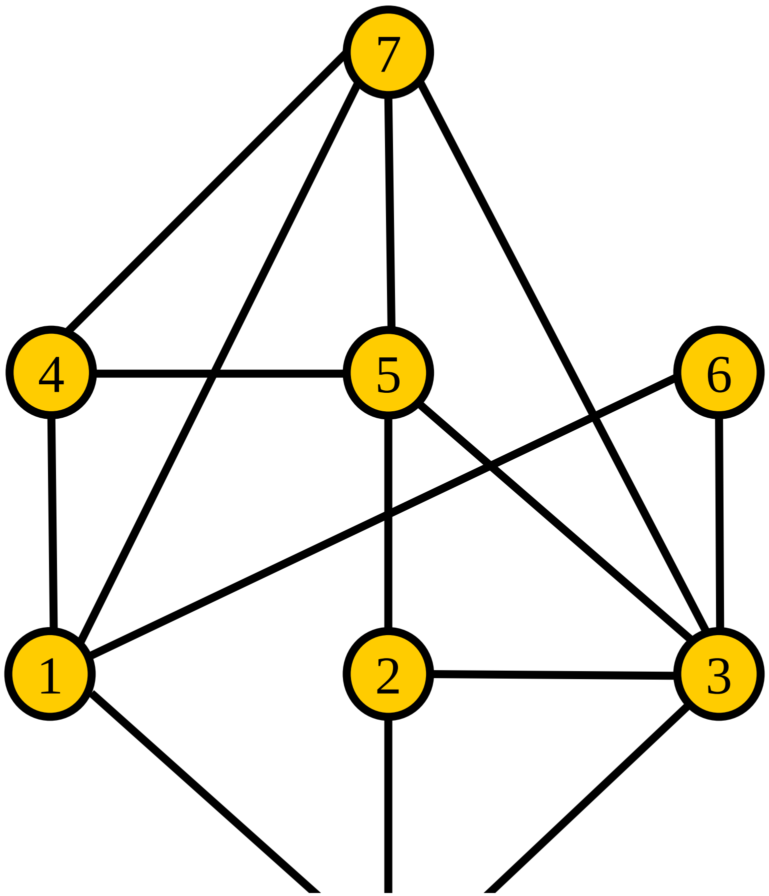


[[1, 2, 3], [0, 4, 6, 7], [0, 3, 5], [0, 2, 5, 6, 7], [1, 5, 7], [2, 3, 4, 7], [1, 3], [1, 3, 4, 5]]

内側のリストの値が昇順に並んでいる必要はないが、二分探索などを使うことを考えるとその方が便利

- 1本の枝に対して2つの値を保持
  - 例：点0と点2の間に枝→点0の隣接リストに2、点2の隣接リストに0が含まれる

無向グラフの隣接行列の例



隣接行列

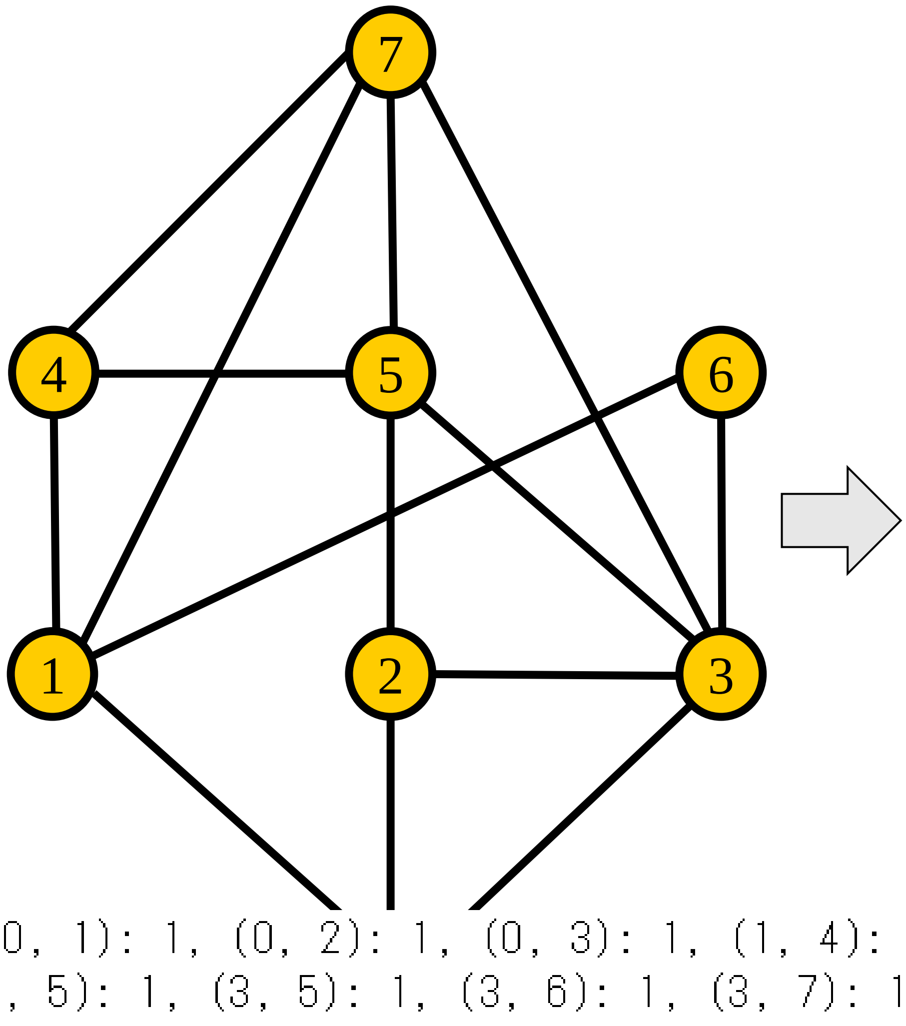
枝の伸びている先

$$
\text{枝の伸びている元}\quad
\begin{array}{c}
\\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7
\end{array}
\begin{array}{c}
\begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
\begin{pmatrix}
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 & 0
\end{pmatrix}
\end{array}
$$

- 点vと点uの間に枝がある場合、v行u列とu行v列を1とする
- 対称な行列になっている

{(0, 1): 1, (0, 2): 1, (0, 3): 1, (1, 0): 1, (1, 4): 1, (1, 6): 1, (1, 7): 1,
(2, 0): 1, (2, 3): 1, (2, 5): 1, (3, 0): 1, (3, 2): 1, (3, 5): 1, (3, 6): 1,
(3, 7): 1, (4, 1): 1, (4, 5): 1, (4, 7): 1, (5, 2): 1, (5, 3): 1, (5, 4): 1,
(5, 7): 1, (6, 1): 1, (6, 3): 1, (7, 1): 1, (7, 3): 1, (7, 4): 1, (7, 5): 1}

無向グラフの隣接行列の例



隣接行列

枝の伸びている先

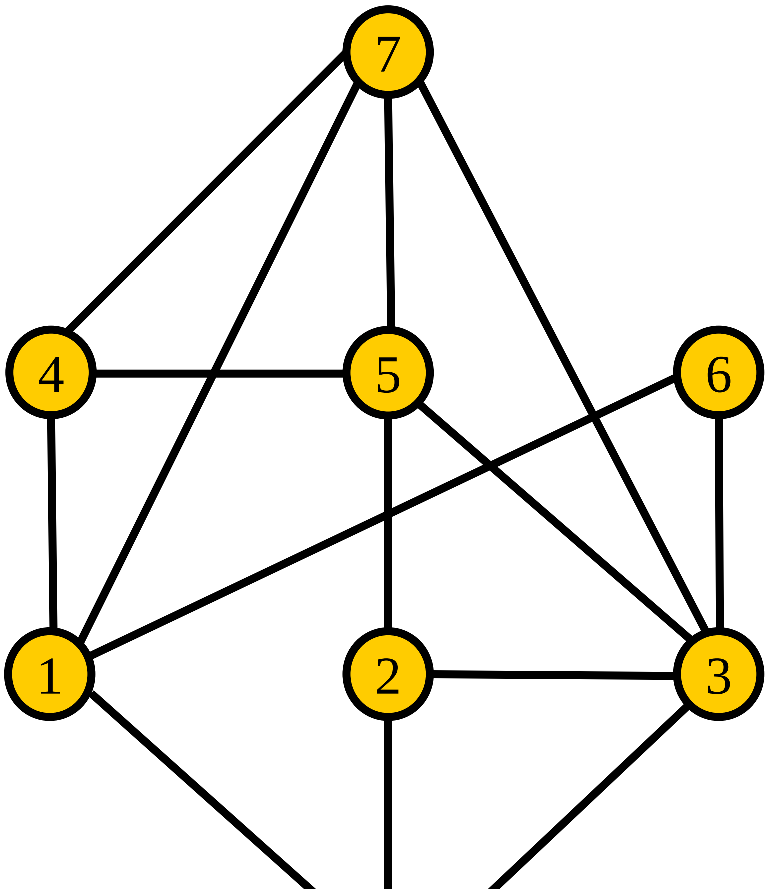
枝の伸びている元

$$
\begin{array}{cc}
 & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} \\
\begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} &
\begin{pmatrix}
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\end{array}
$$

- 対称性からv行u列とu行v列のどちらから一方のみを1とするだけでも良い
  - u行v列（u < v）のみを1にする場合

```
{(0, 1): 1, (0, 2): 1, (0, 3): 1, (1, 4): 1, (1, 6): 1, (1, 7): 1, (2, 3): 1,
(2, 5): 1, (3, 5): 1, (3, 6): 1, (3, 7): 1, (4, 5): 1, (4, 7): 1, (5, 7): 1}
```

無向グラフの隣接行列の例



隣接行列

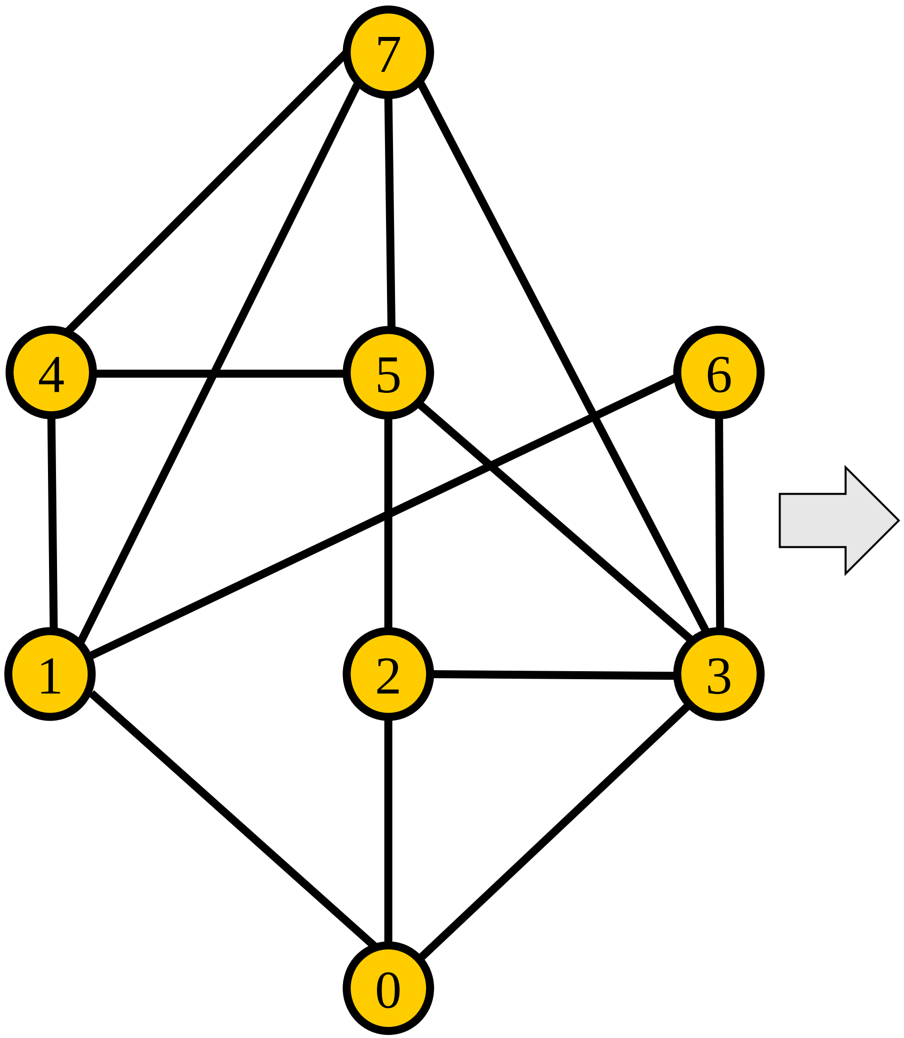
枝の伸びている先

$$
\text{枝の伸びている元}\quad
\begin{array}{c}
 \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7
\end{array}
\begin{array}{c}
\begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 & 0
\end{pmatrix}
\end{array}
$$

- 対称性からv行u列とu行v列のどちらから一方のみを1とするだけでも良い
  - u行v列（v < u）のみを1にする場合

```
{(1, 0): 1, (2, 0): 1, (3, 0): 1, (3, 2): 1, (4, 1): 1, (5, 2): 1, (5, 3): 1,
(5, 4): 1, (6, 1): 1, (6, 3): 1, (7, 1): 1, (7, 3): 1, (7, 4): 1, (7, 5): 1}
```

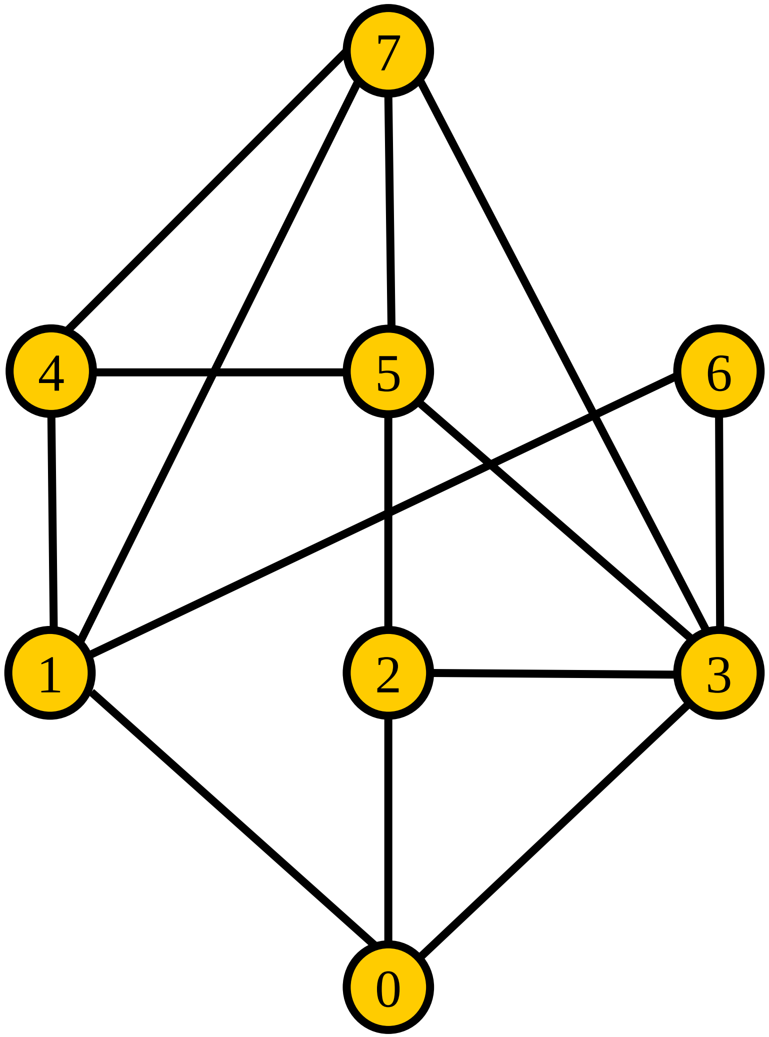
無向グラフの隣接行列の例



多重リスト

```
[[0, 1, 1, 1, 0, 0, 0, 0],
 [1, 0, 0, 0, 1, 0, 1, 1],
 [1, 0, 0, 1, 0, 1, 0, 0],
 [1, 0, 1, 0, 0, 1, 1, 1],
 [0, 1, 0, 0, 0, 1, 0, 1],
 [0, 0, 1, 1, 1, 0, 0, 1],
 [0, 1, 0, 1, 0, 0, 0, 0],
 [0, 1, 0, 1, 1, 1, 0, 0]]
```

無向グラフの隣接行列の例

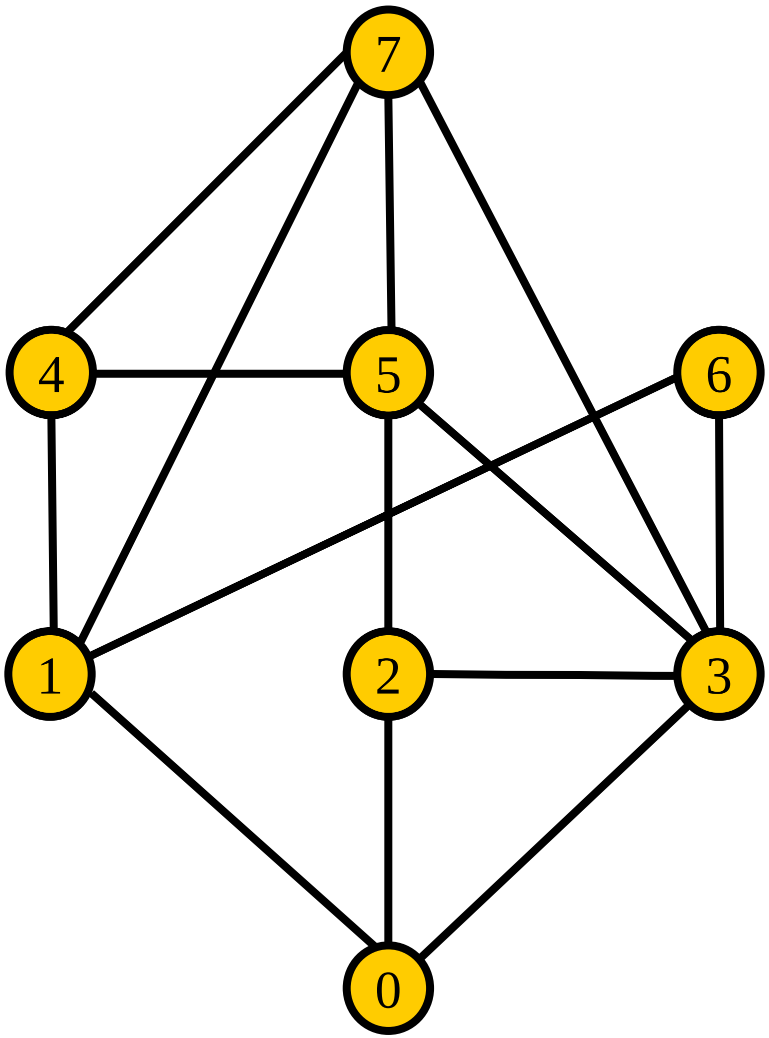


多重リスト

```
[[0, 1, 1, 1, 0, 0, 0, 0],
 [0, 0, 0, 0, 1, 0, 1, 1],
 [0, 0, 0, 1, 0, 1, 0, 0],
 [0, 0, 0, 0, 0, 1, 1, 1],
 [0, 0, 0, 0, 0, 1, 0, 1],
 [0, 0, 0, 0, 0, 0, 0, 1],
 [0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0]]
```

- 対称性から[u][v]と[v][u]のどちらか一方を1にするだけでも良い
  - [u][v] (u < v) のみ1にする場合

無向グラフの隣接行列の例



多重リスト

```
[[0, 0, 0, 0, 0, 0, 0, 0],
 [1, 0, 0, 0, 0, 0, 0, 0],
 [1, 0, 0, 0, 0, 0, 0, 0],
 [1, 0, 1, 0, 0, 0, 0, 0],
 [0, 1, 0, 0, 0, 0, 0, 0],
 [0, 0, 1, 1, 1, 0, 0, 0],
 [0, 1, 0, 1, 0, 0, 0, 0],
 [0, 1, 0, 1, 1, 1, 0, 0]]
```

- 対称性から[u][v]と[v][u]のどちらか一方を1にするだけでも良い
  - [u][v] (v < u) のみ1にする場合

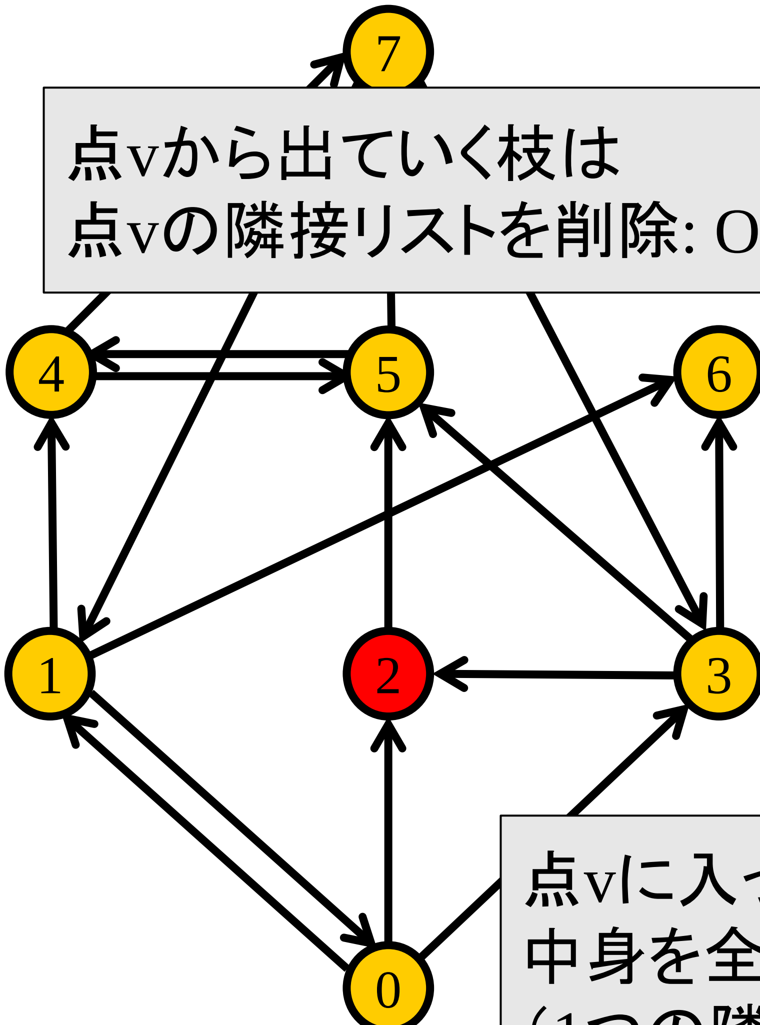
隣接リスト vs. 隣接行列

- 計算量の比較:
  - n個の点があり、点vの隣接点の数がa個の場合:
  - 点vの隣接(する)点uをk回利用する場合:
    - 隣接リスト:
      - 線形探索でuを探す: O(ka)
      - リストがソートされていれば二分探索: $O(k \log_2 a)$ (ただし、ソート は $O(a \log_2 a)$)
        - » 隣接点の利用回数kが大きい場合: ソート→二分探索
        - » 第2回のスライド参照
        - » グラフが変化しない場合は、uを格納するインデックスを記憶
    - 隣接行列: O(k)
  - 点vの隣接点を全て利用する場合:
    - 隣接リスト: O(a)
    - 隣接行列: 隣接点の数何個かあるか分からないので、(v, ?) となる「?」 が1からnまでの場合を調べる必要がある=O(n)

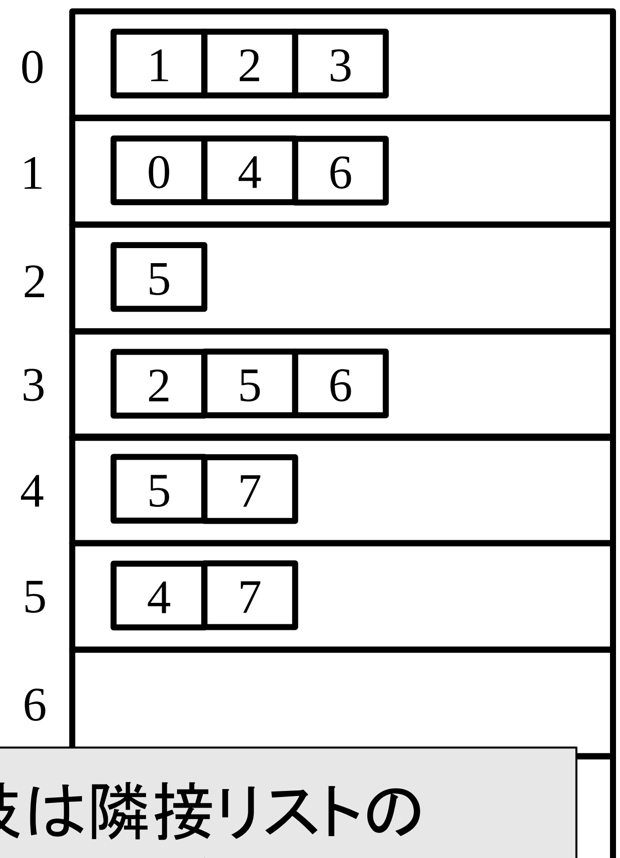
隣接リスト vs. 隣接行列

- 計算量の比較:
  - n個の点、点vの隣接点の数がa個、一番多く隣接点を持つ点の隣接点数がAの場合:
  - 点vを削除する場合:
    - 点v と他の点の間にある枝も削除する必要がある
    - 隣接リスト:

点削除 有向グラフ 隣接リスト



多重配列



点vから出ていく枝は
点vの隣接リストを削除: O(1)

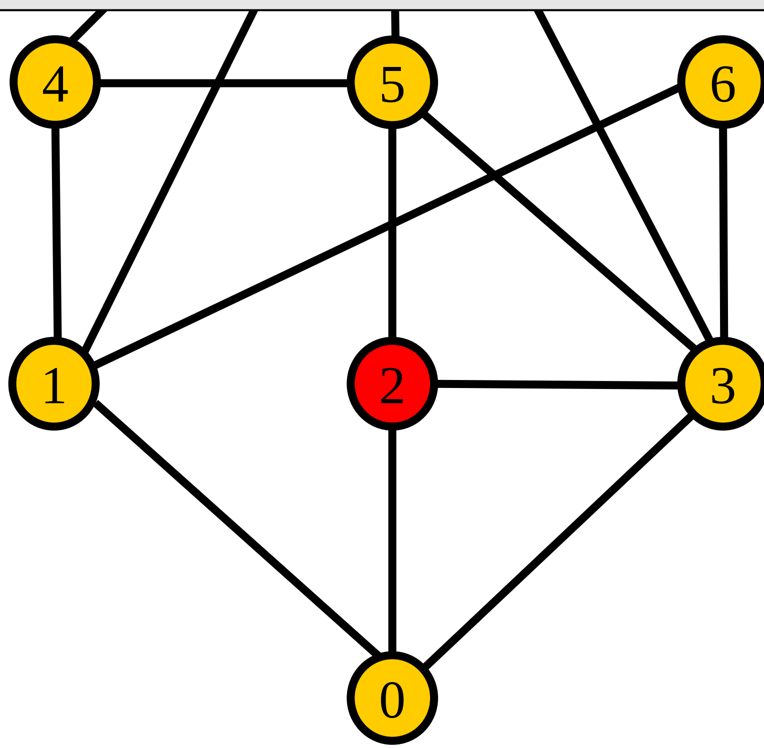
点vに入ってくる枝は隣接リストの
中身を全て調べる必要がある:O(n A)
（1つの隣接リストにつき最大O(A)）

隣接リスト vs. 隣接行列

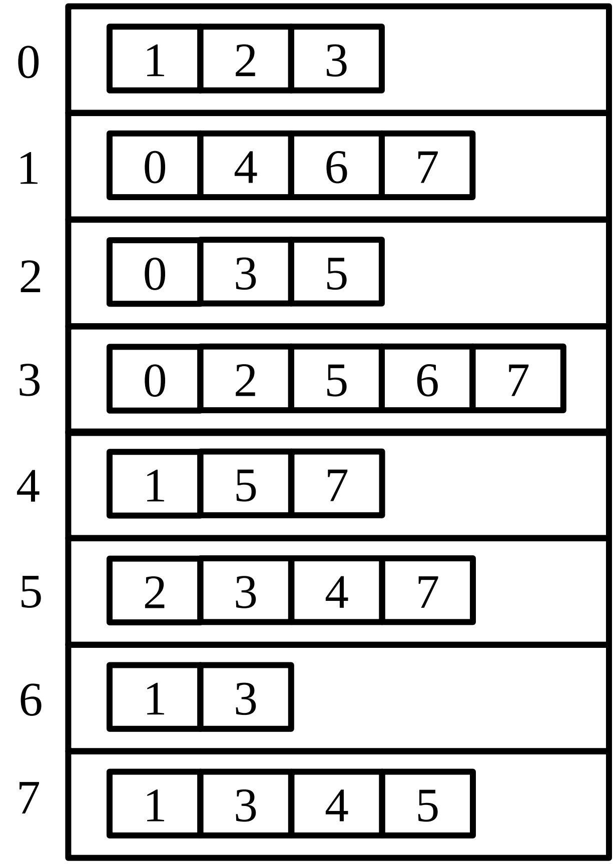
- 計算量の比較:
  - n個の点、点vの隣接点の数がa個、一番多く隣接点を持つ点の隣接点数がAの場合:
  - 点vを削除する場合:
    - 点v と他の点の間にある枝も削除する必要がある
    - 隣接リスト:
      - 有向グラフ: vのリストを削除→vを含むリストの更新 O(n A)
        - » リストがソートされていれば二分探索でO(n $\log_2$ A) （ただし、ソートは O(A $\log_2$ A) ）

点削除 無向グラフ 隣接リスト

点vの隣接リストを見れば、
点vを含む隣接リストが分かる:O(a A)
(1つの隣接リストにつきO(A))



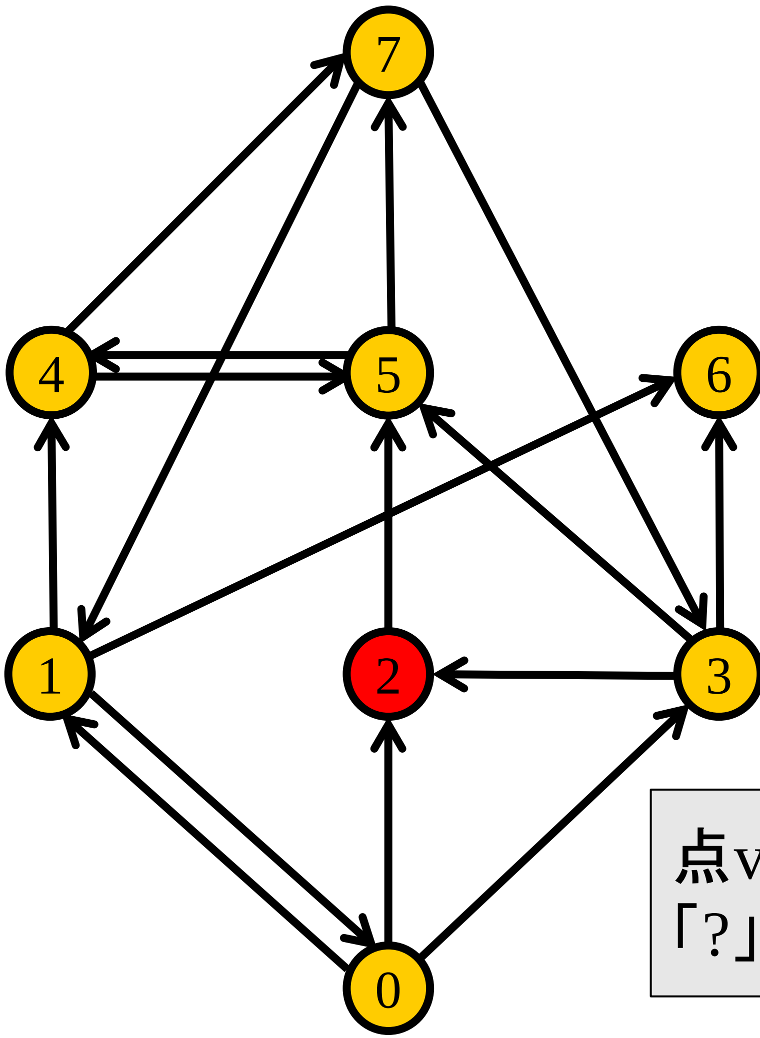
多重配列



隣接リスト vs. 隣接行列

- 計算量の比較:
  - n個の点、点vの隣接点の数がa個、一番多く隣接点を持つ点の隣接点数がAの場合:
  - 点vを削除する場合:
    - 点v と他の点の間にある枝も削除する必要がある
    - 隣接リスト:
      - 有向グラフ: vのリストを削除→vを含むリストの更新 O(n A)
        - » リストがソートされていれば二分探索で$O(n \log_2 A)$ （ただし、ソートは $O(A \log_2 A)$ ）
      - 無向グラフ: vのリストを削除→vを含むリストの更新 O(a A)
        - » 事前にリストがソートしてあれば、$O(a \log_2 A)$

点削除 有向グラフ 隣接行列



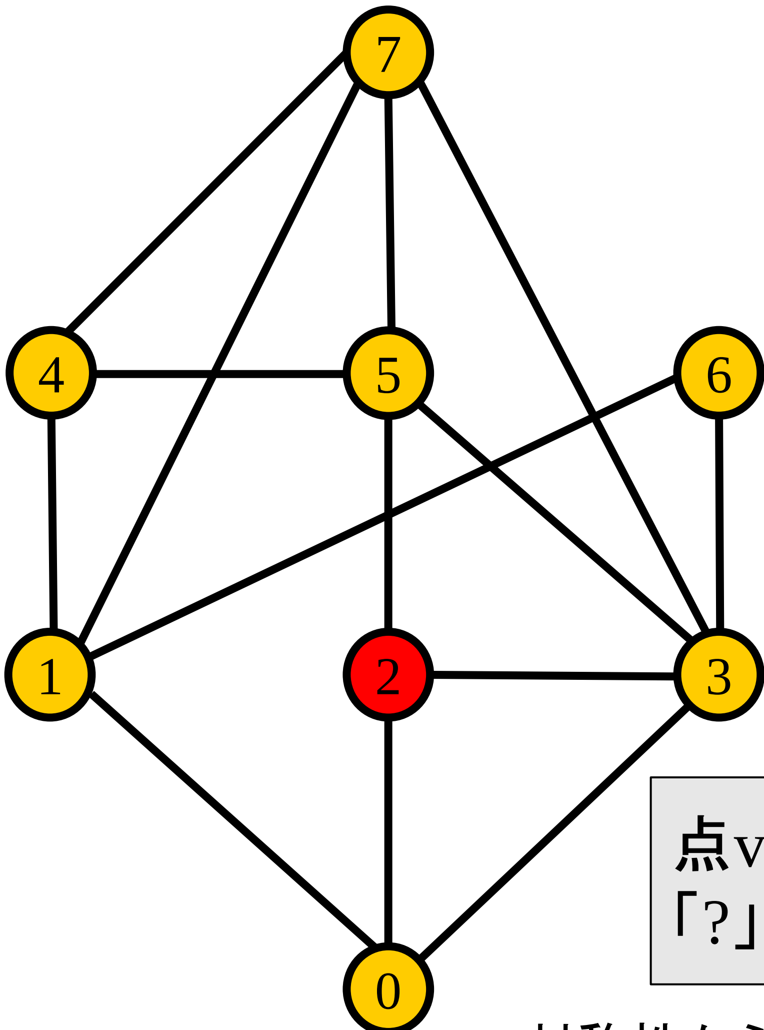
隣接行列

枝の伸びている先

$$
\text{枝の伸びている元}\quad
\begin{array}{c}
\\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7
\end{array}
\begin{array}{c}
\begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
\begin{pmatrix}
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0
\end{pmatrix}
\end{array}
$$

点vを消す場合、(v,?)と(?,v)を全ての「?」について調べる：O(n)

点削除 無向グラフ 隣接行列



隣接行列

枝の伸びている先

$$
\begin{array}{c}
\\
\text{枝の伸びている元}
\end{array}
\begin{array}{c|cccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
0 & 0 & 1 & \color{red}{0} & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
2 & \color{red}{0} & 0 & 0 & \color{red}{0} & 0 & \color{red}{0} & 0 & 0 \\
3 & 1 & 0 & \color{red}{0} & 0 & 0 & 1 & 1 & 1 \\
4 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
5 & 0 & 0 & \color{red}{0} & 1 & 1 & 0 & 0 & 1 \\
6 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
7 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0
\end{array}
$$

点vを消す場合、(v,?)と(?,v)を全ての「?」について調べる：O(n)

- 対称性から、行（列）を調べれば列（行）の位置も分かる

隣接リスト vs. 隣接行列

- 計算量の比較:
  - n個の点、点vの隣接点の数がa個、一番多く隣接点を持つ点の隣接点数がAの場合:
  - 点vを削除する場合:
    - 点v と他の点の間にある枝も削除する必要がある
    - 隣接リスト:
      - 有向グラフ: vのリストを削除→vを含むリストの更新 O(n A)
        - » リストがソートされていれば二分探索でO(n $\log_2$ A) (ただし、ソートは O(a $\log_2$ A))
      - 無向グラフ: vのリストを削除→vを含むリストの更新 O(k K)
        - » 事前にリストがソートしてあれば、O(a $\log_2$ A)
    - 隣接行列: O(n)

隣接リスト vs. 隣接行列

- 領域計算量の比較：
  - n個の点、m本の枝の場合：
    - 隣接リスト：O(m)
    - 隣接行列：
      - 多重配列（リスト）：$O(n^2)$
        - » m > n
        - » 実際のネットワークは m=O(n) の場合が多い
      - ハッシュ（辞書、集合）：O(M)

```
#点1000、枝1000くらいのグラフをランダムに作成
a1 = total_size(list_adjlist) #隣接リスト
a2 = total_size(dic_adjmatrix) #隣接行列（辞書）
a3 = total_size(list_adjmatrix) #隣接行列（多重リスト）
print("隣接リスト：", a1, ", 隣接行列（辞書）：", a2, ", 隣接行列（多重リスト）：", a3)
```

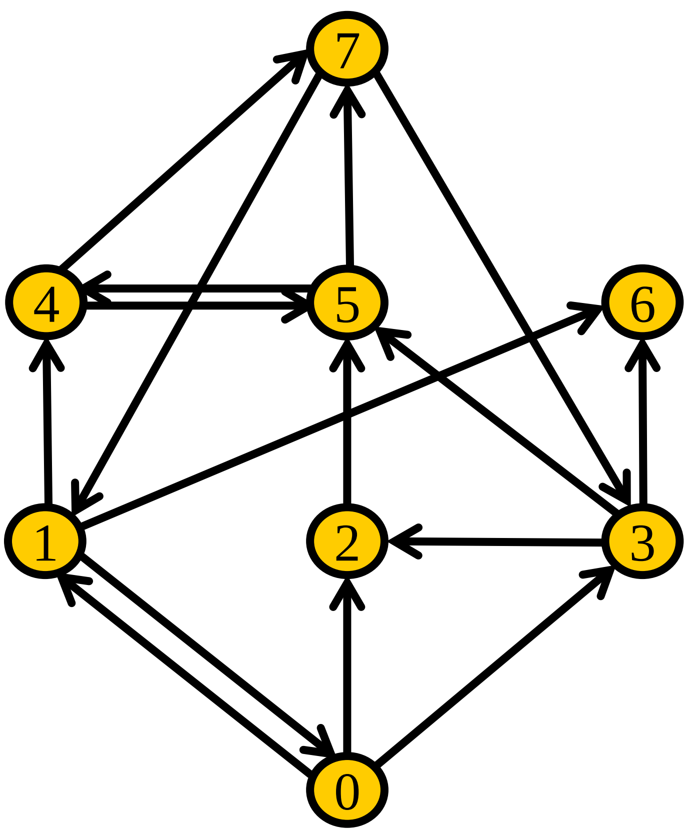
```
隣接リスト： 117944 , 隣接行列（辞書）： 148240 , 隣接行列（多重リスト）： 8072116
```

隣接リスト vs. 隣接行列

- リンク先・元の管理の手間の比較:
  - 隣接リストは面倒
    - 点vの削除:vを探索する手間
    - 点vの追加:vを2つ含まないようにする
      - リストがソートされているときは、追加時にソート状態を保つために適切な位置に追加する手間もある
  - 隣接行列は楽
    - 点vの削除、追加:O(1)
- 実用的には、両者を上手く組み合わせた様なデータ構造を用いる

隣接行列の性質

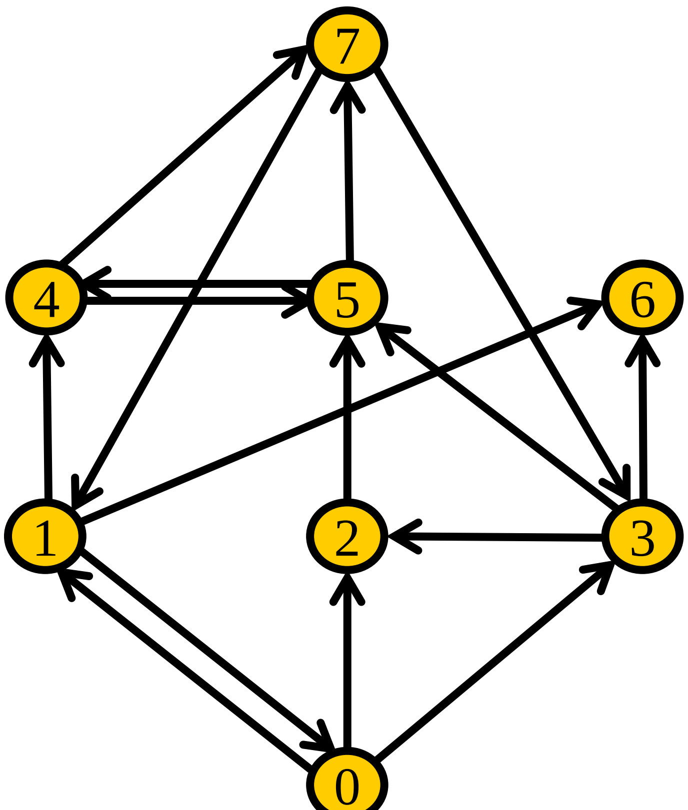
- 隣接行列Aを数学の意味での行列と見なした場合、Aの k 乗 A^kのv行u列の値は、点vから点uへの長さkの経路の数になっている



$$A = \begin{array}{cc} & \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} & \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

隣接行列の性質

- 隣接行列Aを数学の意味での行列と見なした場合、Aの k 乗 A^kのv行u列の値は、点vから点uへの長さkの経路の数になっている



$$A^2 = \begin{array}{cc} & \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} & \begin{pmatrix} \boxed{1} & 0 & 1 & 0 & 1 & 2 & \boxed{2} & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 2 & 0 \end{pmatrix} \end{array}$$

隣接行列の積による長さkの経路の総数の正当性

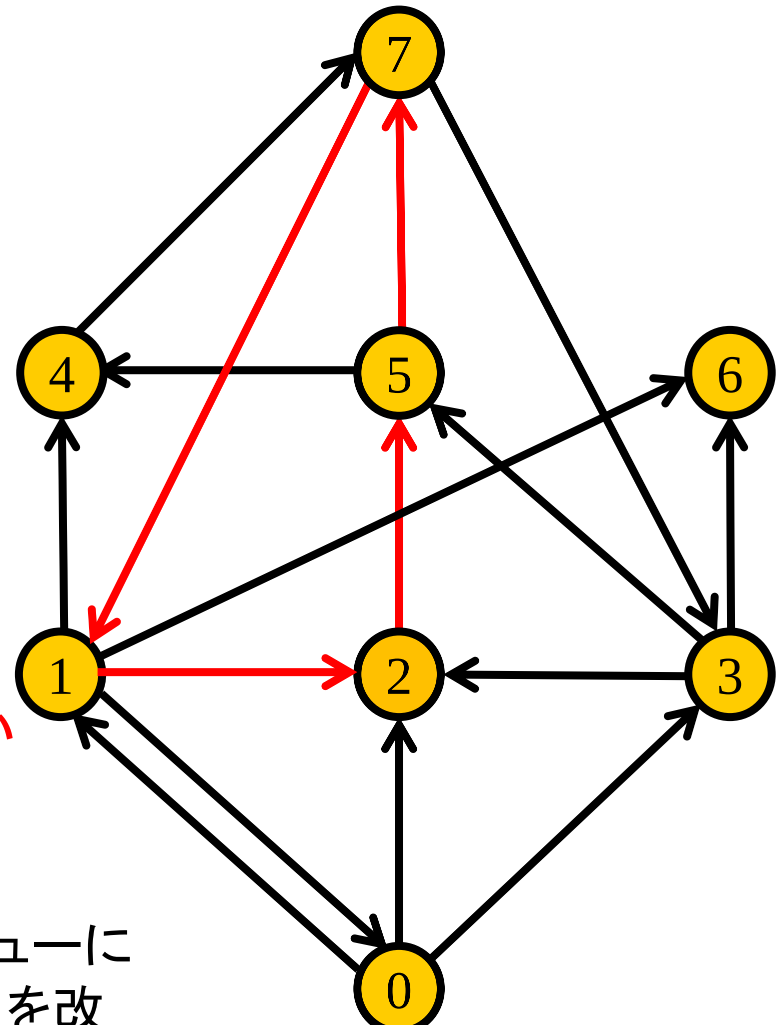
- 隣接行列A (n×n行列)の k 乗 A^kのv行u列の値は、点vから点uへの長さkの経路の数になっていることを k に関する帰納法で示す
- v行u列を(v,u)で表し、行列Xの(v,u)の値をX(v,u)で表す
- k=1の場合、隣接行列の定義より、点vから点uへの枝がある場合にA(v,u)の値が1、そうでなければ0であるので確かにuからvへの長さ1の経路数になっている
- k≦x-1の場合に、A^k(v,u)は、点vから点uへの長さkの経路の数になっていると仮定し、A^x(v,u)は点vから点uへの長さxの経路の数になっていることを示す
- A^x(v,u)は、各i=0,…n-1に対して、A^{x-1}(v,i)×A(i,u)の総和Sになっている
- 帰納法の仮定より、A^{x-1}(v,i)は点vから点iへの長さx-1の経路の数、A(i,u)は点iから点uへの枝の数であり、Sは点vからから点uへの長さxの経路の数になっている

グラフの基本操作

- 基本操作
  - 木構造同様、指定した値（点）を含むかどうか、指定した2点の木構造上の関係（経路、距離など）、条件を満たす点を全て求めるなど
    - 経路＝2点間をつなぐ点と枝の列
    - 距離＝経路上の枝の数
  - グラフの探索によって多くが実現される
    - 幅優先探索
    - 深さ優先探索

有向グラフに対する幅優先探索

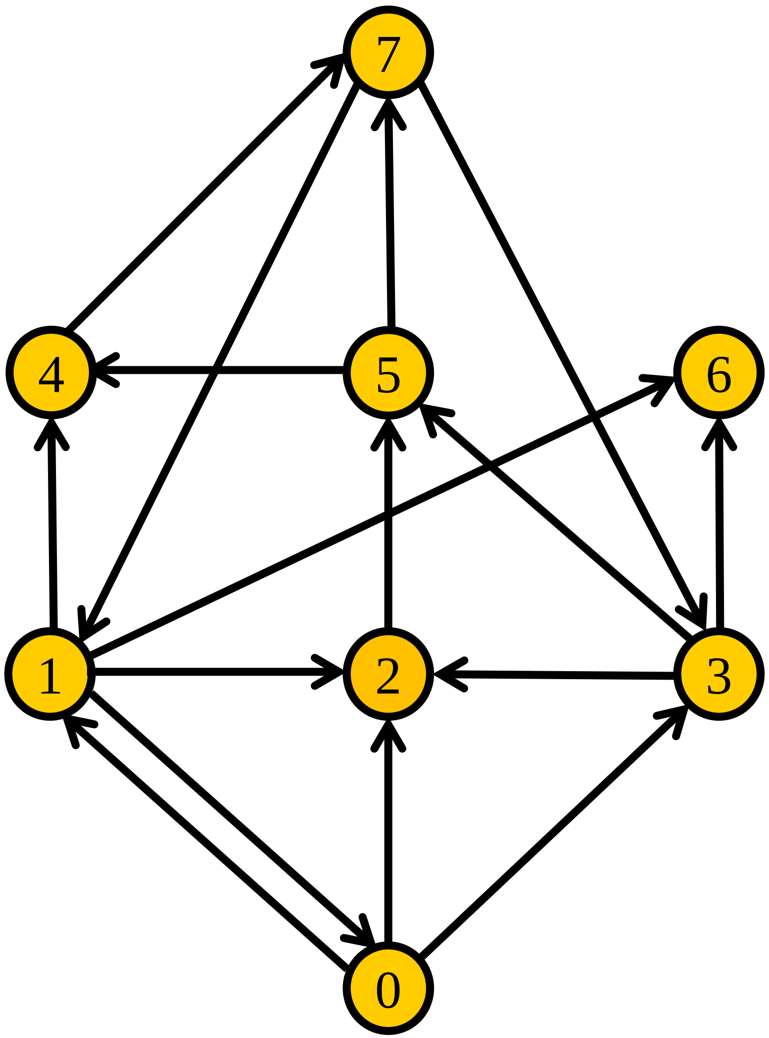
- グラフの幅優先探索
  - 木構造の幅優先探索と同じ
    - 1つの点(値)vを起点とし、v, vの子、vの子の子、…という順序、すなわち、起点に近い順で(全ての)点を調べていく探索
      - 探索しながら、起点からの経路や距離を計算する手続きなども目的に合わせて行う
    - 2つの点を結ぶ複数の経路が存在する可能性があることに注意
      - 木構造では1通りの経路しか存在しない
      - FIFOキューに入れたことがある点は入れない様にする
      - 例えば、点1,2,5,7をこの順序でFIFOキューに入れた後に、7をキューから取り出して1を改めて入れてしまうと無限ループになる



グラフの幅優先探索の定義

- グラフGを起点sから探索する
  - 1. QをFIFOキューとする
  - 2. Qにsを追加
  - 3. Qが空の場合、探索を終了
  - 4. FIFOキューの規則に従ってQから頂点（の名前）vを取り出す
    - Qから頂点vを取り出すことを「vを訪問する」という
  - 5. Qにvの各隣接点（ただし、まだQに入れたことがない点に限る）を追加し、3 に戻る

有向グラフに対する幅優先探索



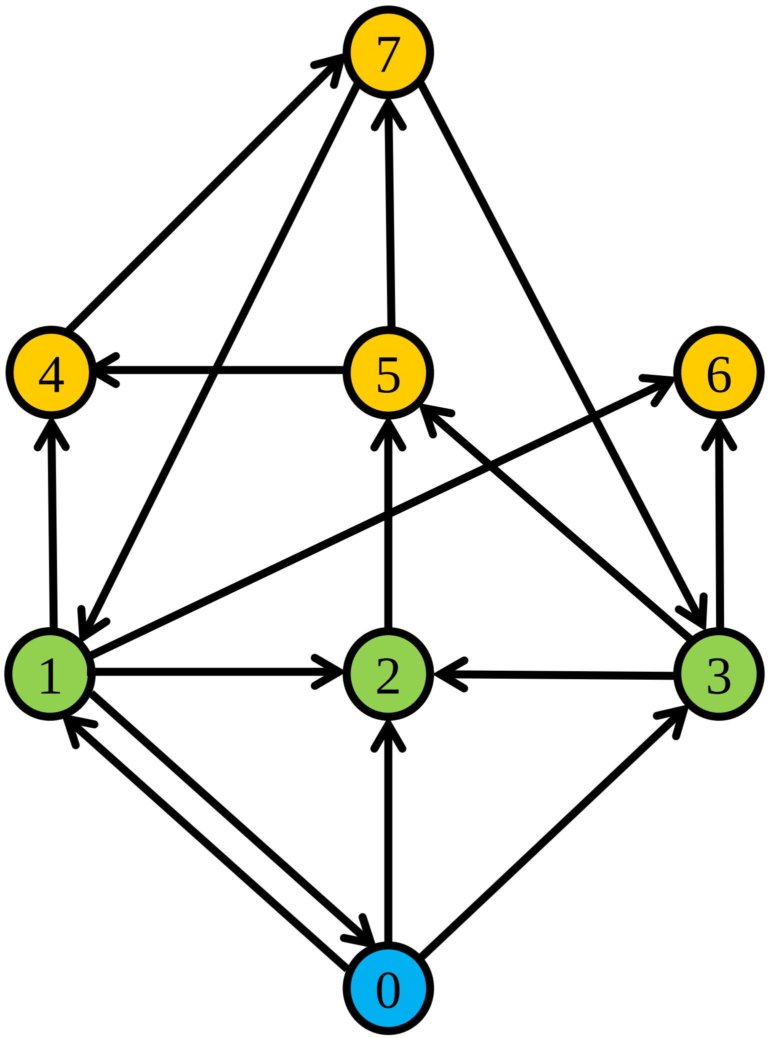
- キュー Q = [3,2,1]

(Step 5: 0の隣接点を追加。追加する順序は自由。ここでは仮に値の大きい順に入れる)

- 訪問順序：0

グラフGを起点sから探索する
1. QをFIFOキューとする
2. Qにsを追加
3. Qが空の場合、探索終了
4. FIFOキューQから頂点vを取り出す
5. Qにvの各隣接点（ただし、まだQに入れたことがない点に限る）を追加、3 に戻る

有向グラフに対する幅優先探索



- キュー Q = [2,1,6,5]

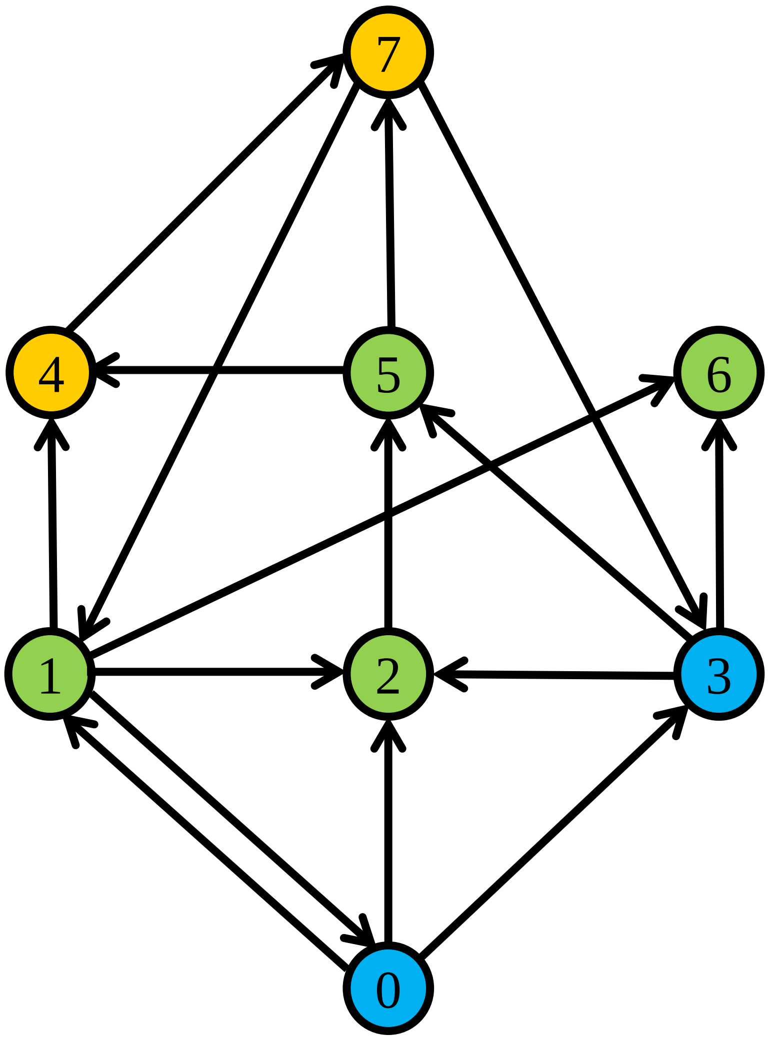
(Step 5：3の隣接点6,5を追加。2は既にQ内にあるので追加しない)

(Step 4：キューQの先頭の値を取り出す)

- 訪問順序：0 3

グラフGを起点sから探索する
1. QをFIFOキューとする
2. Qにsを追加
3. Qが空の場合、探索終了
4. FIFOキューQから頂点vを取り出す
5. Qにvの各隣接点（ただし、まだQに入れたことがない点に限る）を追加、3 に戻る

有向グラフに対する幅優先探索



- キュー Q = [2,1,6,5]

[1,6,5]

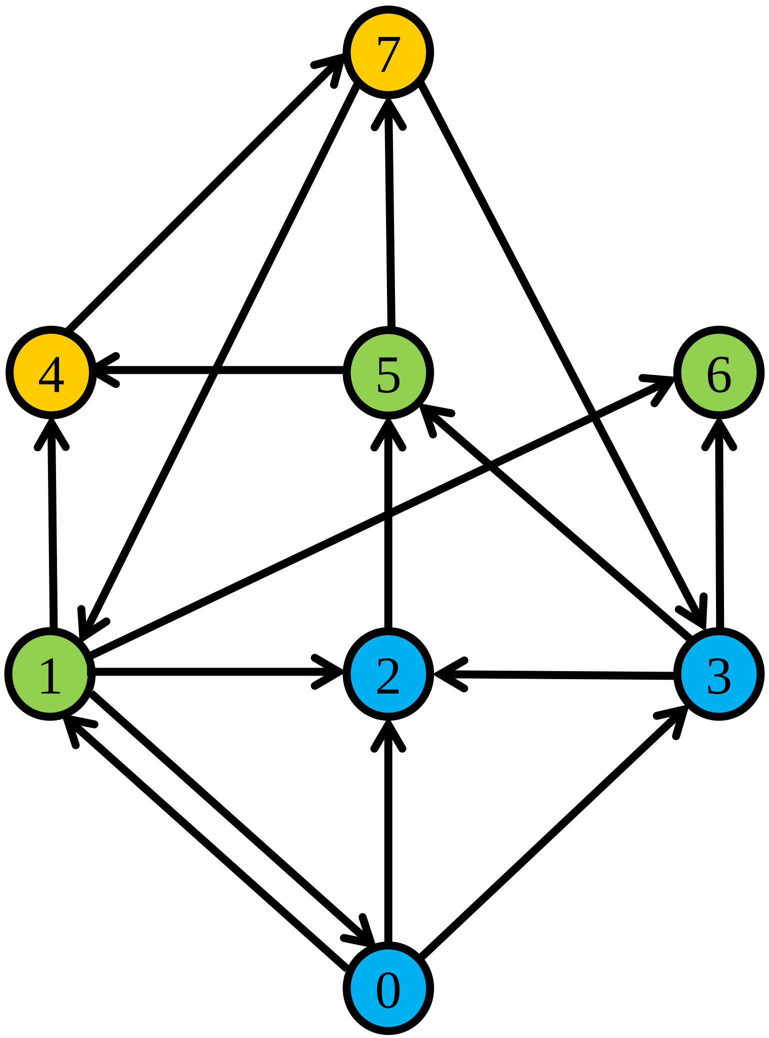
(Step 4: Q(FIFOキュー)の一番先頭の値を取り出す)

(Step 5: 2の隣接点を追加。5は既にQ内にあるので追加しない)

- 訪問順序：0 3 2

グラフGを起点sから探索する
1. QをFIFOキューとする
2. Qにsを追加
3. Qが空の場合、探索終了
4. FIFOキューQから頂点vを取り出す
5. Qにvの各隣接点（ただし、まだQに入れたことがない点に限る）を追加、3 に戻る

有向グラフに対する幅優先探索



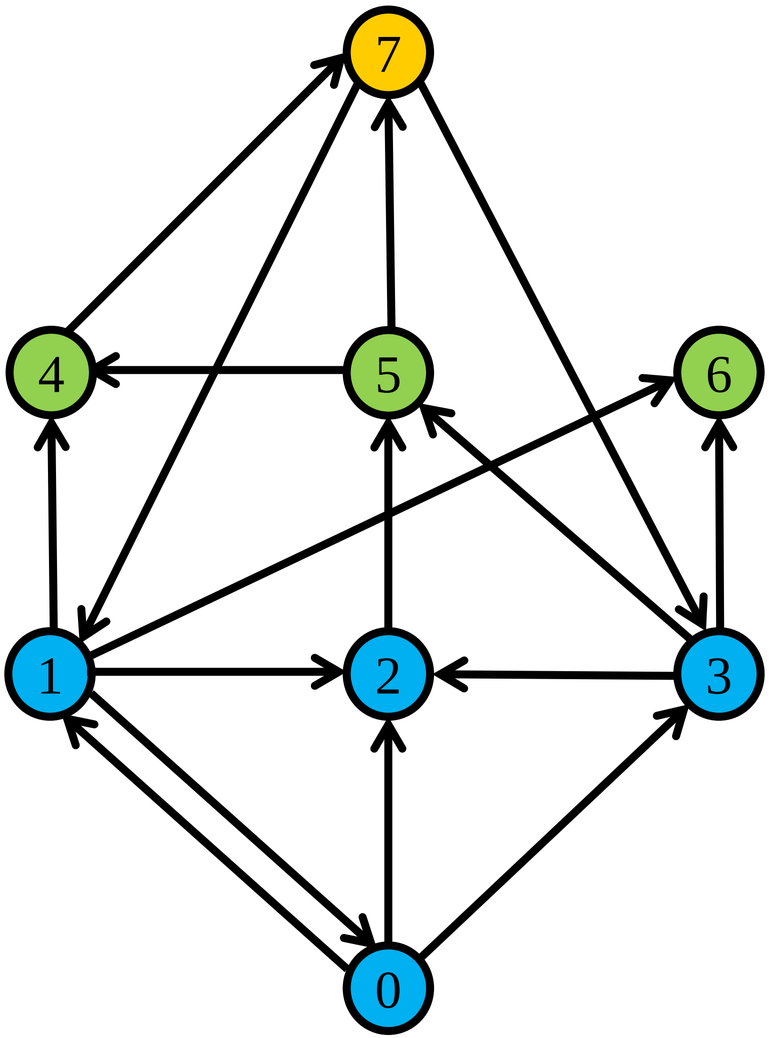
- キュー Q = [6,5,4]

(Step 5: Qに隣接点を追加。0と2は訪問済みか既にQにある)ので追加しない)

- 訪問順序：0 3 2 1

グラフGを起点sから探索する
1. QをFIFOキューとする
2. Qにsを追加
3. Qが空の場合、探索終了
4. FIFOキューQから頂点vを取り出す
5. Qにvの各隣接点（ただし、まだQに入れたことがない点に限る）を追加、3 に戻る

有向グラフに対する幅優先探索



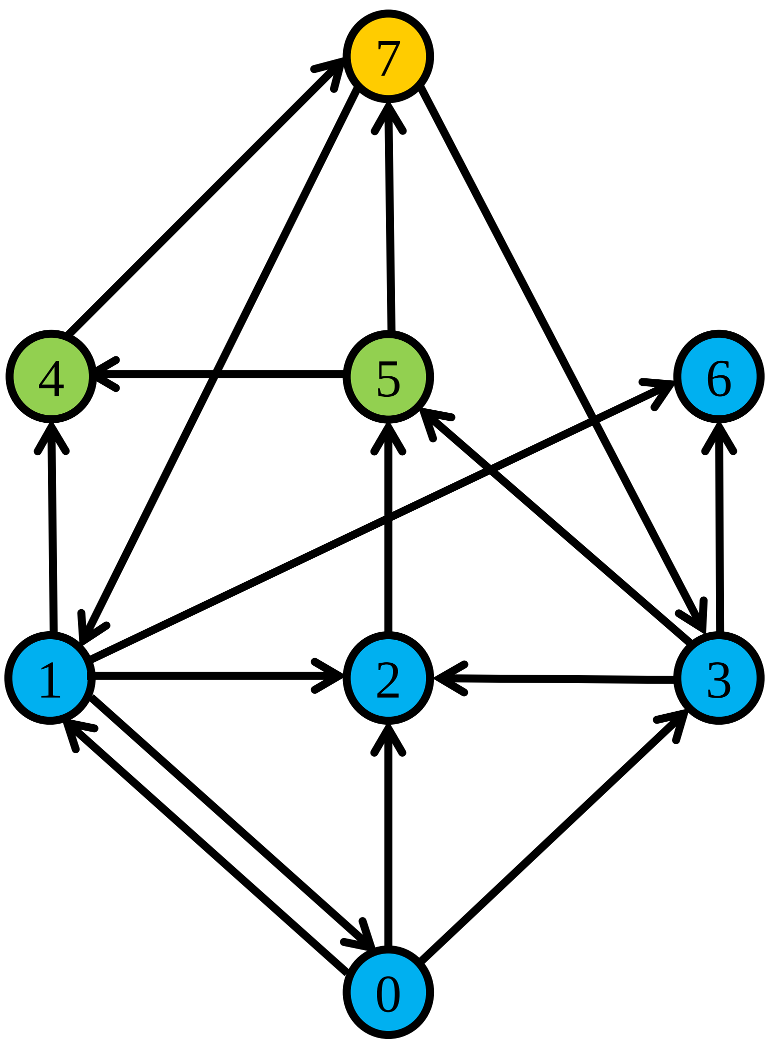
- キュー Q = [6,5,4]

  (Step 4: Qから6を取り出す。FIFOキューなので一番先頭の値を取り出す)

- 訪問順序：0 3 2 1 6

グラフGを起点sから探索する
1. QをFIFOキューとする
2. Qにsを追加
3. Qが空の場合、探索終了
4. FIFOキューQから頂点vを取り出す
5. Qにvの各隣接点（ただし、まだQに入れたことがない点に限る）を追加、3 に戻る

有向グラフに対する幅優先探索



- キュー Q = [4, 7]

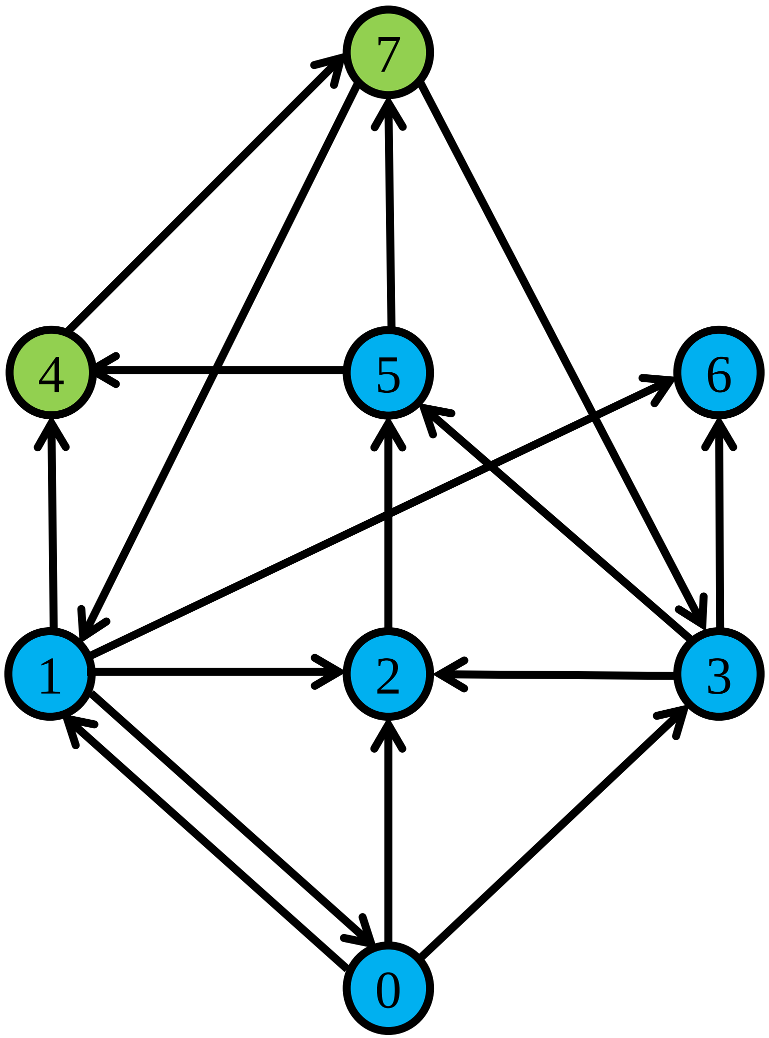
(Step 5: 5の隣接点を追加。4は既にQ内にあるので追加しない)

(Step 4: キューQの一番先頭の値を取り出す)

- 訪問順序：0 3 2 1 6 5

グラフGを起点sから探索する
1. QをFIFOキューとする
2. Qにsを追加
3. Qが空の場合、探索終了
4. FIFOキューQから頂点vを取り出す
5. Qにvの各隣接点（ただし、まだQに入れたことがない点に限る）を追加、3 に戻る

有向グラフに対する幅優先探索



- キュー Q = [4,7]

  [7]

  (Step 3: Qは空ではない)

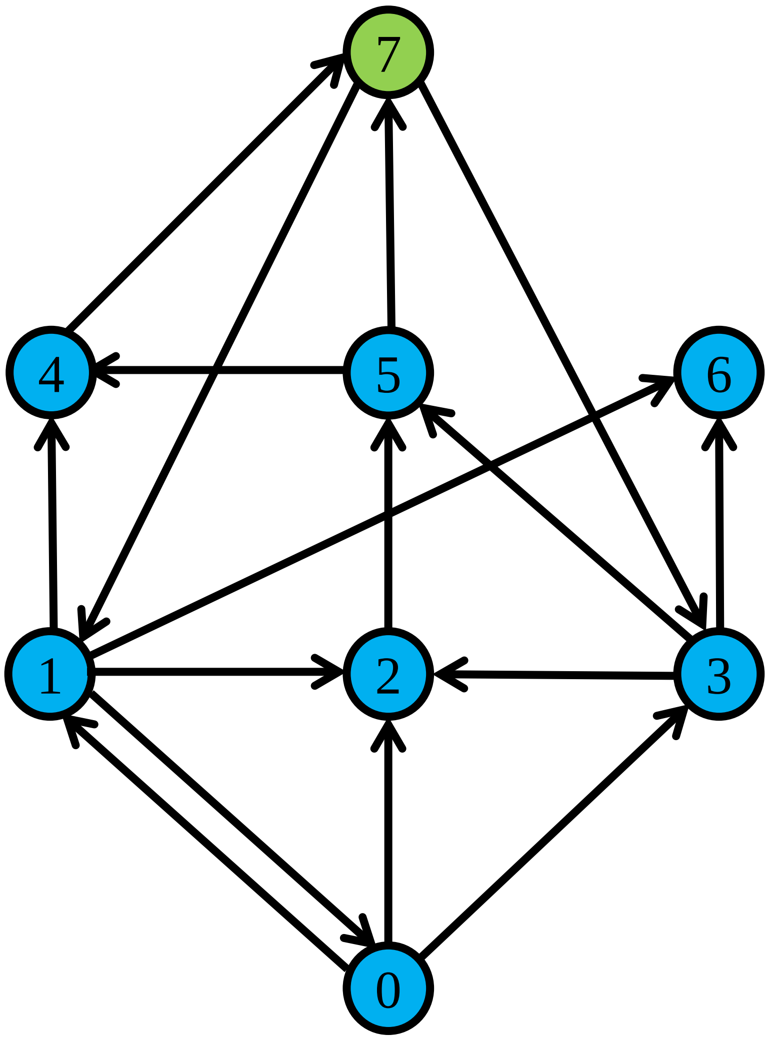
  (Step 4: FIFOキューQの先頭の値を取り出す)

  (Step 5: 4の隣接点7は既にQの中にあるので追加しない)

- 訪問順序：0 3 2 1 6 5 4

グラフGを起点sから探索する
1. QをFIFOキューとする
2. Qにsを追加
3. Qが空の場合、探索終了
4. FIFOキューQから頂点vを取り出す
5. Qにvの各隣接点（ただし、まだQに入れたことがない点に限る）を追加、3 に戻る

有向グラフに対する幅優先探索



- キュー Q = [7]

(Step 5: 7の隣接点を追加。1と3は既に訪問済みなので追加しない)

- 訪問順序：0 3 2 1 6 5 4 7

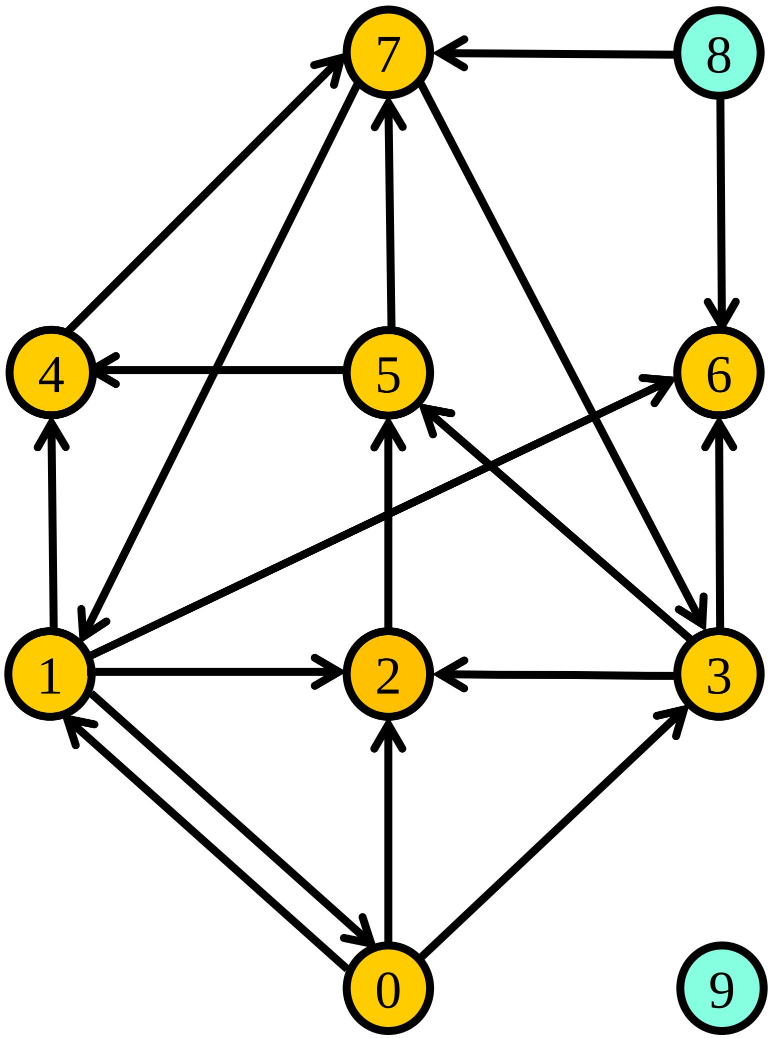
グラフGを起点sから探索する
1. QをFIFOキューとする
2. Qにsを追加
3. Qが空の場合、探索終了
4. FIFOキューQから頂点vを取り出す
5. Qにvの各隣接点（ただし、まだQに入れたことがない点に限る）を追加、3 に戻る

グラフの幅優先探索の実装

- Pythonで実装する場合:
  - (データ構造):
    - グラフ(隣接リスト):多重配列＝リスト
      - 隣接行列との相性は？
    - 幅優先探索:FIFOキュー＝リスト
  - (アルゴリズム)幅優先探索:
    - FIFOキューで点を管理
    - 目的の点を見つけるまで起点から順に点を調べる
      - 全ての点を調べる

```
#グラフの幅優先探索の典型的実装
# list_adjlist=探索するグラフGの隣接リスト
# stnode=探索を開始するGの頂点（起点）
def GraphSearch(list_adjlist, stnode):
    FIFOキューQに起点stnodeを入れる
    # FIFOキューQが空になるまで探索（ループ）
    while(Qが空ではない):
        FIFOキューQから点node1を取り出す＝node1を訪問
        node1に隣接する全ての点（ただし、まだQに入れたことがない点に限る）を全てQに入れる
```

有向グラフに対する幅優先探索

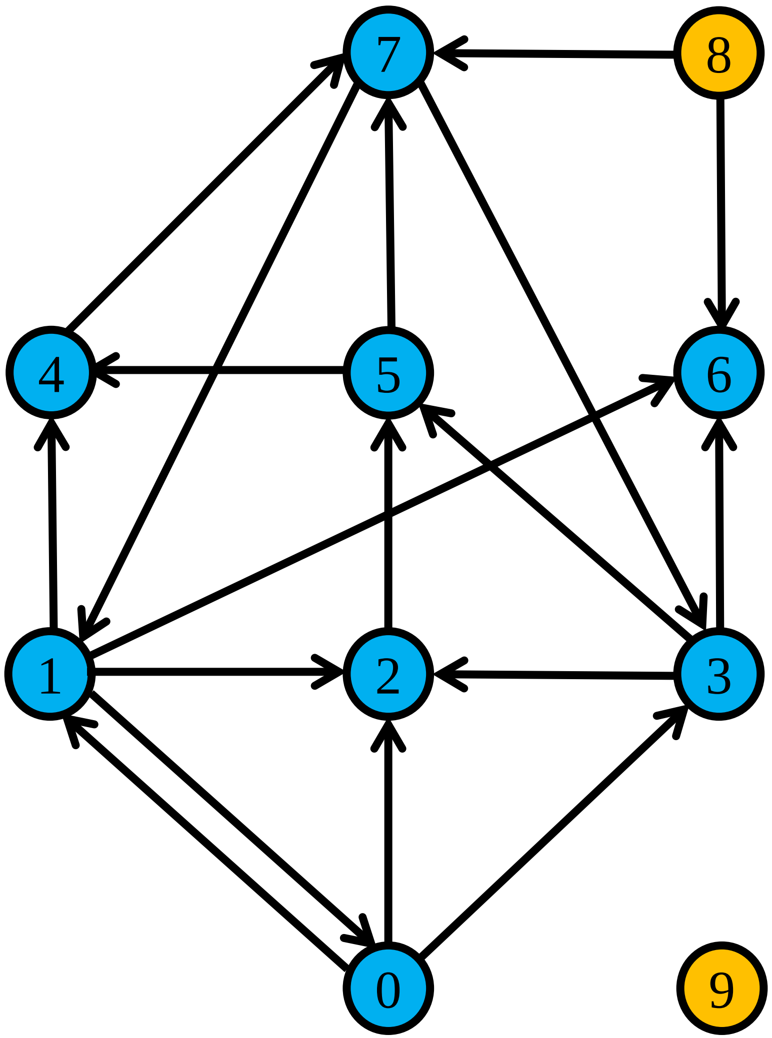


- 訪問できない点があるかも？

グラフの幅優先探索の定義

- グラフGを起点sから探索する
  - 1. QをFIFOキューとする
  - 2. Qにsを追加
  - 3. Qが空で全ての点を訪問済みならば、探索終了。Qが空で訪問していない点があるならば、その点を新たな起点sとして2に戻る
  - 4. FIFOキューの規則に従ってQから頂点(の名前)vを取り出す
    - Qから頂点vを取り出すことを「vを訪問する」という
  - 5. Qにvの各子供(ただし、まだQに入れたことがない点に限る)を追加し、3 に戻る

有向グラフに対する幅優先探索

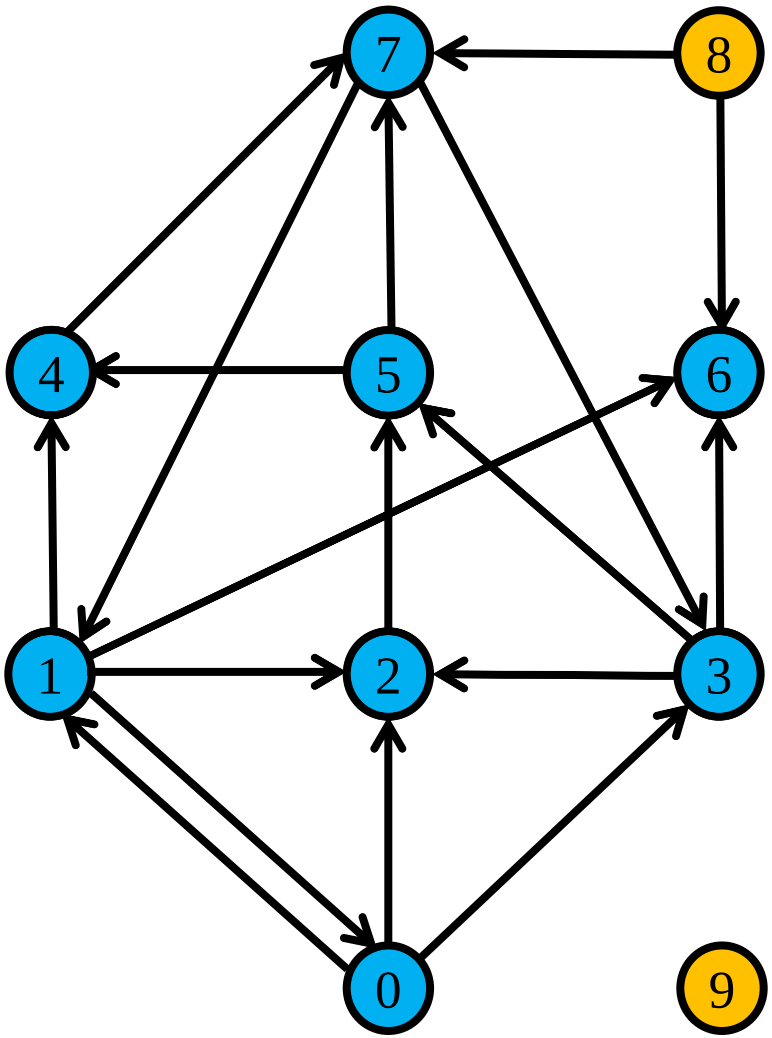


- キュー Q = []

  (Step 3: Qは空だが、訪問していない頂点があるので、Step 1へ)

- 訪問順序：0 3 2 1 6 5 4 7

有向グラフに対する幅優先探索

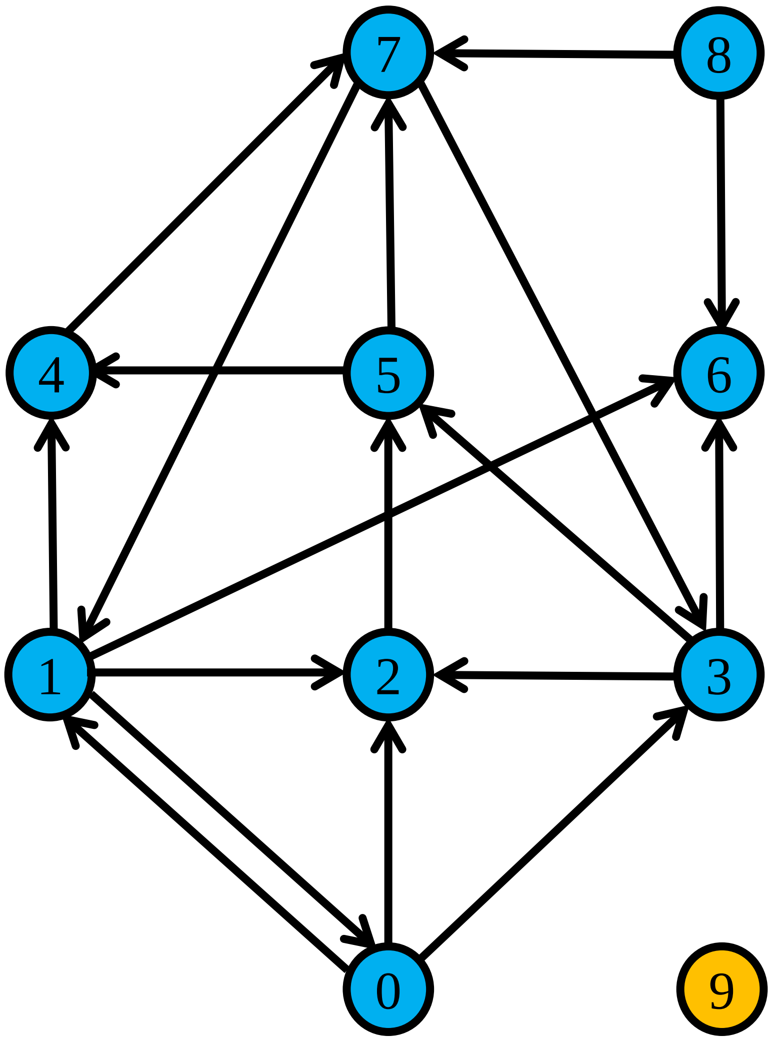


- キュー Q = [8]

(Step 2: 未訪問の点を選択。どの未訪問の点を選んでも良いが、一番小さい数の点を選ぶことにする)

- 訪問順序：0 3 2 1 6 5 4 7 8

有向グラフに対する幅優先探索

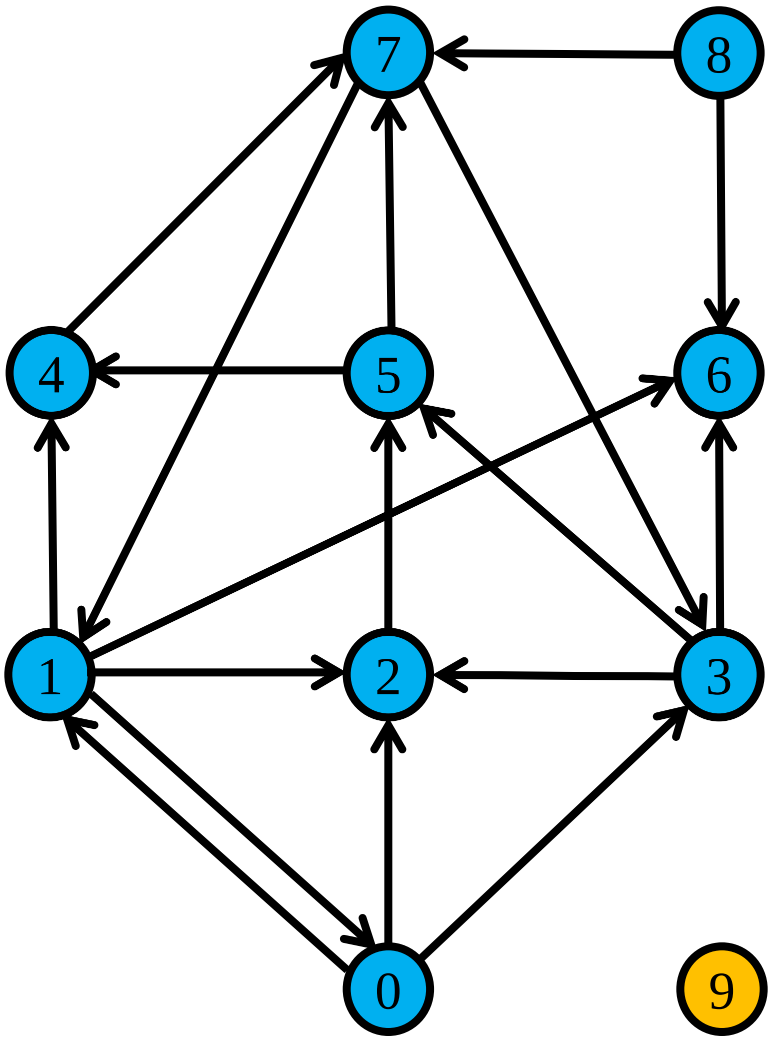


- キュー Q = []

  (Step 3: Qは空だが、訪問していない頂点があるので、Step 1へ)

- 訪問順序：0 3 2 1 6 5 4 7 8

有向グラフに対する幅優先探索

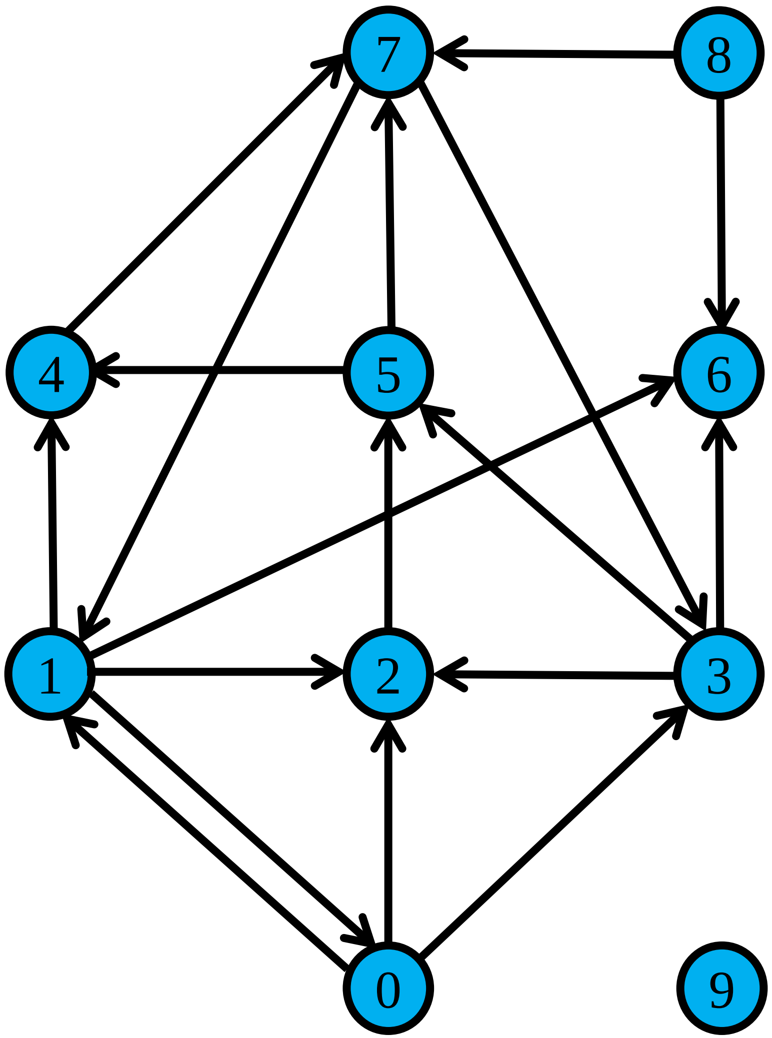


- キュー Q = [9]

  (Step 2: 9 … 選択)

- 訪問順序：0 3 2 1 6 5 4 7 8 9

有向グラフに対する幅優先探索



- キュー Q = []

  (Step 3: Qは空であり、全ての点を訪問したので探索を終了)

- 訪問順序：0 3 2 1 6 5 4 7 8 9

グラフの2点間の最短距離

- 点Aと点Bの最短距離はAを起点として幅優先探索を実行したときのBまでの距離で求まる
  - 木構造で幅優先探索を実行した場合と同じ
  - 幅優先探索では、常にFIFOキューの中で起点に最も近い点をFIFOキューから取り出す
  - 点Aから点Bまでの距離がxであり、BをFIFOキューから取り出したとき、x-1以下の距離の点は全て取り出し終えている
  - AからBまで距離がx-1以下の経路は存在しない

グラフに対する幅優先探索の計算量

- 頂点数が n 個、枝数が m 個の場合:
  - 全ての頂点を訪問する場合（最悪の場合） $O(n+m)$
  - 木構造では、$n = m+1$ が成立していた
  - 有向グラフ/無向グラフどちらでも同じ

課題

- グラフ構造を体験しましょう
  - 本日（30日）の夜にアップロードします
  - basic4.ipynb：基礎課題（2日締め切り）
  - ex4.ipynb：本課題（7日締め切り）
    - 提出先を間違えない様にして下さい
    - 配布しているファイル（.ipynbのファイル）をそのまま提出して下さい