人工知能基礎
第6回　グラフ

ソフトウェア情報学部
David Ramamonjisoa

目次

- グラフの定義
- グラフの種類
- グラフの特徴
- 例題
- 需要なグラフ
- まとめ

言葉の説明

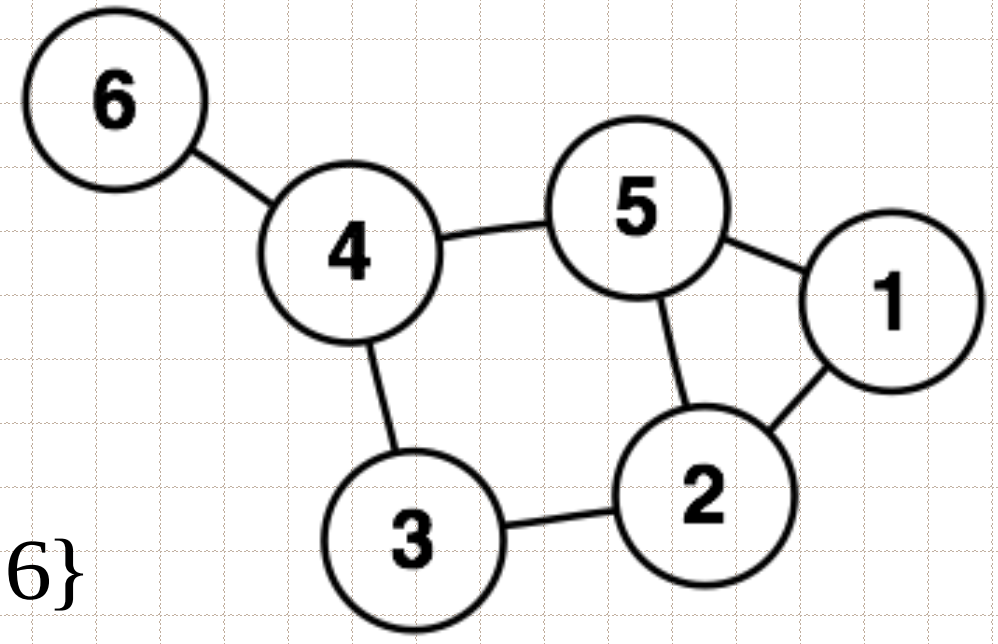
- 数学で関数のグラフ
  - 解析学の時、関数ということはある変数に依存して決まる値あるいはその対応を表す式の事であった：y = f(x)
  - 関数f(x)は平面内の曲線もしくは空間内の曲面としてダイアグラム状に視覚化したものである
- 数学でグラフ理論
  - ネットワーク的な構造のようなものを頂点(ノード)、辺(エッジ、枝)、などで構成される。
- 省略として、両方ともグラフという

グラフの定義(1)

- ある問題を点と線を用いた図によって表現するとき、その図はグラフ(graph)と呼ばれる。
- グラフGは、ノードの有限集合V ={ノード/点/vertex/vertices}、およびエッジの有限集合E={エッジ/辺/線/edge/edges}からなる組(V,E)で定義される。すなわち、G=(V,E)で表現される。
- エッジはノードのペアであり、順序のない組(u,v)または順序がある組<u,v>で表す。このとき、エッジが前者で表されるグラフを無向グラフ、後者を有向グラフという。

グラフの定義(2)

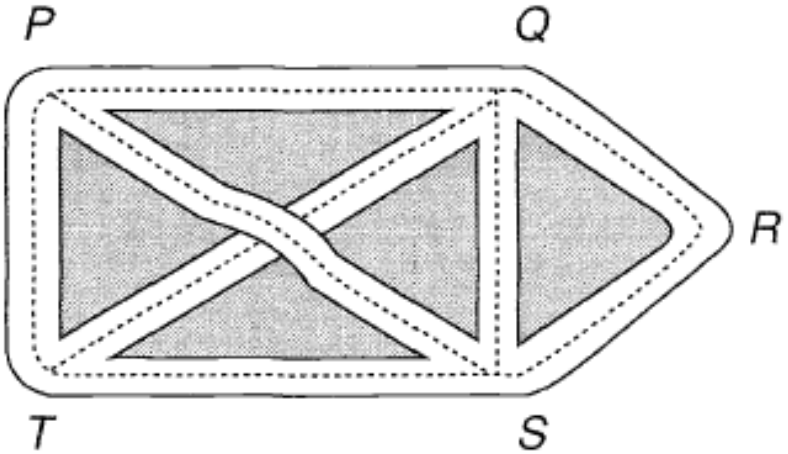
- 6 つのノードと 7 つのエッジから成るグラフの一例

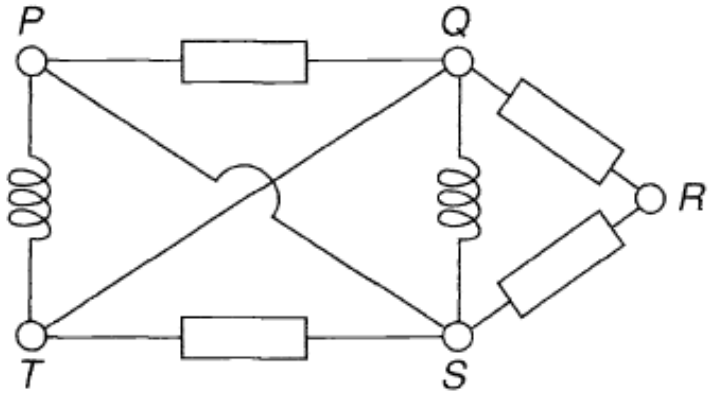


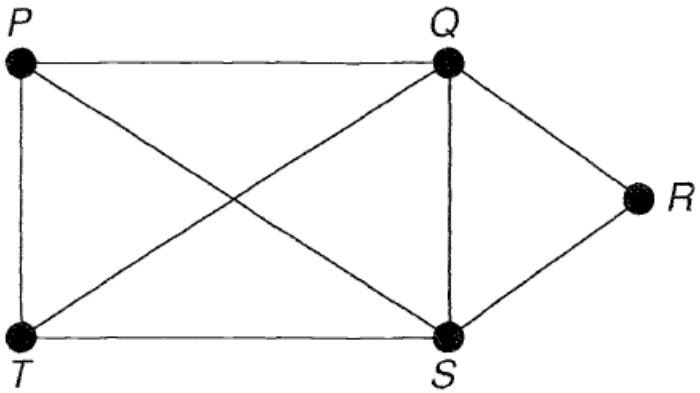
$V := \{1,2,3,4,5,6\}$

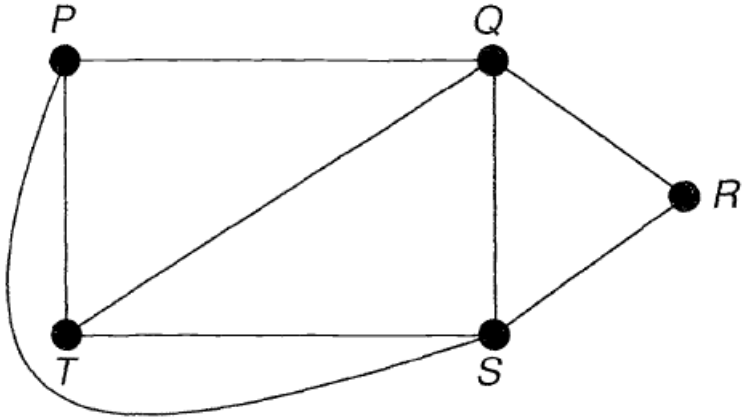
$E := \{(1,2),(1,5),(2,3),(2,5),(3,4),(4,5),(4,6)\}$

グラフの定義(3):4つのグラフは同じ









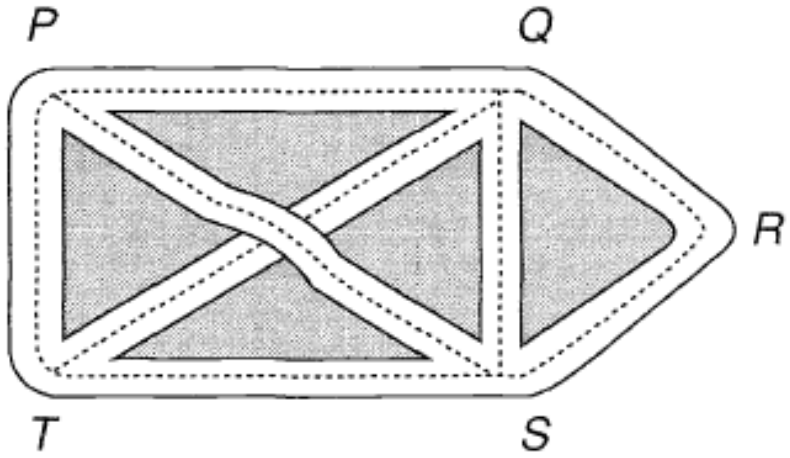
$$V := \{P, Q, R, S, T\}$$

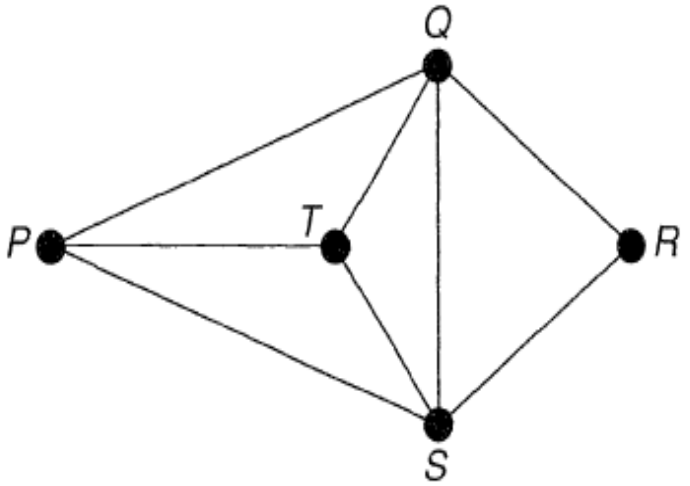
$$E := \{(P,Q),(P,S),(P,T),(T,Q),(T,S),(S,Q),$$

$$(S,R),(Q,R)\}$$

点の次数はその端点とする辺の本数

グラフの定義(3):2つのグラフは同じ





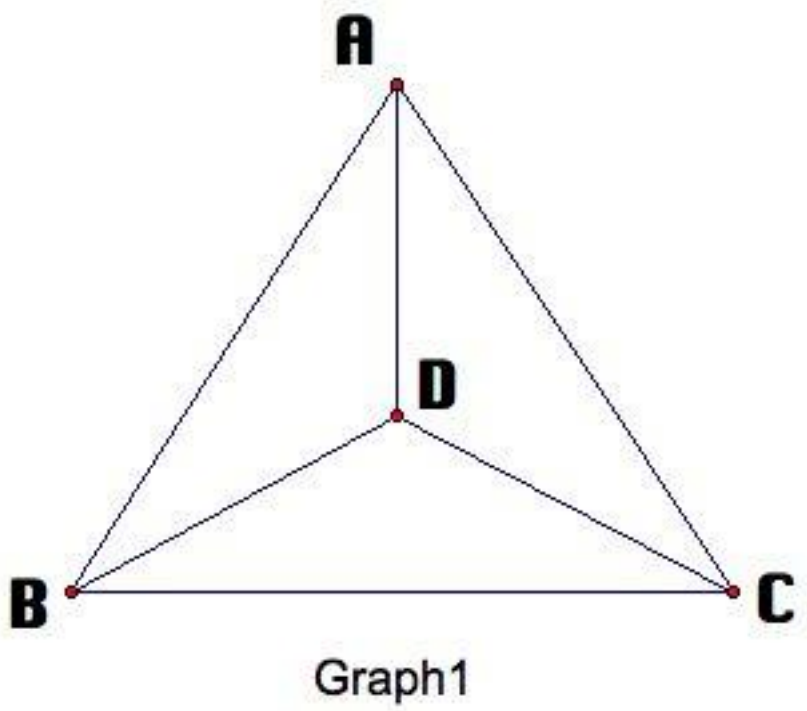
$$V := \{P, Q, R, S, T\}$$

$$E := \{(P,Q),(P,S),(P,T),(T,Q),(T,S),(S,Q),$$

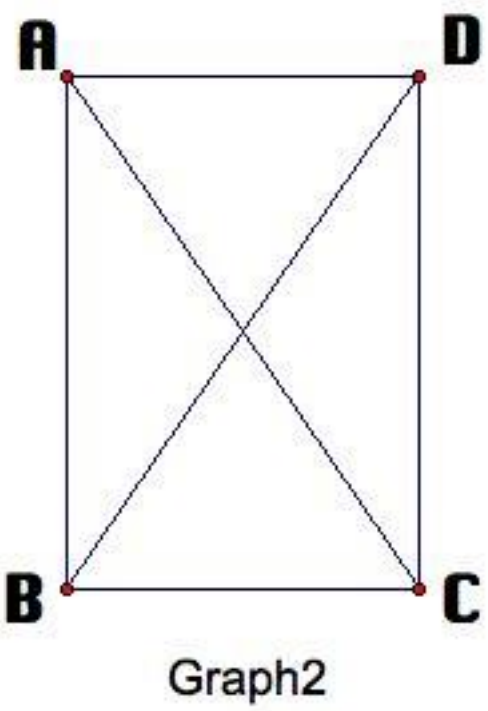
$$(S,R),(Q,R)\}$$

点の次数はその端点とする辺の本数

グラフの定義(3):2つのグラフは同じ



Graph1



Graph2

$$V := \{A, B, C, D\}$$

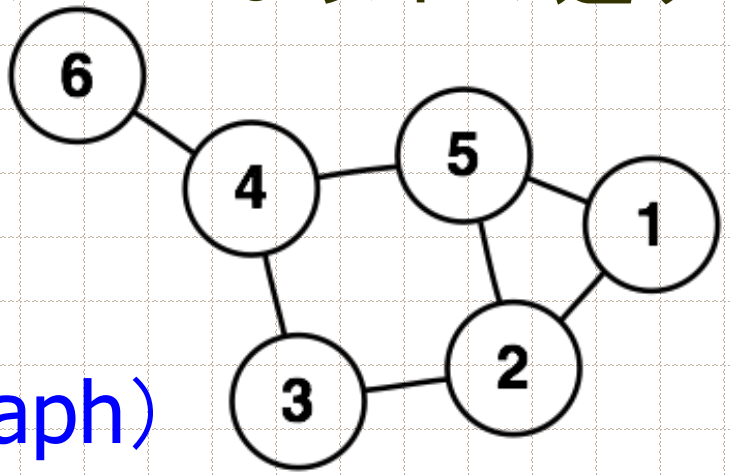
$$E := \{(A, B), (A, C), (A, D), (B, C), (B, D), (C, D)\}$$

グラフの特徴

- 辺 e の両端の点を**端点**といい、端点は 辺eに**接合**しているという
- 辺と辺がある頂点を共有しているとき、その辺同士は**隣接（adjancent）**しているという
- 2 頂点間に複数の辺があるとき、**多重辺 (multiple edges)**という
- ループや多重辺を含むグラフのことを**多重グラフ**という
- 無向グラフで、全ての頂点間にエッジがあるのグラフは完全グラフ(**complete graph**)という

グラフのパス(path)

- ある頂点から別の頂点へいたる連続エッジのことをパス(path)と呼ぶ。
- 例：ノード6からノード2までのパスは以下の通り
  - 6 – 4 – 5 – 2
  - 6 – 4 – 3 – 2
  - 6 – 4 – 5 – 1 – 2



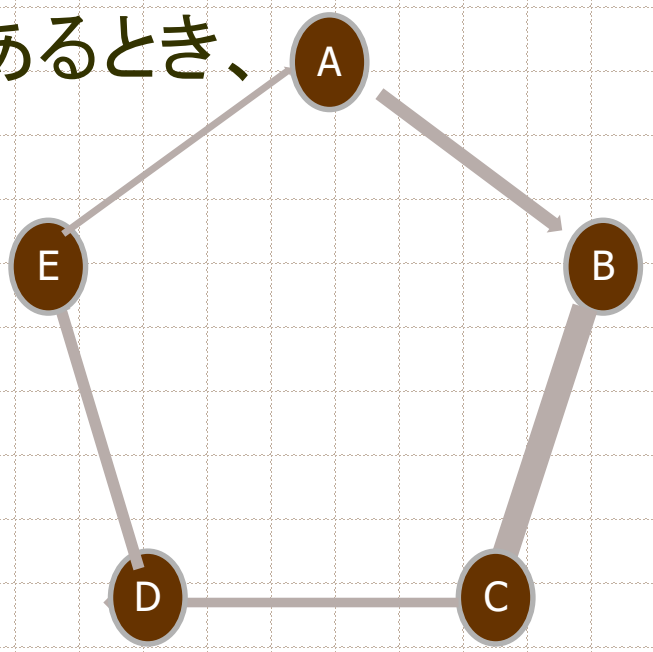
- 連結グラフ（connected graph）

各ノードから他の全てのノードへのパスが1つ以上あるグラフ

- 非連結グラフ（disconnected graph）

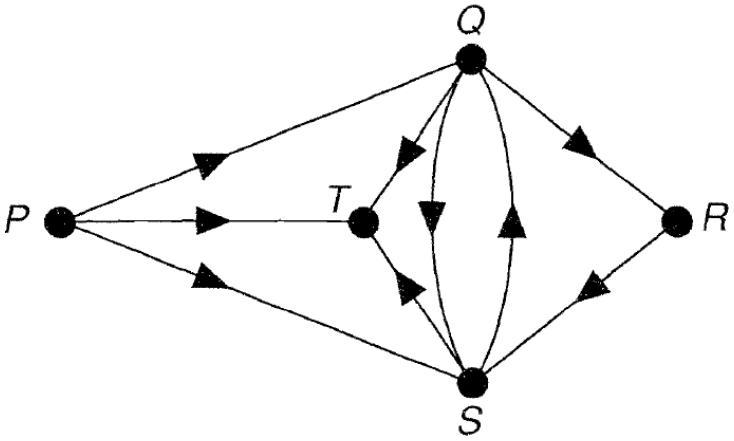
閉路グラフ（cycle graph）

- ループや複数のエッジが連なって1つの輪になっており、1つのサイクルからなっているグラフ
  - 閉路グラフではn個のノードがあるとき、エッジの数もn個
  - ループ：ある辺の両端点が等しいとき、ループ（自己ループ）という



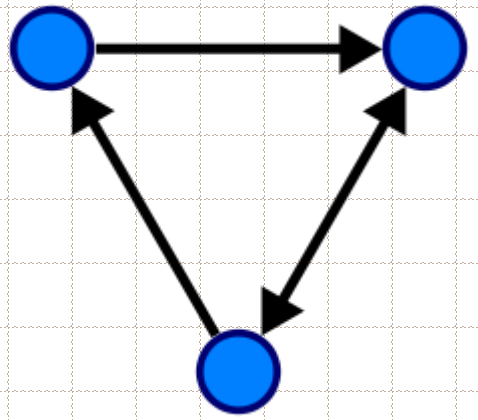
グラフの種類

- 有効グラフ
- 無効グラフ
- 単純なグラフ
- 重み付きグラフ
- ループ



有向グラフ(directedgraph:digraph)

- 集合V,Eと、Eの元に、二つのVを元の対で対応させる写像



$$f : E \to V \times V$$

$$G := (f, E, V)$$

を有効グラフという。V の元を G の**頂点** (vertex) または**ノード**、E の元を G の**辺**（**か枝**） (edge) または**エッジ**と呼ぶ。

英語でdigraphと呼ぶ。

無効グラフ

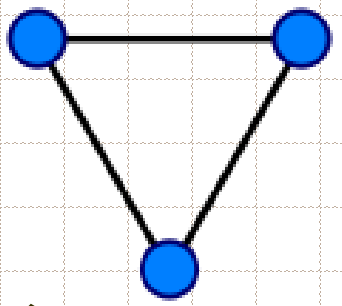
- P(V) を V の幂集合とする。E の元に V の部分集合を対応させる写像があって、E の任意の元 e の像が $g(e) = \{v_1, v_2\}$ のようにちょうど二つの元の集合になっているとする。このとき、三つ組

$$G := (g, V, E)$$

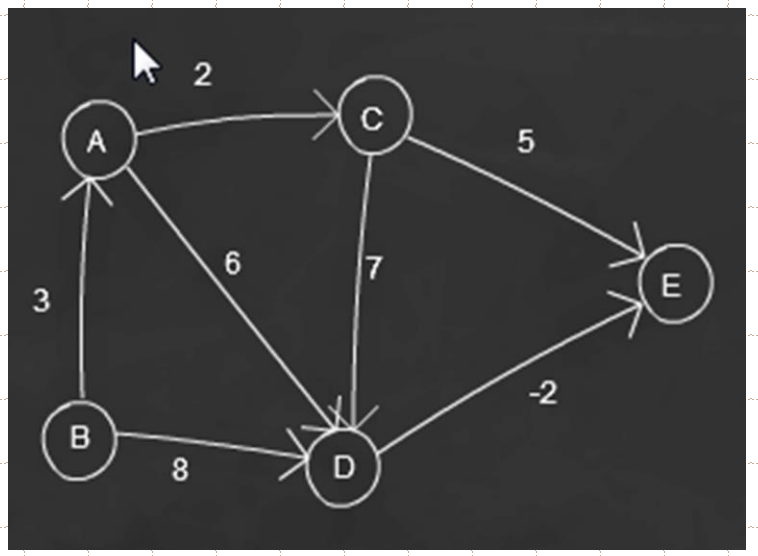
を**無向グラフ**という。V の元を G の頂点、E の元を G の辺と呼ぶ。

単純なグラフと重み付きグラフ

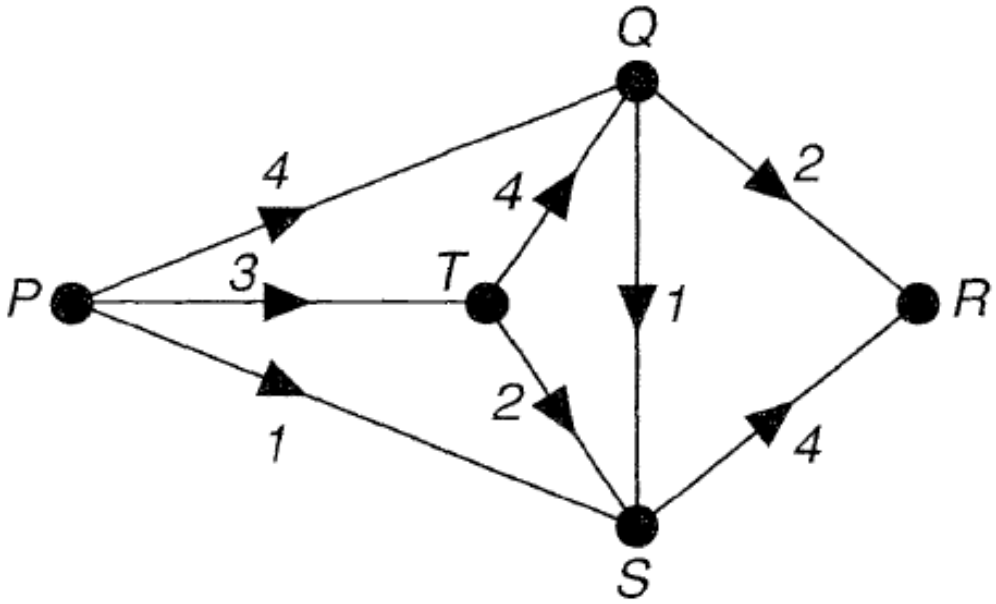
- 単純なグラフ



- 重み付きグラフ：辺に**重み**（**コスト**）が付いていることがある。このようなグラフは、**重み付きグラフ**と呼ばれる



重み付きグラフの集合

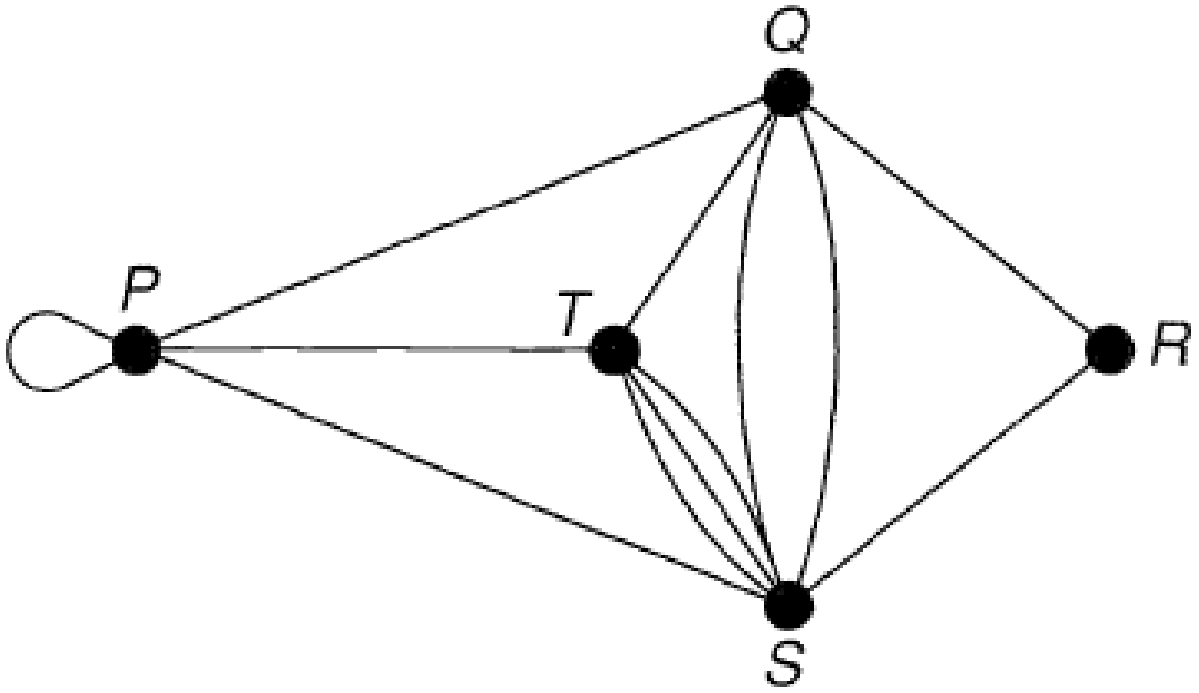


$$V := \{P, Q, R, S, T\}$$

$$E := \{\{\langle P, Q\rangle, 4\}, \{\langle P, S\rangle, 1\}, \{\langle P, T\rangle, 3\}, \{\langle T, Q\rangle, 4\},$$

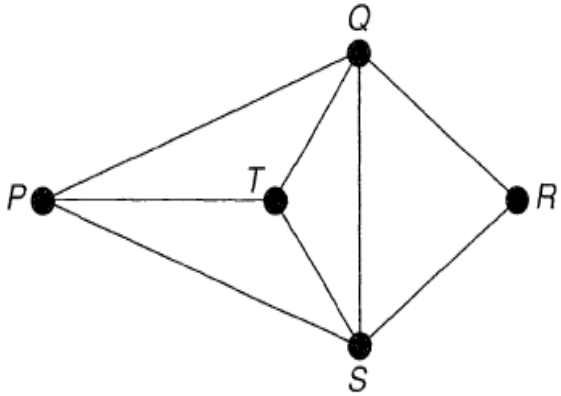
$$\{\langle T, S\rangle, 2\}, \{\langle Q, S\rangle, 1\}, \{\langle S, R\rangle, 4\}, \{\langle Q, R\rangle, 2\}\}$$

その他：単純ではないグラフ、多重辺グラフ、ループ



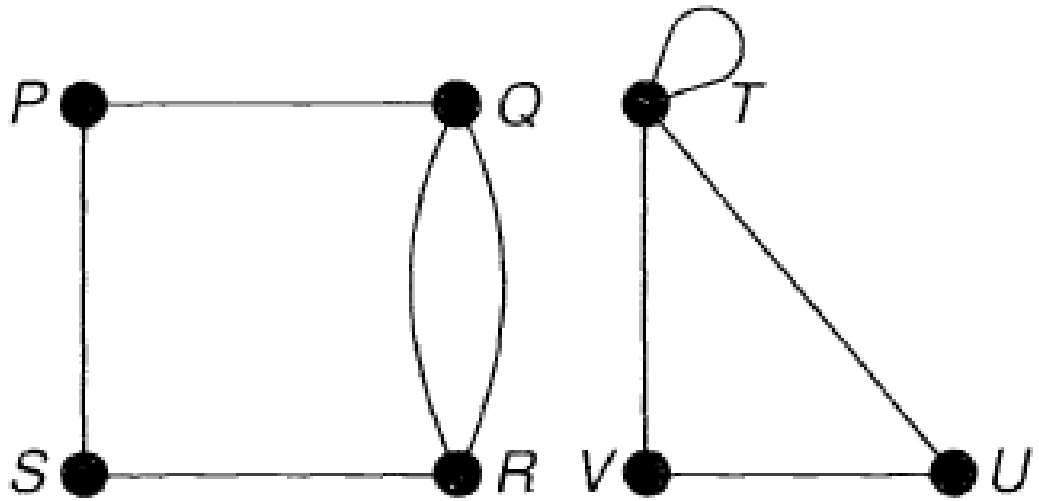
歩道(ワーク：walk)

- ある点から別の点からの行き方
- P-Q-R(パス)は長さ2の歩道である
- P-S-Q-T-S-Rは長さ5の歩道である
- P-T-S-R (パス)は長さ3の歩道である
- Q-S-T-Q (閉路)は長さ3の歩道である



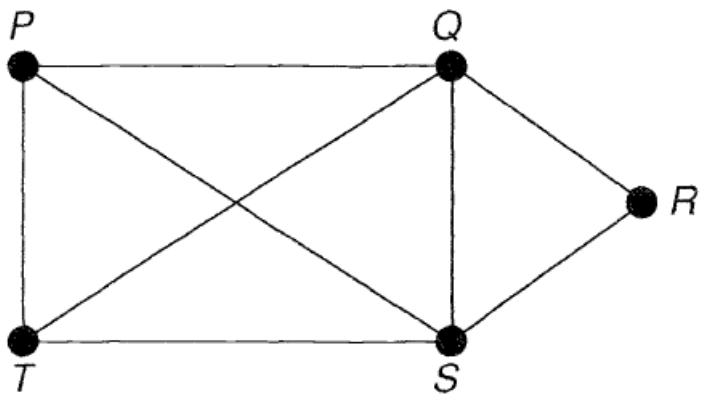
連結グラフと非連結グラフ

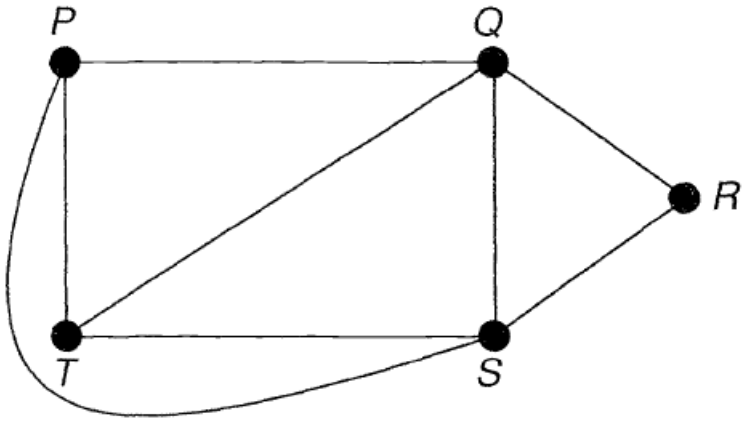
- あるグラフがどの2つの点も道で結ばれているようなグラフは**連結グラフ**と呼ばれる
- 2つ以上のかたまりからなるグラフは**非連結グラフ**と呼ばれる（以下の図）



平面的グラフ(planar graph)

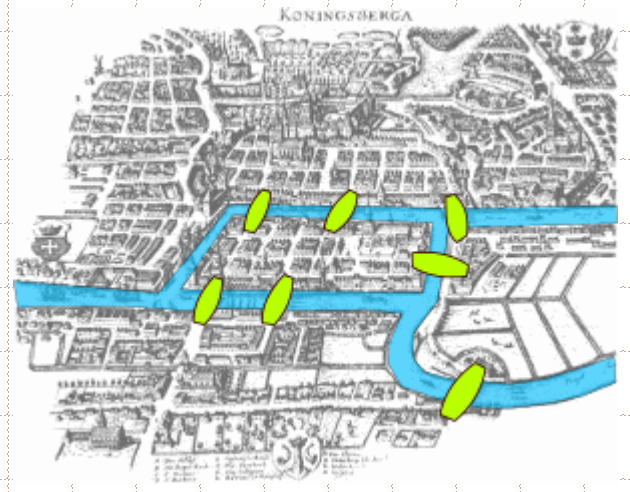
- 交差がないように描き直せるグラフは**平面的グラフ**と呼ばれる。

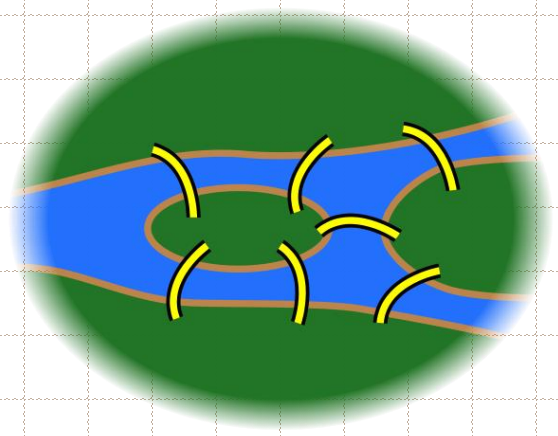


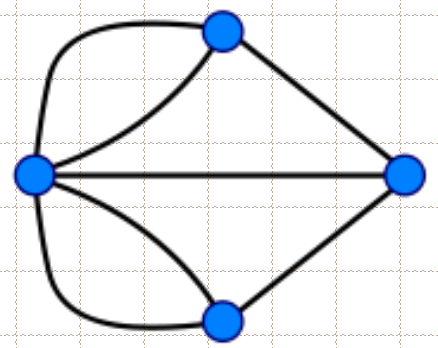


グラフ理論の起源

- グラフ理論は、1736年、「ケーニヒスベルクの問題」に対してオイラーが解法を示したのが起源とされる。
- 7橋の問題

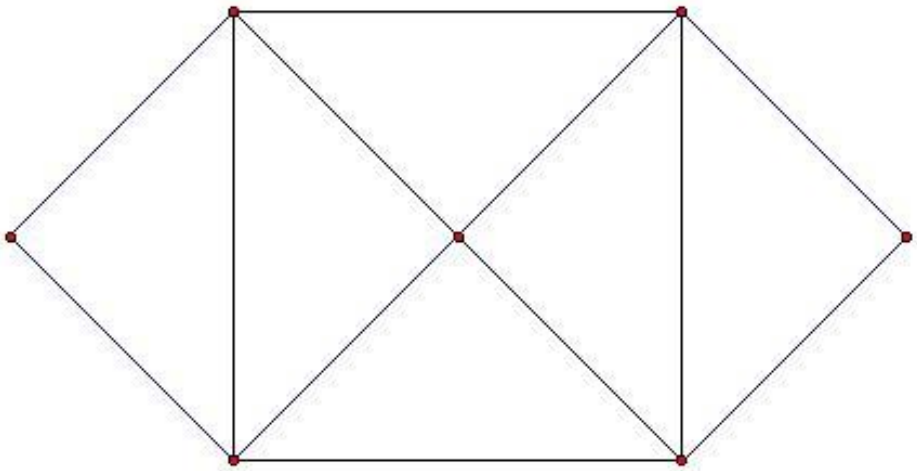




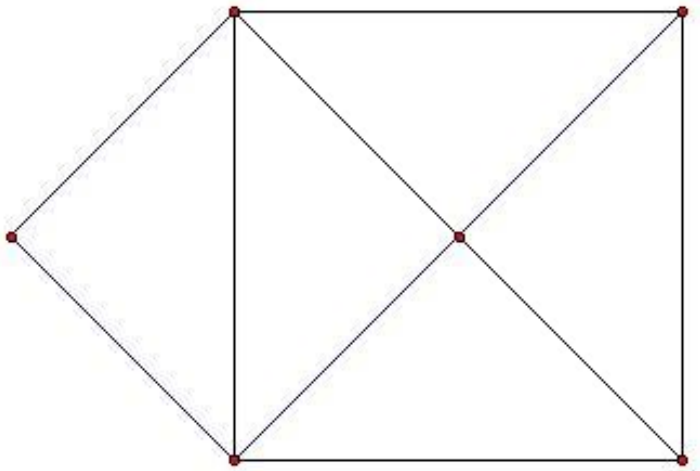


オイラー・グラフ

- 連結グラフGのすべての辺を含む閉じた小道はオイラー・グラフと呼ばれる。
- グラフGraph1はオイラー・グラフである。グラフGraph2は半オイラー・グラフである。



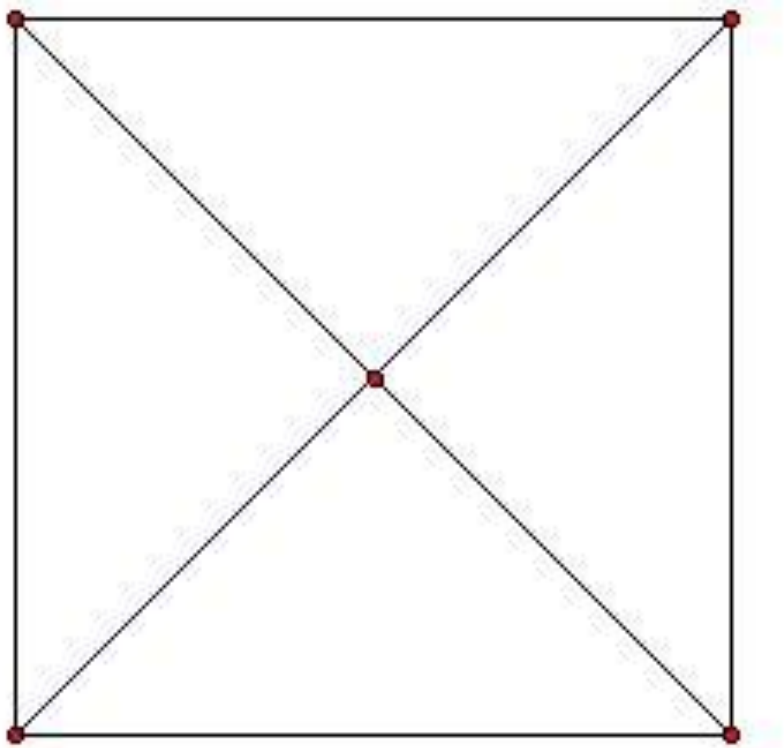
Graph 1



Graph 2

オイラー・グラフではない、

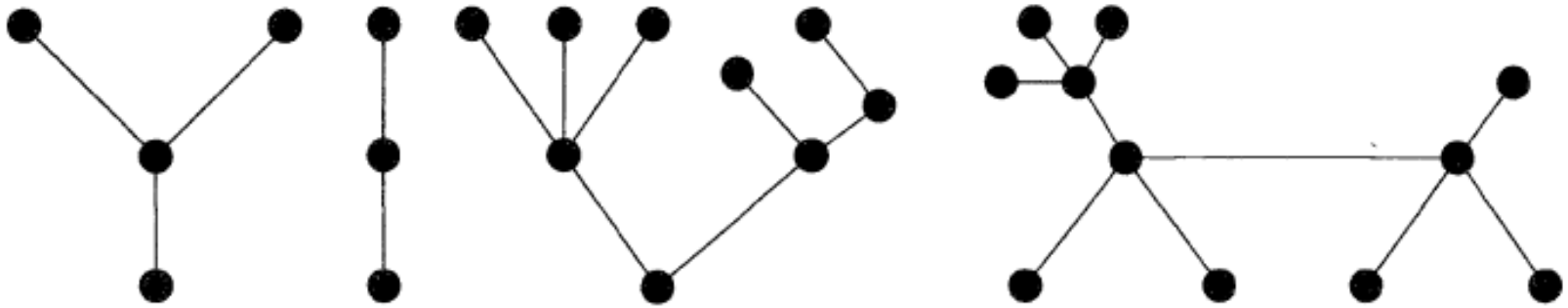
- グラフGraph3はオイラー・グラフではない。



Graph 3

木

- 閉路を含まないグラフを林(forest)と定義し、連結な林を木(tree)と呼ぶ。
- 木と林は単純なグラフである

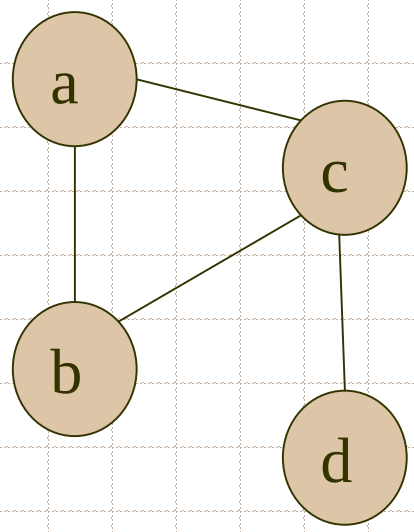


グラフのデータ構造

- 2種類のデータ構造がある
  - 第一は隣接リスト(adjacent lists)と呼ばれるもので、各ノード毎に隣接するノードのリストを保持するデータ構造である。
  - 第二は隣接行列(adjacent matrix)と呼ばれるもので、行と列にエッジの始点と終点となるノードが並んだ2次元の配列で表され、配列の各要素は2つのノード間にエッジがあるかどうかを示す値が格納される。

隣接リスト(adjacent list)

- グラフのノードとエッジをリスト形式で表したもの



a → b c
b → a c
c → a b d
d → c

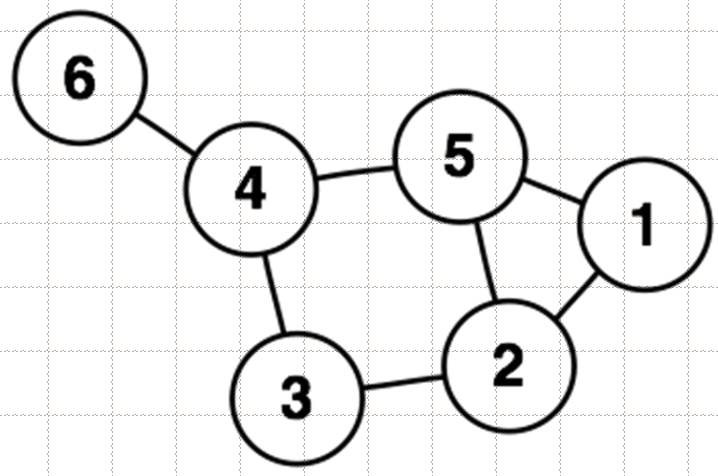
隣接リスト

無効グラフ隣接リスト(Python)

```
# 辞書型を用いて隣接リストの実装
from collections import defaultdict
#グラフのデータ構造を作るルーチン（プログラム関数）
def graph_adj_list(Edges):
    E = len(Edges) # エッジの数をEに代入
    adj_list = defaultdict(list)  #辞書の変数を初期化
    for i in range(E): # 繰り返しループの０からエッジの数まで
        u, v = Edges[i] # u,v = (1,2) <=> u=1;v=2同時に代入
        adj_list[u].append(v) # 無向グラフなのでvはuに隣接している
        adj_list[v].append(u) # uはvに隣接している
        # 辞書に追加される、次の要素を追加
    return adj_list # 辞書を戻り値として返す

## MAIN PROGRAM ##
# edges変数はそれぞれのエッジ(タプル型)のリストを格納される
edges = [(1,2),(1,5),(2,3),(2,5),(3,4),(4,5),(4,6)]
# 上に定義された関数を呼び出し、引数としてエッジのリスト
graph = graph_adj_list(edges)
# グラフを出力する
print "Graph of %d Nodes and %d Edges."%(len(graph),len(edges))
print "Adjacent list:"
#print graph
print "{"
for k in graph:
    print  "\t",k,":",graph[k],","
print "}"
```

無効グラフ隣接リスト(Python)

```
Graph of 6 Nodes and 7 Edges.
Adjacent list:
{
        1 : [2, 5] ,
        2 : [1, 3, 5] ,
        3 : [2, 4] ,
        4 : [3, 5, 6] ,
        5 : [1, 2, 4] ,
        6 : [4] ,
}
```

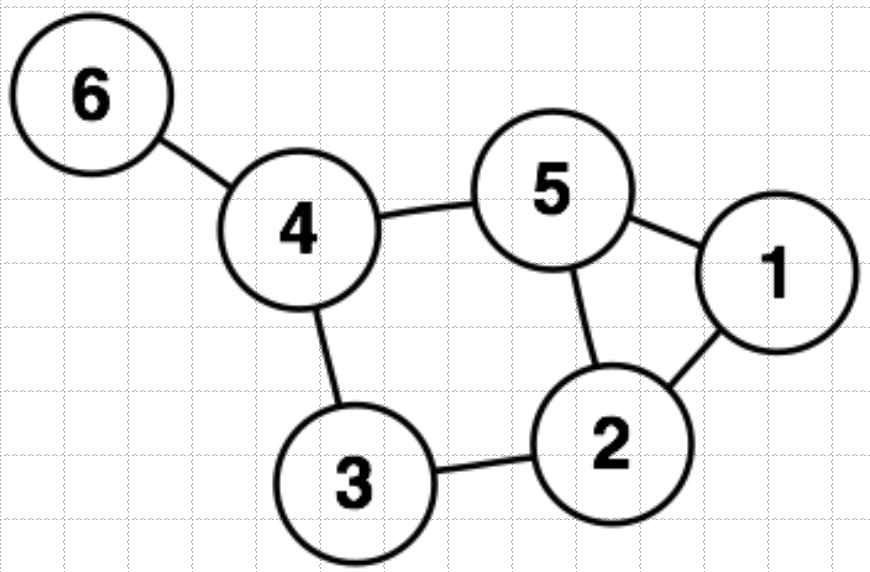


Pythonでは隣接リストのデータ構造は辞書型を用いて実装する。
辞書型はキー（ノード名）と値（リスト型：隣接ノード）を{}(括弧ブレス)にデータを含まれる。

隣接行列：グラフのデータ構造（2）

- 6 つのノードと 7 つの枝(エッジ)から成るグラフの一例：隣接行列の0と1の並びが右上三角と左下三角で対称形（無向、重み無）

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



$V := \{1,2,3,4,5,6\}$

$E := \{\{1,2,1\},\{1,5,1\},\{2,3,1\},\{2,5,1\},\{3,4,1\},\{4,5,1\},\{4,6,1\}\}$

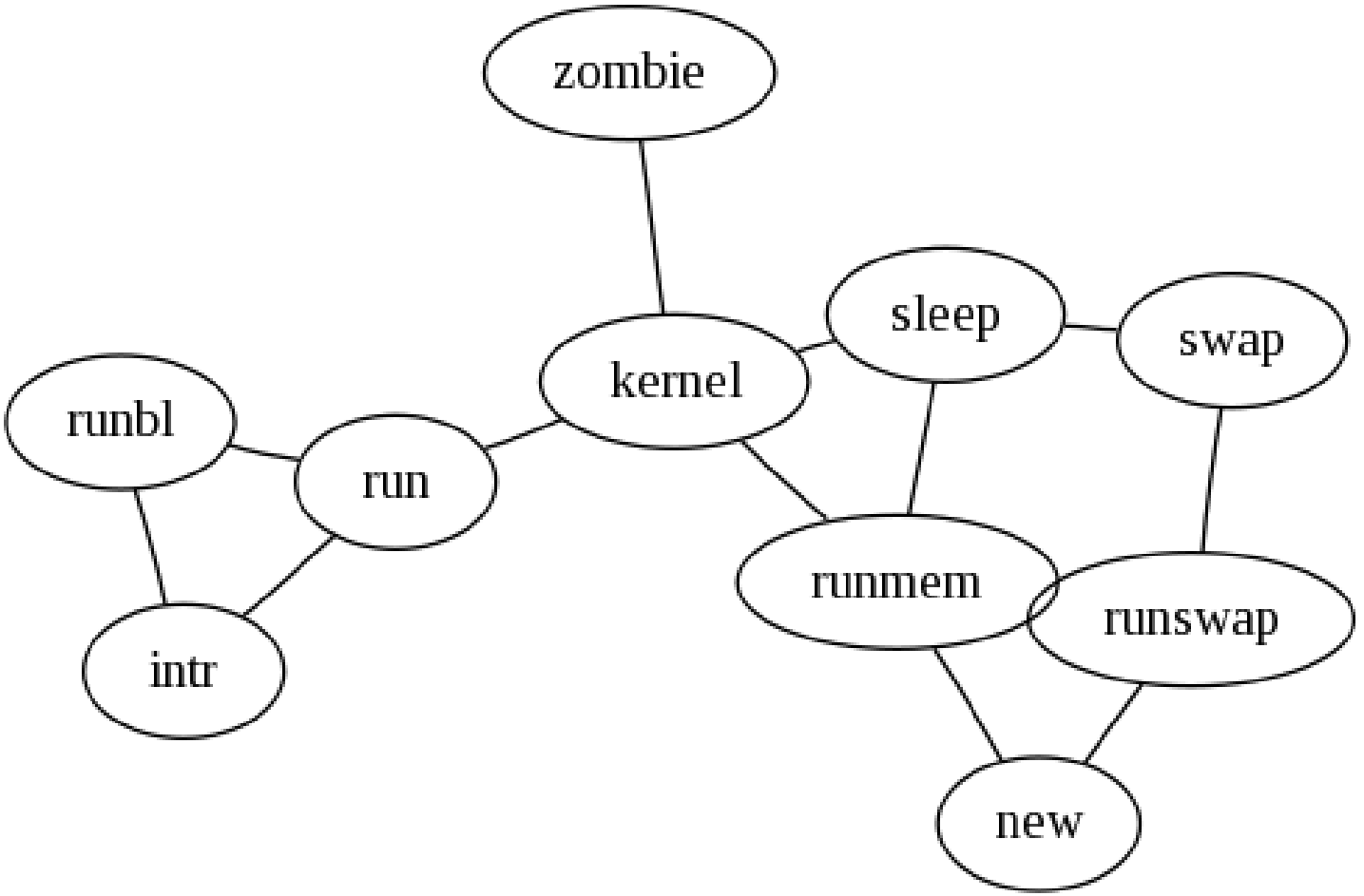
無効グラフ隣接行列の実装例(Python)

```
# リスト型を用いて隣接行列の実装
#グラフのデータ構造を作るルーチン（プログラム関数）
def graph_matrix(Edges,nodes):
    V,E = nodes,len(Edges) # ノードの数とエッジの数をVとE同時に代入
    # adj_matrix行列変数を初期化する（全ての要素を0にする）
    adj_matrix = [[0]*V for i in range(V)]
    # 繰り返しループによる、隣接ノードにadj_matrixの要素を1に変更する
    for i in range(E):
        u, v, w = Edges[i]
        adj_matrix[v-1][u-1] = adj_matrix[u-1][v-1] = w
    #adj_matrixを戻り値として返す
    return adj_matrix

## MAIN PROGRAM ##
edges = [(1,2,1),(1,5,1),(2,3,1),(2,5,1),(3,4,1),(4,5,1),(4,6,1)]

graph = graph_matrix(edges,6)

print "Graph of %d Nodes and %d Edges."%(len(graph),len(edges))
print "Adjacent Matrix:\n"
for each in graph:#each変数という行ごとに行列の一行づつ出力する
    print each
```

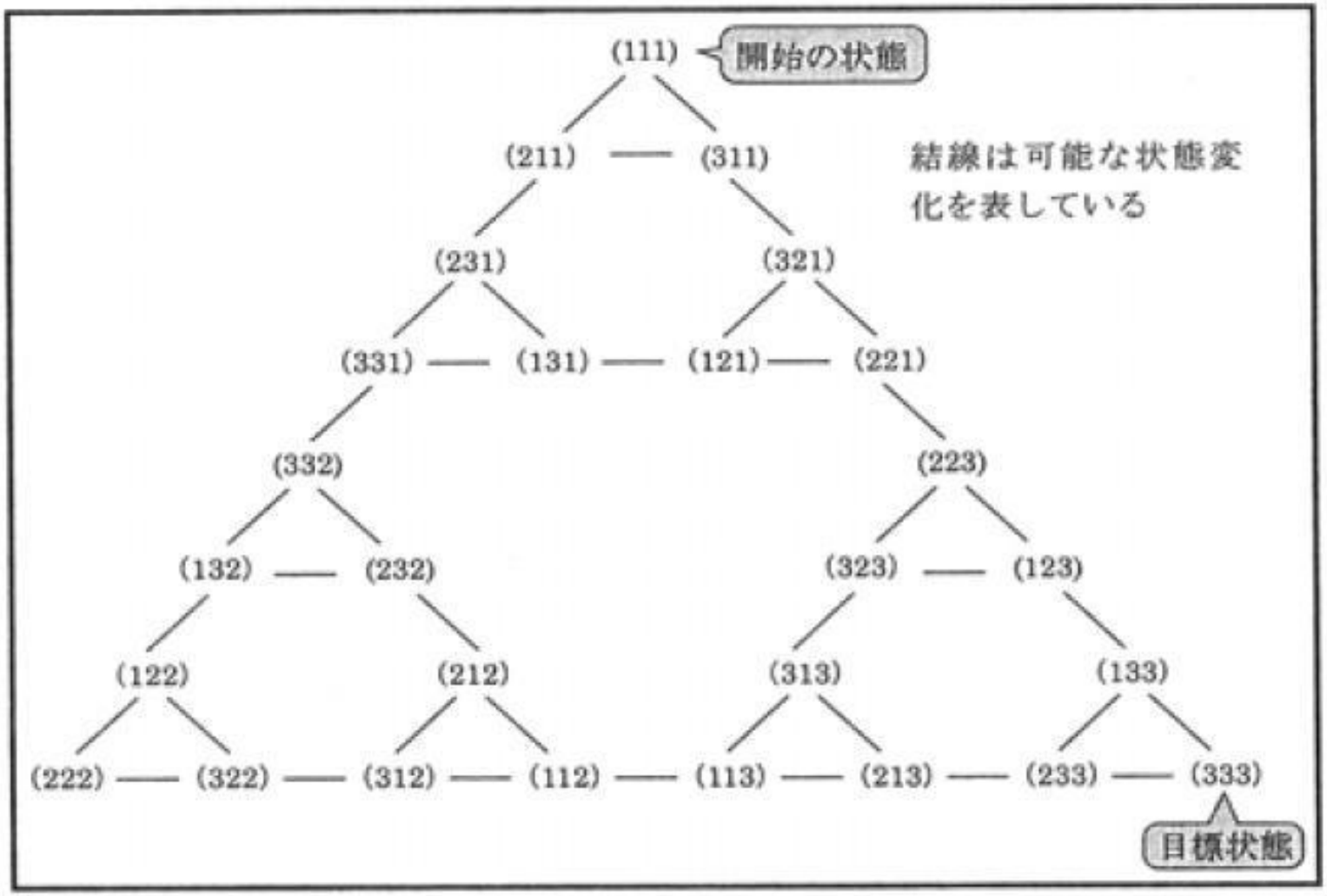
無効グラフ隣接行列の実装例 (Python)

```
>>> ================================
>>>
Graph of 6 Nodes and 7 Edges.
Adjacent Matrix:

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

CPUのプロセス状態のグラフ

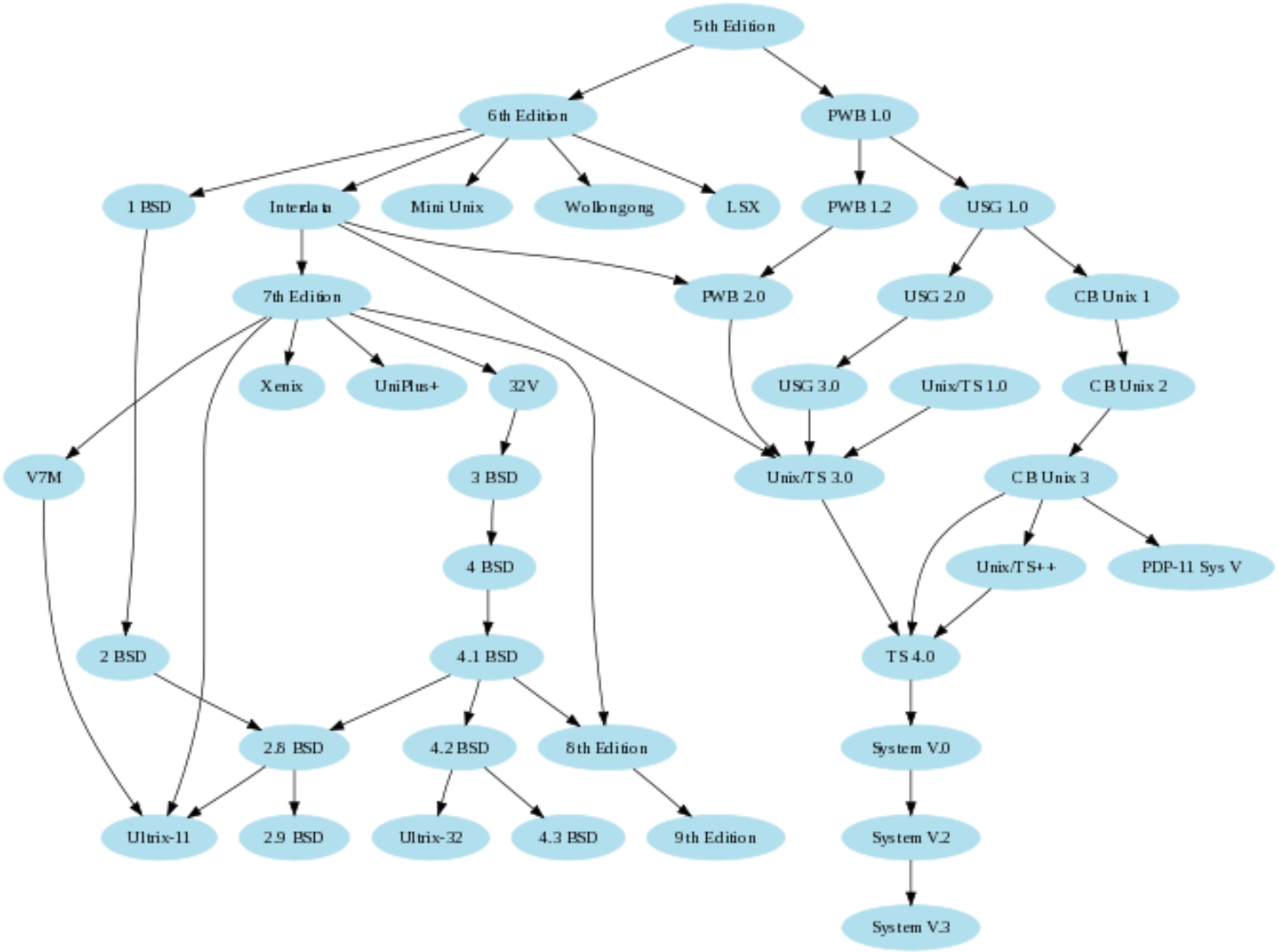


- 解の探索グラフ



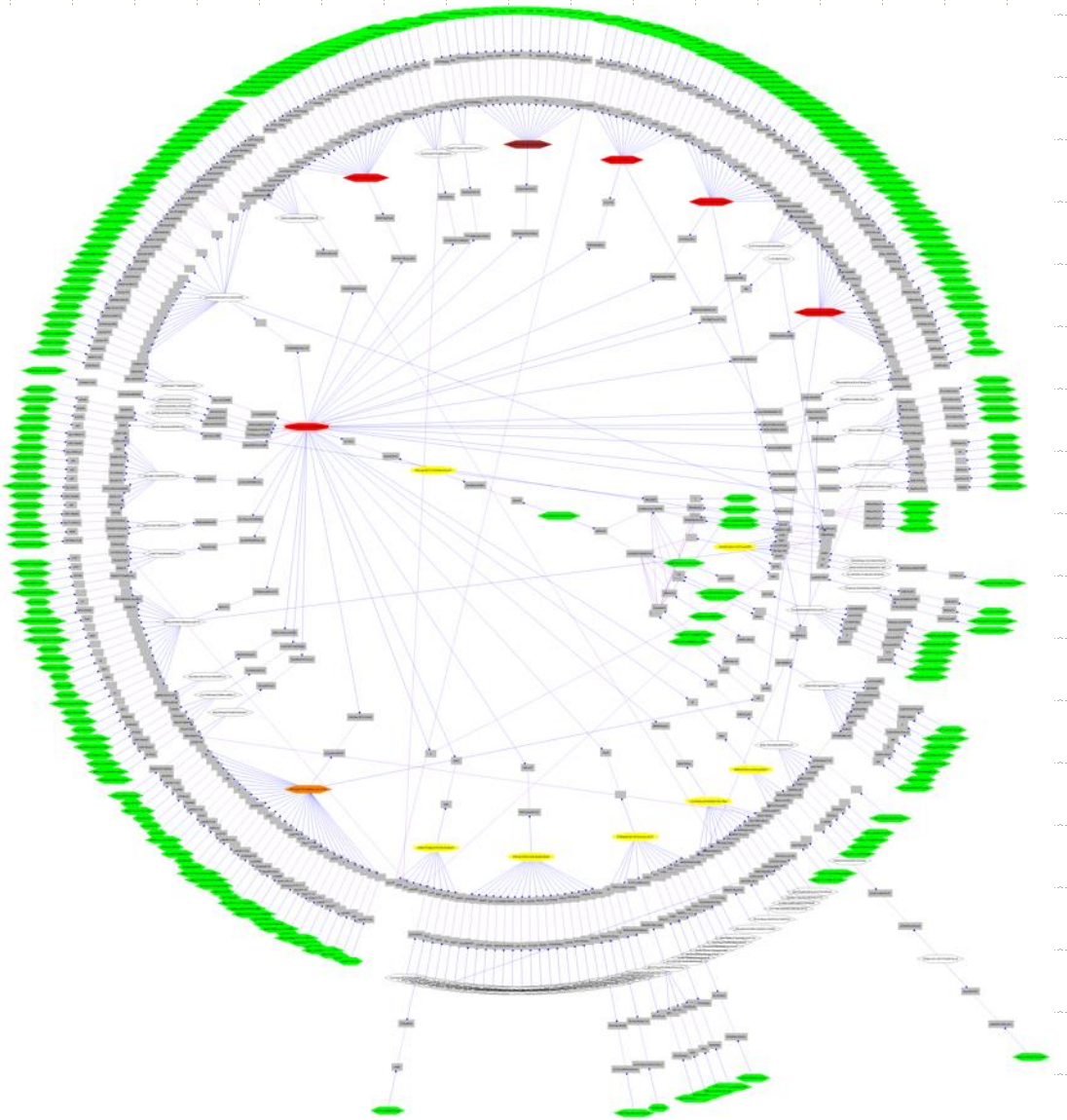
新しい人工知能(基本編) 前田隆他著 オーム社より

UNIX-OS

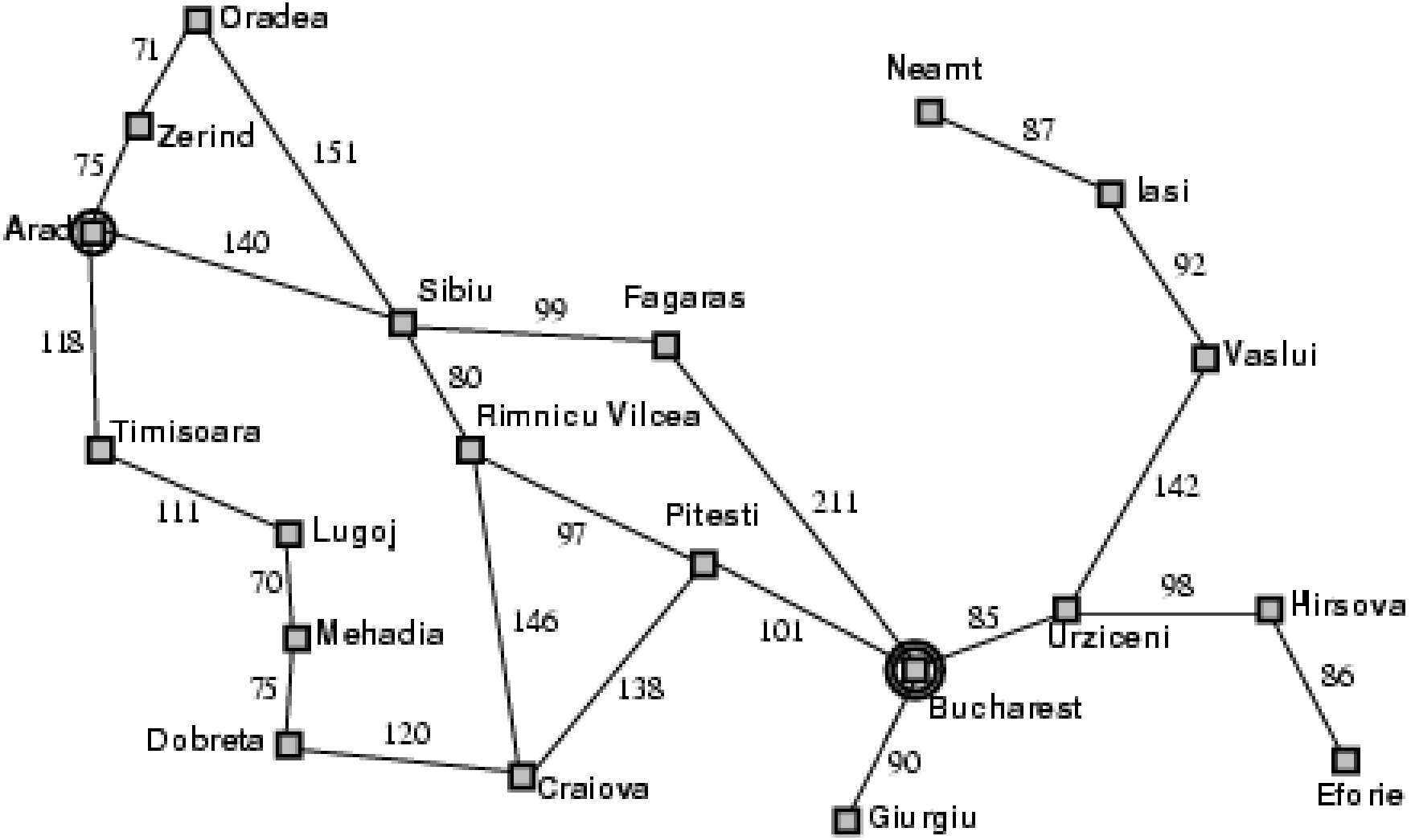

5th Edition
6th Edition
PWB 1.0
1 BSD
Interdata
Mini Unix
Wollongong
LSX
PWB 1.2
USG 1.0
7th Edition
PWB 2.0
USG 2.0
CB Unix 1
Xenix
UniPlus+
32V
USG 3.0
Unix/TS 1.0
CB Unix 2
V7M
3 BSD
Unix/TS 3.0
CB Unix 3
4 BSD
Unix/TS++
PDP-11 Sys V
2 BSD
4.1 BSD
TS 4.0
2.8 BSD
4.2 BSD
8th Edition
System V.0
Ultrix-11
2.9 BSD
Ultrix-32
4.3 BSD
9th Edition
System V.2
System V.3

ネットワークのグラフ

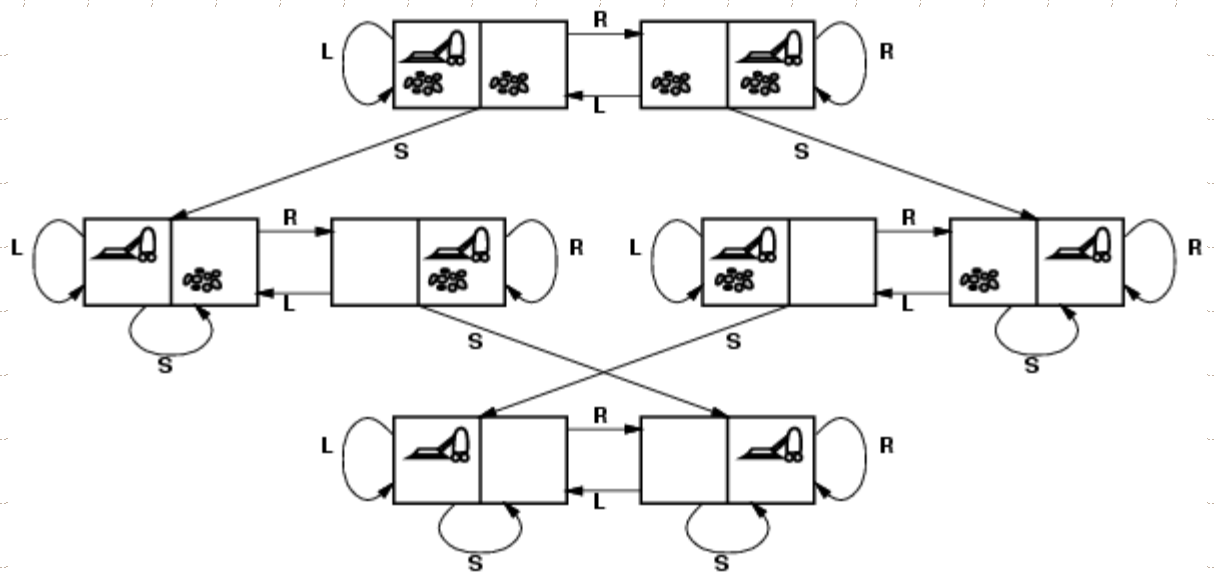
- 40か国の
ネットワーク状態



例: ロマニア(Romania)

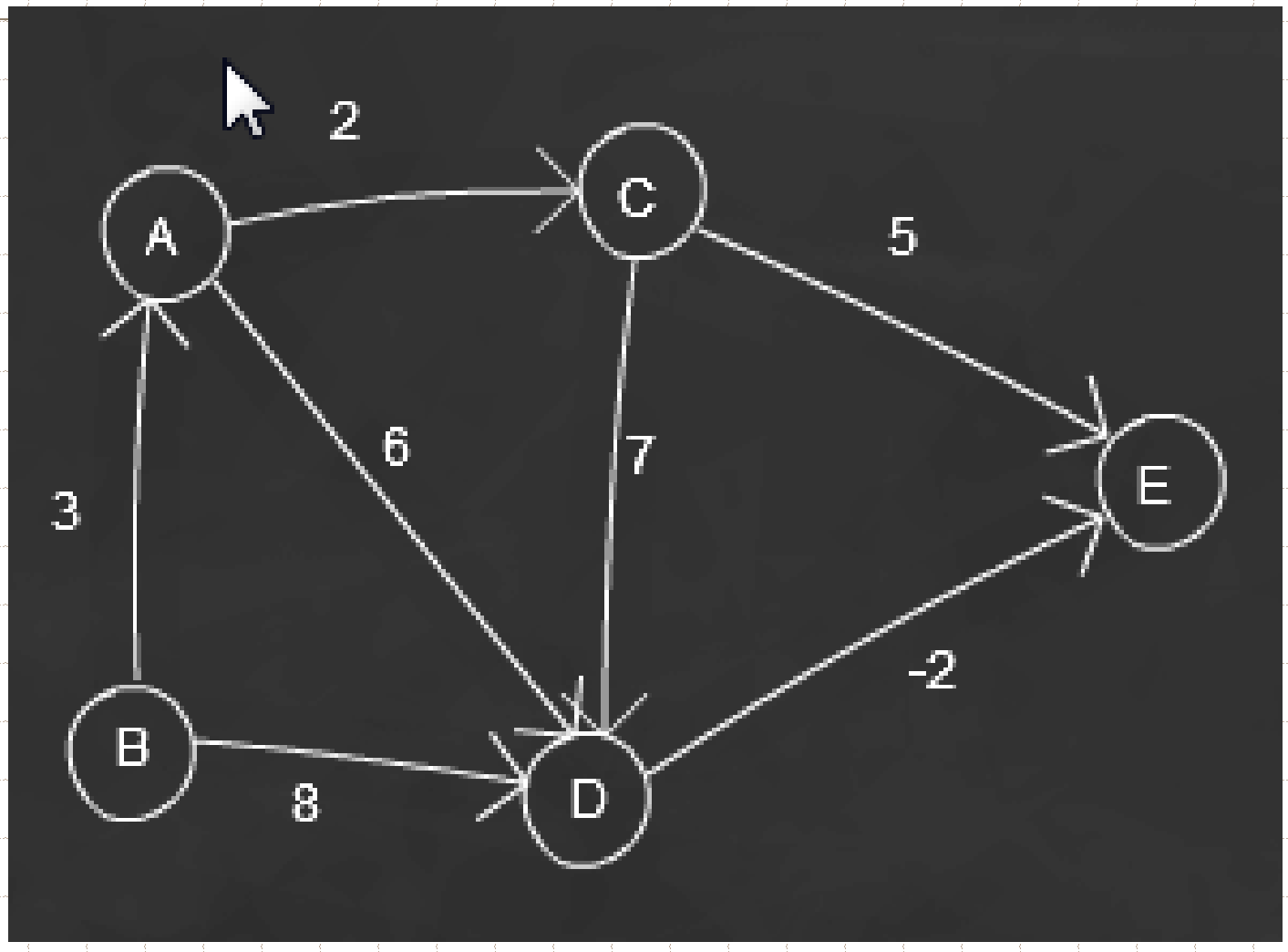


掃除機の世界：状態空間のグラフ (Vacuum world state space)



- 状態(states)? 状態1-8の部分集合(integer dirt and robot location)
- 行為(actions)? L:左に動く、R:右に動く、S:吸い込む(*Left*, *Right*, *Suck)*
- ゴール検査(goal test)?埃がない(no dirt at all locations)
- 経路コスト(path cost)? 1各々の行為は、コスト1を要する(1 per action)

グラフ実装の例:



```
# オブジェクトグラフのクラスとそのメソッドの定義
# オブジェクトグラフの操作
#

#
# 辞書データ構造で用いたグラフを定義する
#
L = {'A': {'C':2, 'D':6}, 'B': {'D':8, 'A':3},
        'C': {'D':7, 'E':5}, 'D': {'E':-2}, 'E': {}}

# Graphクラスを定義する
#
class Graph:
    def __init__(self, g):    # 初期メソッド
        self.g = g
    def Vertex(self):                 # ノードの集合を返すメソッド(辞書のキーの集合)
        return self.g.keys()
    def Adj(self,v):          # ノードvに隣接するノードのリストを返すメソッド
        return self.g[v].keys()
    def w(self,u,v):          # ノードuとノードvの枝の重みwを返すメソッド
        return self.g[u][v]


#  ここがメインプログラムです。

G = Graph(L)  # Lのグラフデータを Gというオブジェクトグラフに代入する

print ("グラフのポインタ: %s" % (G))
print ("グラフの全体: %s " % (G.g))
print ("ノードの集合: %s" % (G.Vertex()))
print ("ノード'D'に隣接するノードのリスト: %s" % (G.Adj('D')))
print ("重み(weight) D->E: %s " % (G.w('D','E')))
```

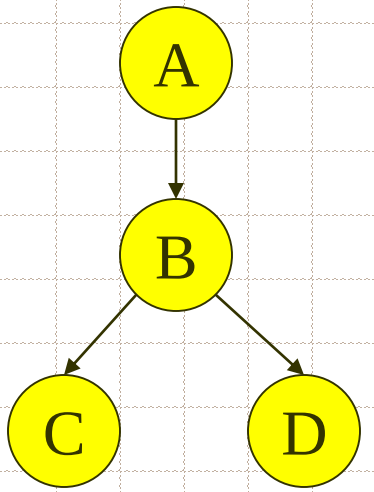
Python 3 プログラム

```
>>> ========== RESTART ==========
>>>
グラフのポインタ: <__main__.Graph object at 0x02AA2690>
グラフの全体: {'A': {'C': 2, 'D': 6}, 'C': {'E': 5, 'D': 7}, 'B': {'A': 3, 'D': 8
}, 'E': {}, 'D': {'E': -2}}
ノードの集合: dict_keys(['A', 'C', 'B', 'E', 'D'])
ノード'D'に隣接するノードのリスト: dict_keys(['E'])
重み(weight) D->E: -2
>>>
```

ベイジアンネット(A Bayesian Network)

ベイジアンネットは:

1. 有効非環式グラフ
（directed acyclic graph)



2. 各々ノードに条件付き確率表(Conditional Probability Table)が付く

A	P(A)
false	0.6
true	0.4

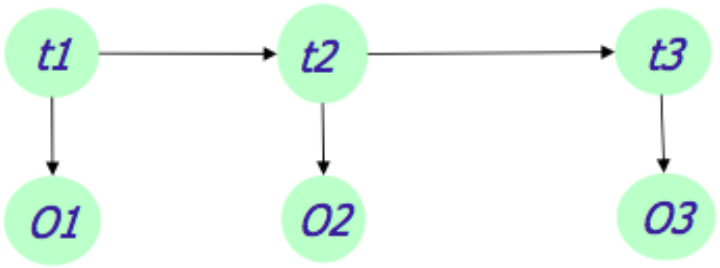
A	B	P(B|A)
false	false	0.01
false	true	0.99
true	false	0.7
true	true	0.3

B	D	P(D|B)
false	false	0.02
false	true	0.98
true	false	0.05
true	true	0.95

B	C	P(C|B)
false	false	0.4
false	true	0.6
true	false	0.9
true	true	0.1

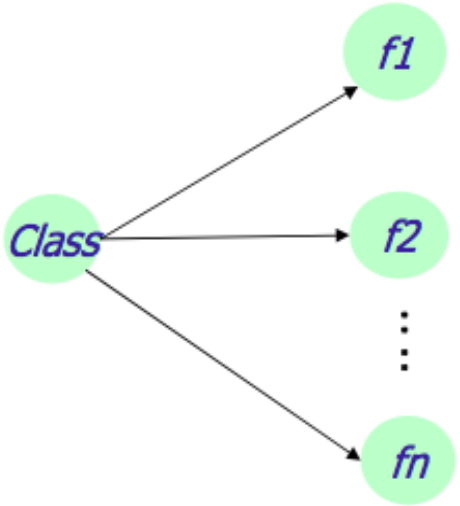
ベイジアンネット(Bayesian Network)とベイジアン分類機(Bayesian Classifier)

音声認識でよく使われるHMM(Hidden Markov Model)
と等価なベイジアンネット



t,…は状態を表すノード
Oは出力記号を表すノード

パターン(画像)認識でよく使われるBayesian Classifier
と等価なベイジアンネット



まとめ

- グラフについて、定義、種類、特徴
- オイラー・グラフ
- 林、木について定義
- グラフオブジェクトのデータ構造とメソッド
- 例題
- グラフの応用例

8パズルの探索木

深さ0初期状態：ノード0、根(ルートノード)

深さ1

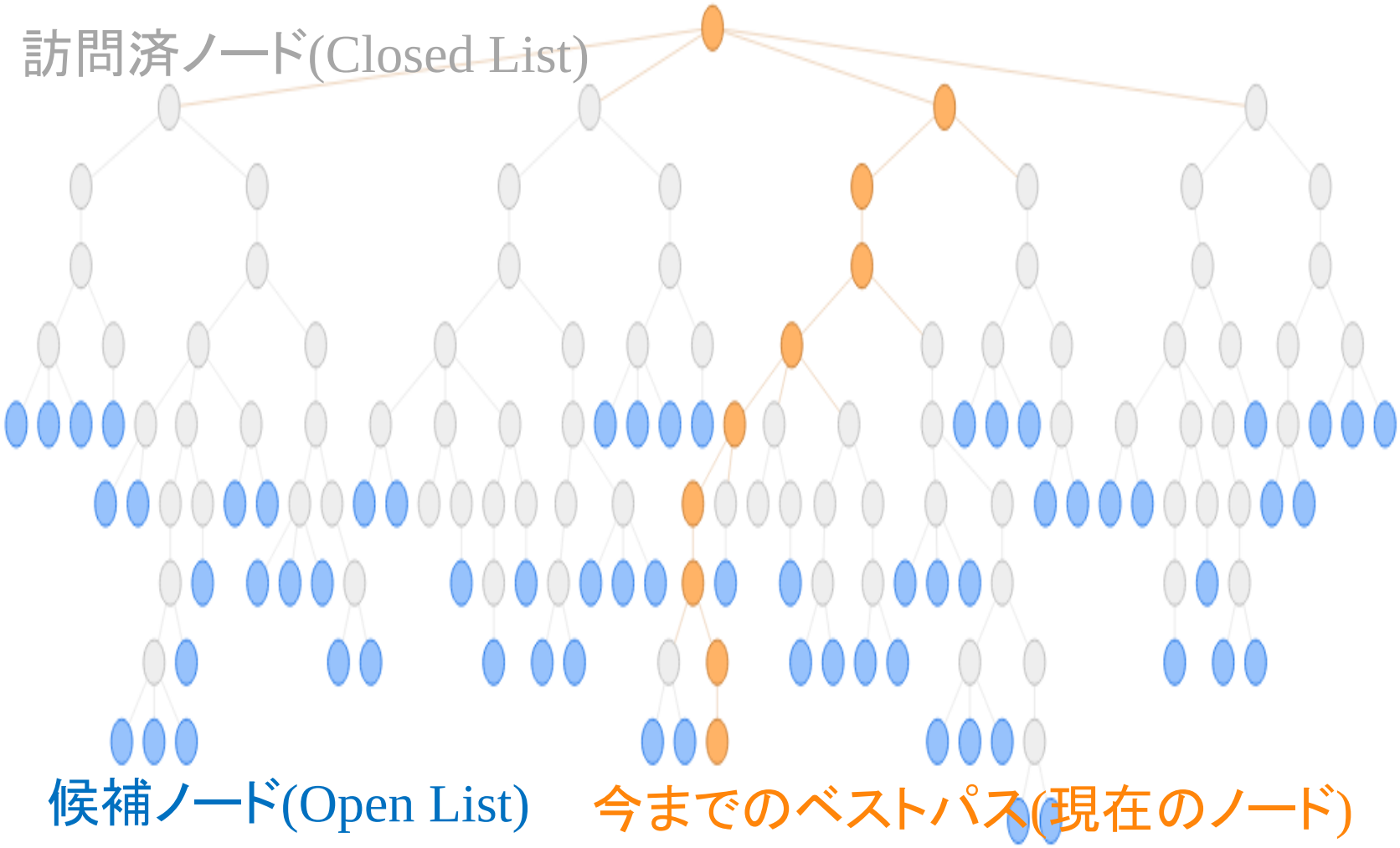
深さ2

深さ3

深さ4

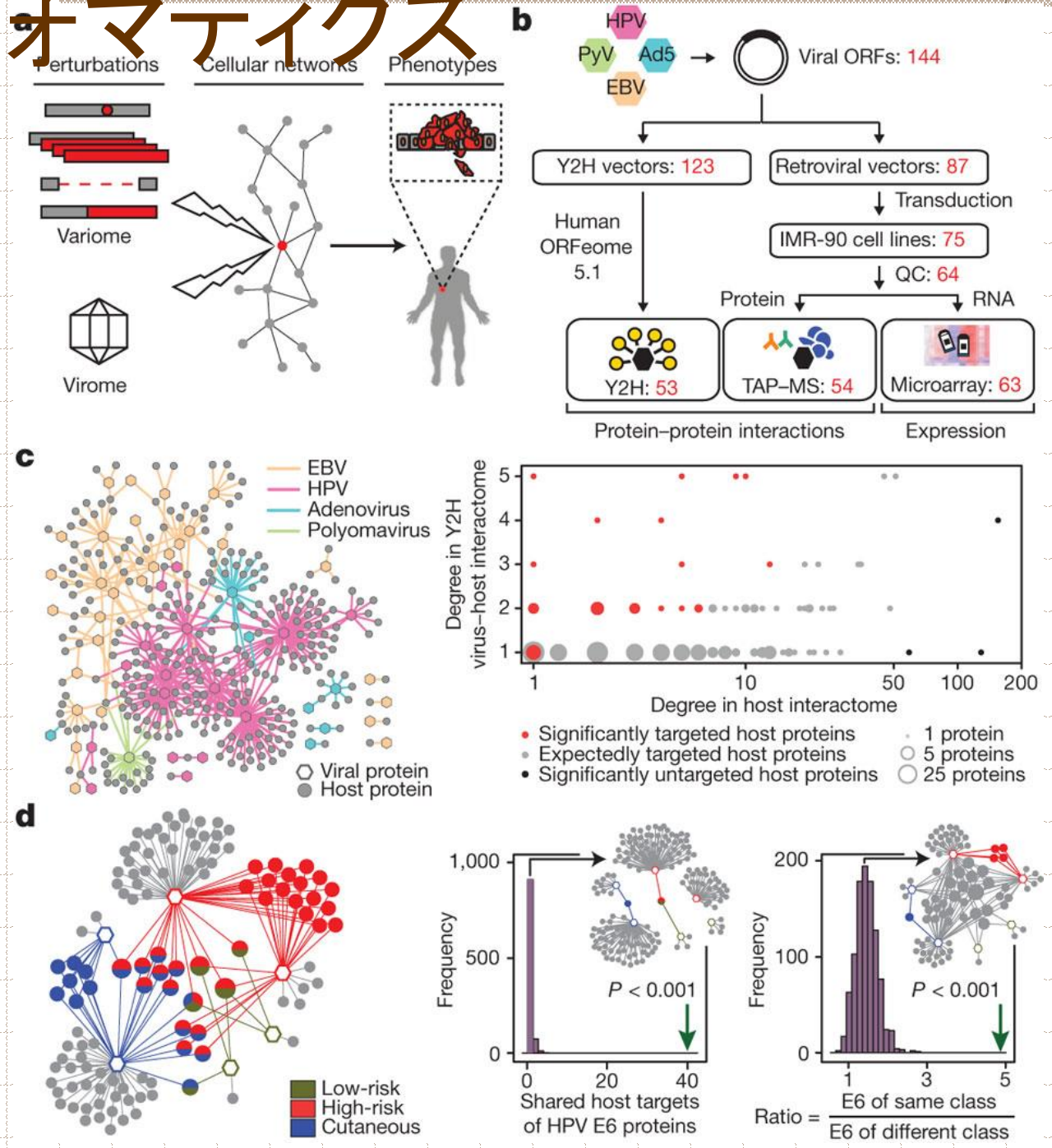
深さ5

…

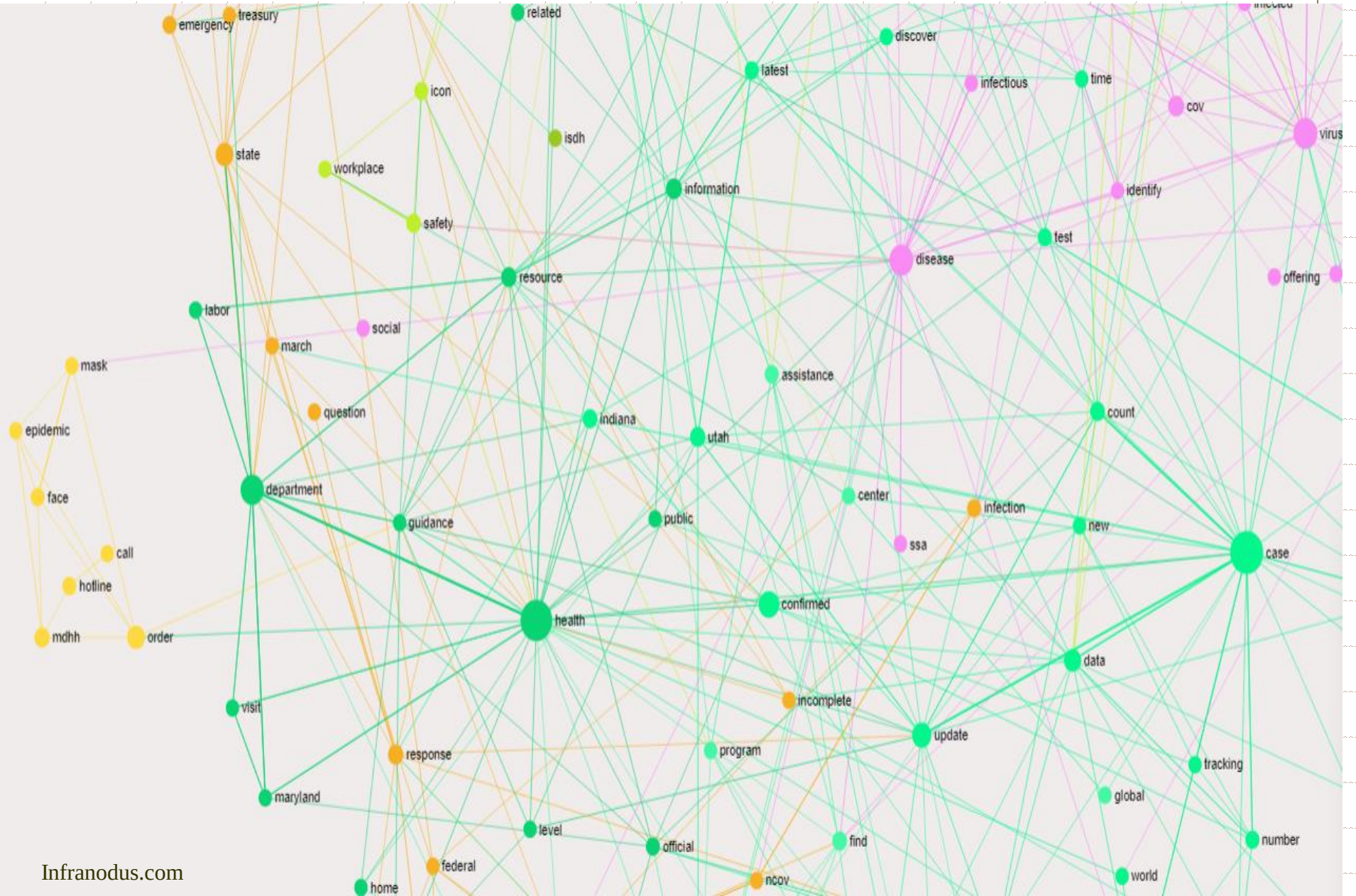


バイオインフォマティクス

- ウイルス検出
- タンパク質
- RNA
- ネットワーク
- クラスタ化
- 分析



COVID-19意味ネットワークグラフ



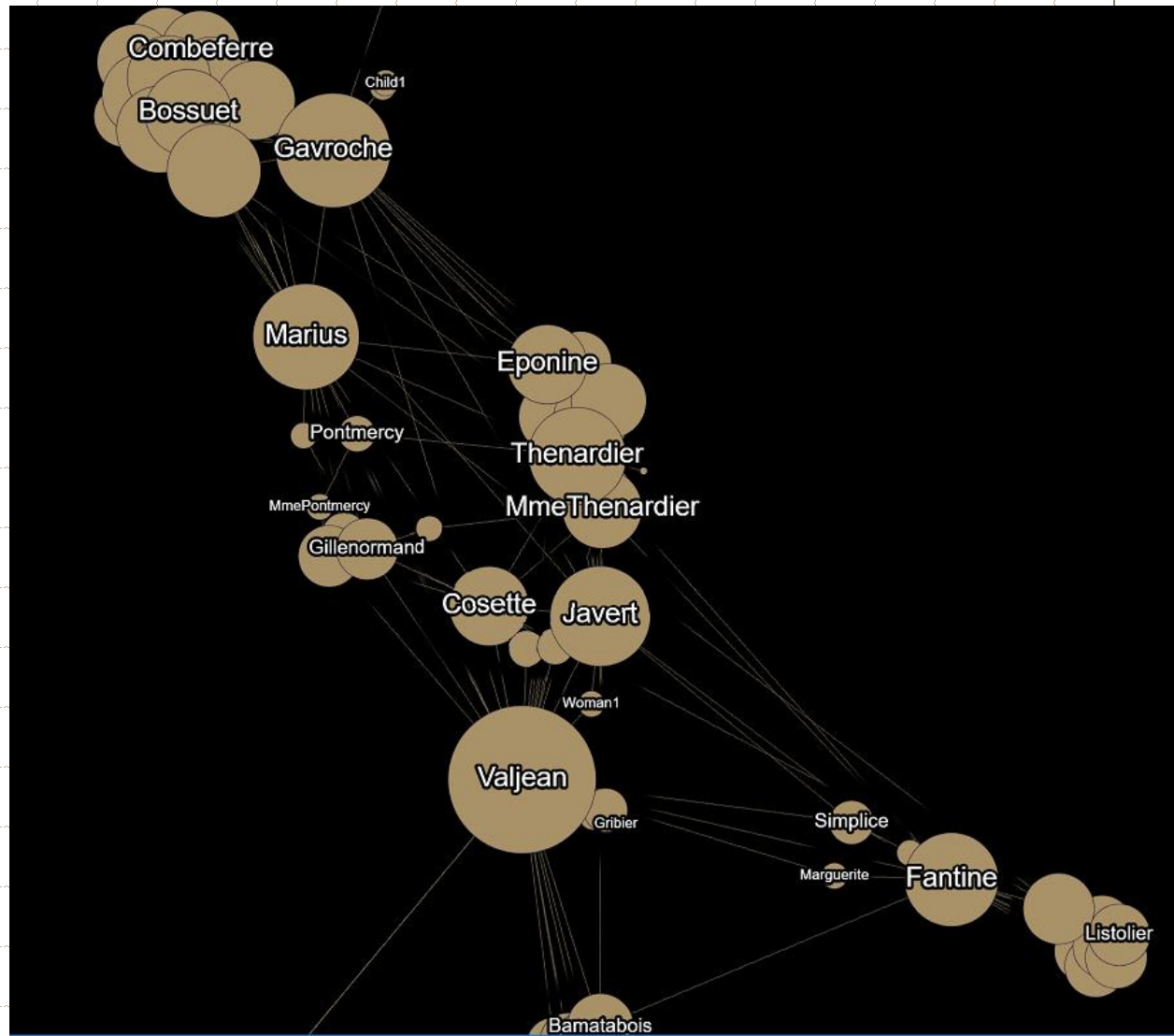
Infranodus.com

レ・ミゼラブル

登場人物
グラフネットワーク
主人公：Valjean, Thenardier, Cosette, Marius, Gavroche

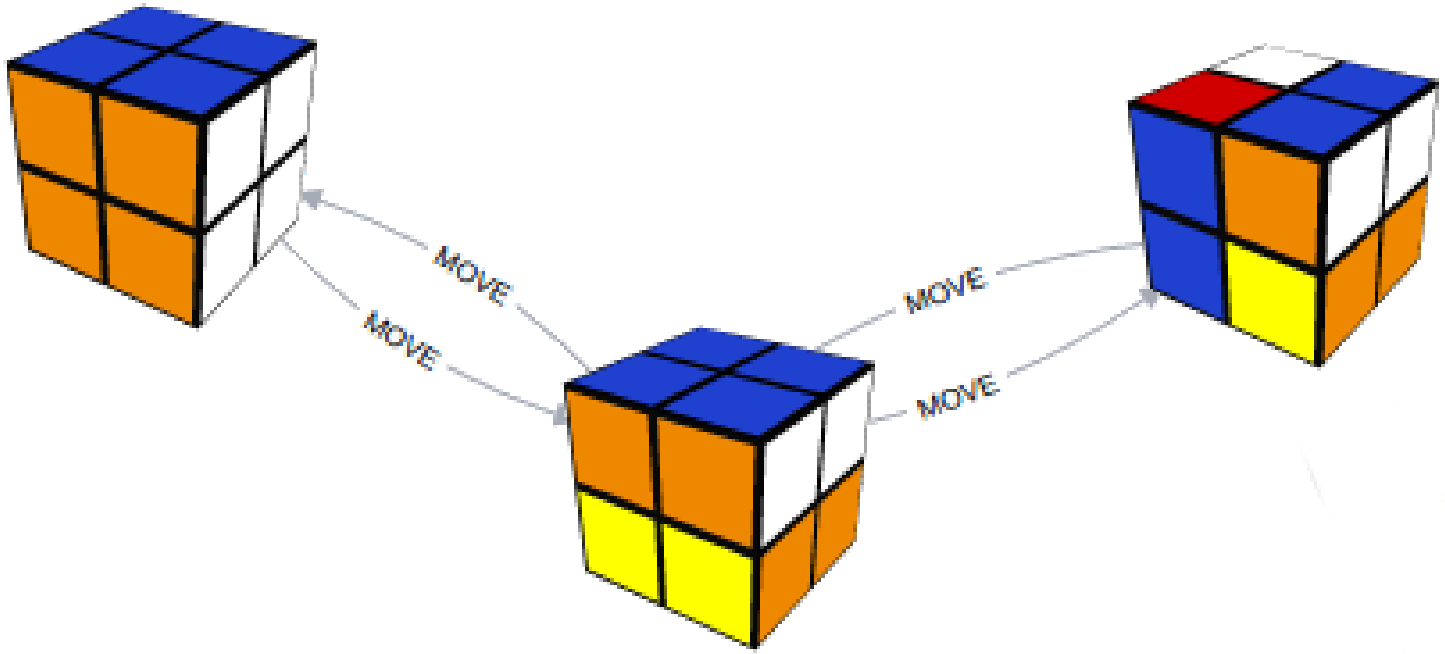
レ・ミゼラブル

登場人物
グラフネットワーク
主人公：Valjean, Javert, Cosette, Marius, Eponine, Fantine



ルッビクキューブの状態空間とグラフ

- グラフの応用例
- オペレーターは状態遷移（辺、線）(edges)
- 状態は点又はノード(vertex)



ルッビク

- キューブ
- 2x2
- 状態空間
- 6色