クレジット:
UTokyo Online Education 数理手法Ⅲ 2018 寒野善博



東京大学 THE UNIVERSITY OF TOKYO

CC BY NC ND

# NetworkX によるネットワーク計画問題の解き方

この資料では，NetworkX という Python のパッケージ[*1]を用いてネットワーク計画問題を解く方法を解説する．

---

- この資料は，主に「最適化のプログラムを実際に動かしてみたい」という人向けの，補助資料です．講義の履修や単位の取得に，プログラミングは必要ではありません．
- 実装例のファイルは，https://www.or.mist.i.u-tokyo.ac.jp/kanno/lecture/ にあります．

---

## 1. 最短路問題の例

重み付き有向グラフ $G = (V, E, c)$ について，二つの頂点 $s, t \in V$ が指定されたとき，$s$ から $t$ への道のうち重みが最小のものを求める問題を最短路問題という．ただし，辺の重みはすべて 0 以上であるとする．また，$s$ を始点とよび，$t$ を終点とよぶ．

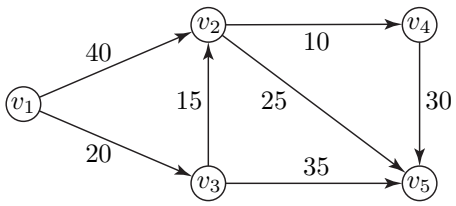


図 1: 最短路問題の例で用いるグラフ

最短路問題は，ダイクストラ (Dijkstra) 法を用いると効率よく解けることが知られている．NetworkX にも，ダイクストラ法が実装されている．図 1 のグラフにおいて，頂点 $v_1$ を始点とし頂点 $v_5$ を終点とする最短路は，次のような Python のコードで求めることができる．

```
# ファイル名：'shortest_path_ex.py'
import networkx as nx          #まずはパッケージの読み込み
G = nx.DiGraph()               #空な有向グラフGを生成
G.add_nodes_from(['v1','v2','v3','v4','v5'])  #Gに頂点v1,...,v5を追加
G.add_weighted_edges_from([\
    ('v1','v2',40.0),('v1','v3',20.0),('v2','v4',10.0),\
    ('v2','v5',25.0),('v3','v2',15.0),('v3','v5',35.0),\
    ('v4','v5',20.0)])                    #Gに辺とその重みを追加
print( nx.dijkstra_path(G,'v1','v5') )  #ダイクストラ法の実行と最適解の出力
```

```
>>>
['v1', 'v3', 'v5']
>>>
```

つまり，$v_1 \to v_3 \to v_5$ が最適解であることが分かる．

[*1] https://networkx.github.io/

## 2. 最小木問題の例

連結なグラフであって，閉路をもたないものを，木とよぶ．連結な重み付き無向グラフ $G = (V, E, c)$ に対して，$G$ の部分グラフのうち $V$ を頂点集合とする木のことを，$G$ の全域木とよぶ．最小木問題は， $G$ の全域木のうち重みが最小のものを求める問題である．

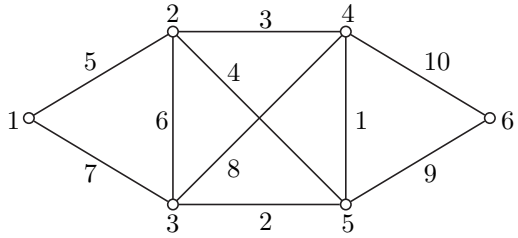


図 2: 最小木問題の例で用いるグラフ

クラスカル (Kruskal) のアルゴリズムは，最小木問題の解法の一つである．NetworkX でクラスカルのアルゴリズムを用いて図 2 のグラフの最小木を求めるには，次のようにコーディングすればよい．

```
1  # ファイル名：spanning_tree_ex.py
2  import networkx as nx                #まずはパッケージの読み込み
3  G = nx.Graph()                       #空な無向グラフGを生成
4  G.add_nodes_from([1,2,3,4,5,6])      #Gに頂点を追加
5  G.add_weighted_edges_from([ \
6        (1,2,5.0), (1,3,7.0), (2,3,6.0), (2,4,3.0), \
7        (2,5,4.0), (3,4,8.0), (3,5,2.0), (4,5,1.0), (4,6,10.0), \
8        (5,6,9.0) ])                   #Gに辺とその重みを追加
9  T = nx.minimum_spanning_tree(G)      #クラスカルのアルゴリズムを実行
10 print( T.edges(data=True) )          #最適解の出力
```

このコードを実行すると，次のような出力が得られる．

```
>>>
[(1, 2, {'weight': 5.0}), (2, 4, {'weight': 3.0}), (3, 5, {'weight': 2.0}),
 (4, 5, {'weight': 1.0}), (5, 6, {'weight': 9.0})]
>>>
```

つまり，最小木の辺は $(1,2)$, $(2,4)$, $(3,5)$, $(4,5)$, $(5,6)$ であることがわかる．

（以上）