Python勉強会@HACHINONE

第17章

ナップサック問題とグラフ最適化問題

お知らせ

Python勉強会@HACHINOHEでは、ジョン・V・グッターグ『Python言語による
プログラミングイントロダクション』近代科学社、2014年をみんなで勉強しています。

この本は自分で読んで考えて調べると力が付くように書かれています。

自分で読んで考えて調べる前に、このスライドを見るのは、いわばネタバレを
聞かされるようなものでもったいないです。

是非、本を読んでからご覧ください。

最適化問題

Python勉強会@HACHINOHE

- 制約条件のもとで、目的関数を最適化する
  - 目的関数: ボストンとイスタンブール間の航空運賃
  - 制約条件: 旅行時間の上限
- 代表的な最適化問題
  - ナップサック問題: 重さ上限のある最大価格の組み合わせ
  - グラフ最適化問題: 最短経路とか
- 解法
  - 完全列挙法: 全部試す。時間がかかる
  - 貪欲法: ある簡単な基準で決めていく。局所最適化で、本当に最適とは限らない

ナップサック問題

Python勉強会@HACHINOHE

- 20ポンド(9kgくらい)まで盗めるとき、何を盗むべきか？

品物	値段	重さ	値段/重さ
置き時計	175	10	17.5
絵画	90	9	10
ラジオ	20	4	5
花瓶	50	2	25
本	10	1	10
PC	200	20	10

貪欲法

Python勉強会@HACHINOHE

- 3つの貪欲法
  - 高い方からつめる
    - 200ドル: PC
  - 軽い方からつめる
    - 170ドル: 本、花瓶、ラジオ、絵画
  - 「値段/重さ」が大きい方からつめる
    - 255ドル: 花瓶、置き時計、本、ラジオ

- 本当の最適解
  - 275ドル: 置き時計、絵画、本

貪欲法プログラム

Python勉強会@HACHINOHE

品物
名前 価格 重さ
初期化(名前, 価格, 重さ) 名前を取得() 価格を取得() 重さを取得() 文字列化()

- 目的関数3種
  - 価格を取得(品物)
  - 重さの逆数を取得(品物)
  - 価格/重さを取得(品物)

- 品物群を生成()
- ある貪欲法を実行(品物群, 運べる重さ, 目的関数)
- ある貪欲法の実行結果を表示(品物群, 条件, 目的関数)
- 3通りの貪欲法の実行結果を表示(運べる重さ)

貪欲法の計算時間

Python勉強会@HACHINOHE

```
def greedy(items, maxWeight, keyFunction):
    """Itemsはリスト、maxWeight >= 0とし、
       keyFunctionはそれぞれのitemをその属性値にマップする"""
    itemsCopy = sorted(items, key=keyFunction, reverse = True)
    result = []
    totalValue = 0.0
    totalWeight = 0.0
    for i in range(len(itemsCopy)):
        if (totalWeight + itemsCopy[i].getWeight()) <= maxWeight:
            result.append(itemsCopy[i])
            totalWeight += itemsCopy[i].getWeight()
            totalValue += itemsCopy[i].getValue()
    return (result, totalValue)
```

O(n log n)

O(n)

0/1ナップサック問題の定式化

Python勉強会@HACHINOHE

- ある品物を盗る、盗らないの2択のナップサック問題
- 品物ベクトル $\vec{I} = (item_0, item_1, \ldots, item_{n-1})$
  - 各品物itemは、価格valueと重さweightを持つ
- 盗品ベクトル $\vec{V} = (1,0,0,...,1)$ ※盗む品物のところが1

- 目的関数 $\sum_{i=0}^{n-1} V[i] \times I[i].value$ 盗む品物の価値の合計
- 制約条件 $\sum_{i=0}^{n-1} V[i] \times I[i].weight \leq w$ 盗む品物の重さがw以下

完全列挙法プログラム

Python勉強会@HACHINOHE

品物
名前 価格 重さ
初期化(名前, 価格, 重さ) 名前を取得() 価格を取得() 重さを取得() 文字列化()

※貪欲法と同じ

- 品物群を生成() ※貪欲法と同じ
- 最適解を求める(組合せ, 運べる重さ, 価格を取得するメソッド, 重さを取得するメソッド)
- 2進数化(値, 2進数の桁数)
- 品物のすべての組合せを生成(品物群)
- 完全列挙法で最適解を求める(運べる重さ)

完全列挙法の計算量

Python勉強会@HACHINOHE

- 内外のループをかけると、計算量は $O(n\,2^n)$

```
def chooseBest(pset, maxWeight, getVal, getWeight):
    bestVal = 0.0
    bestSet = None
    for items in pset:
        itemsVal = 0.0
        itemsWeight = 0.0
        for item in items:
            itemsVal += getVal(item)
            itemsWeight += getWeight(item)
        if itemsWeight <= maxWeight and itemsVal > bestVal:
            bestVal = itemsVal
            bestSet = items
    return (bestSet, bestVal)
```

← 2^n個

← 最大n個

ナップサック問題と解法

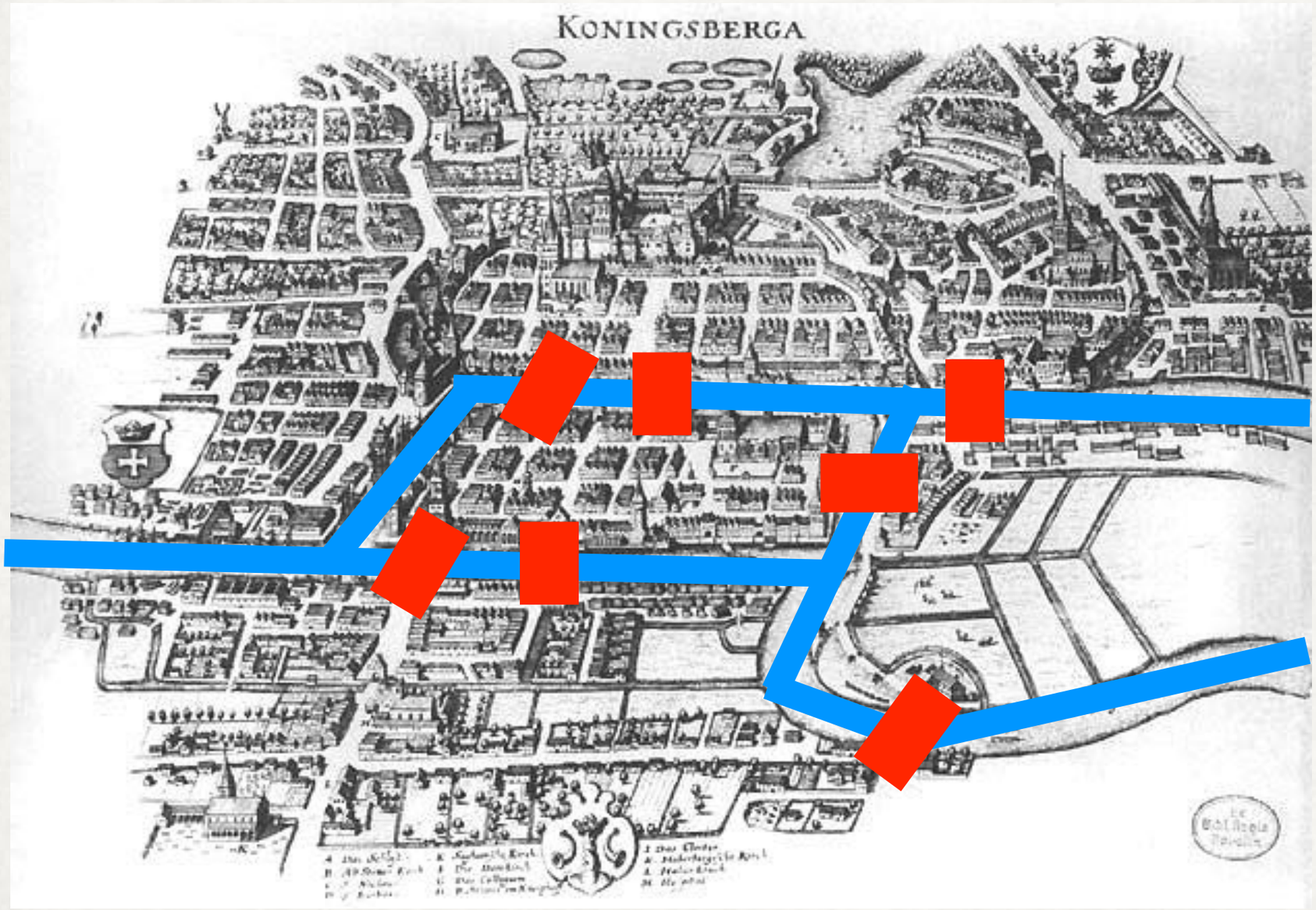
Python勉強会@HACHINOHE

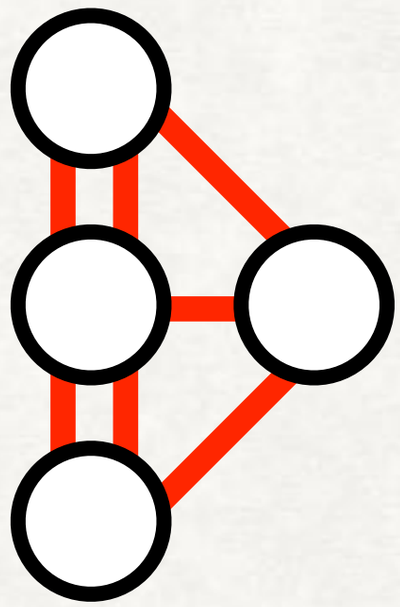
- 0/1ナップサック問題の例
  - 品物を選ぶ/選ばないの2択
  - 貪欲法は必ずしも最適解にならない
- 同じ品物が複数あるナップサック問題もある
- 分数ナップサック問題
  - 品物が個数でなく量(粉とか液体とか)の場合
  - 貪欲法で最適解になる

ケーニヒスベルクの橋

Python勉強会@HACHINOHE

- 東プロイセンの首都ケーニヒスベルク





By Merian-Erben - http://www.preussen-chronik.de/_/bild_jsp/key=bild_kathe2.html, Public Domain, https://commons.wikimedia.org/w/index.php?curid=1447648

グラフのクラス図

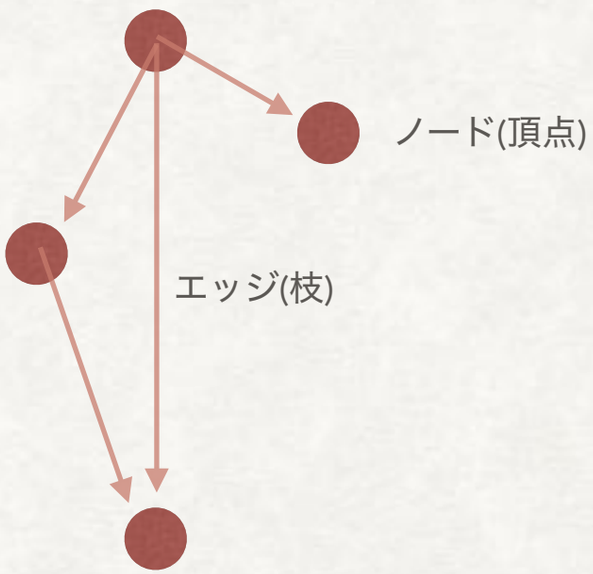
Python勉強会@HACHINOHE

ノード
名前
初期化(名前) 名前を取得() 文字列化()

エッジ
始点 終点
初期化(始点, 終点) 始点を取得() 終点を取得() 文字列化()

有向グラフ
ノード群 エッジ群
初期化() ノードの追加(ノード) エッジの追加(エッジ) 子ノードの一覧(ノード) ノードを含むか(ノード) 文字列化()

有向グラフ



重み付きエッジ
重み
初期化(始点, 終点, 重み) 重みを取得() 文字列化()

グラフ
エッジの追加(エッジ)

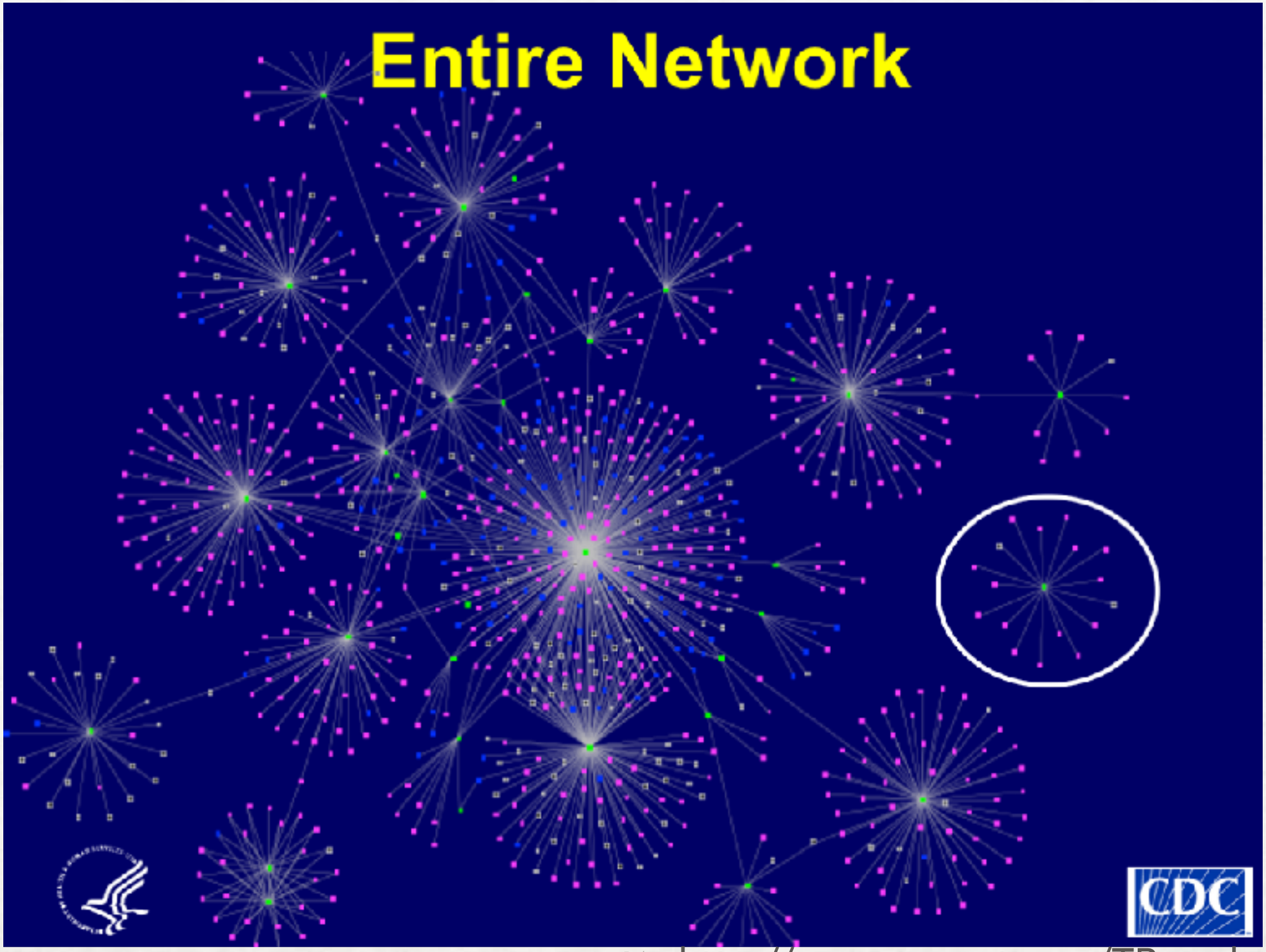
古典的なグラフ理論の問題

Python勉強会@HACHINOHE

- 最短経路問題: 最少のステップで到達できる経路
- 最短重み付き経路問題: 距離や料金などが最少の経路
- クリーク(派閥)問題: すべて互いにつながりあっているノード群
- 最小カット問題: グラフを切断するのに必要な最小のノード群
  - 伝染病の感染経路

結核の拡散

Python勉強会@HACHINOHE



http://www.orgnet.com/TB_web.ppt p.23

六次の隔たり

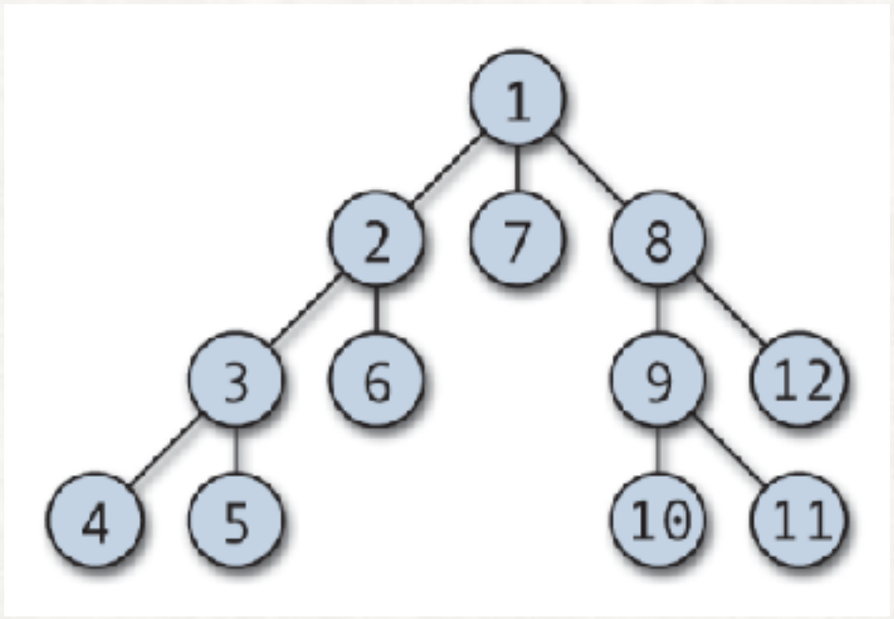
Python勉強会@HACHINOHE

- small world network
- 世界中の誰とでも6人の知り合いを介せば繋がっている
  - 映画「私に近い6人の他人」。ジョン・グエア原作
  - GREEの名前の由来でもある
- 1967年のスタンレー・ミルグラムの実験
  - アメリカ中部の人160人に、東海岸のある人に、バケツリレー方式で手紙を送るように指定。手紙は直接の知り合いにしか送ってはならない
  - およそ5、6人経由で届いた

最短経路を求める方法

Python勉強会@HACHINOHE

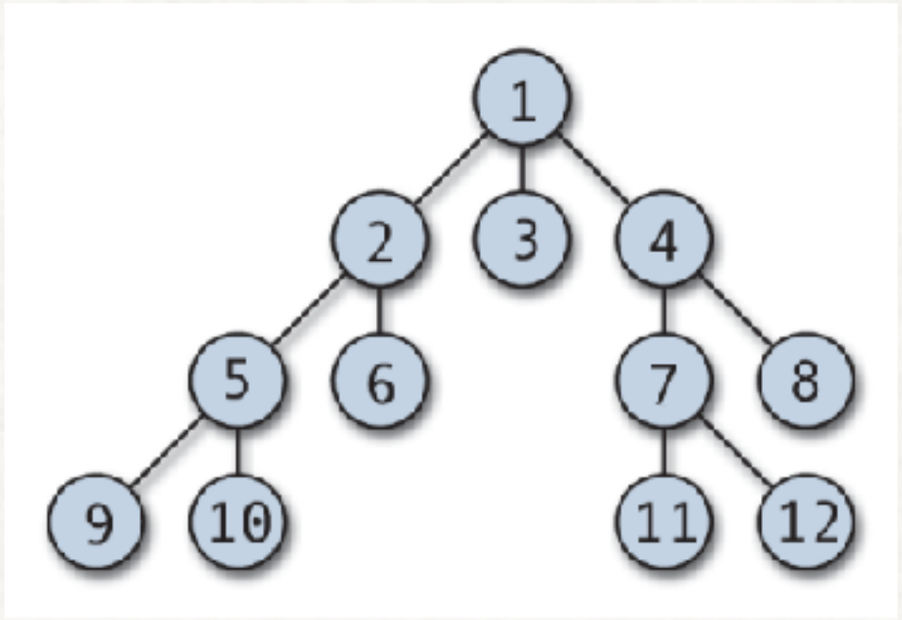
- 深さ優先探索
  - 1本ずつ(?)調べていく



By Wolfram Esser - Copied over from german Wikipedia where it was saved under the name Tiefensuche.png. The original text for the picture was: Beschreibung:Tiefensuche: Beispielbaum mit Reihenfolge, in der die Baumknoten besucht werdenQuelle: Eigene ZeichnungErsteller: Wolfram EsserCopyright Status: GNU Freie Dokumentationslizenz CC 表示-継承 3.0,

https://commons.wikimedia.org/w/index.php?curid=354665

- 幅優先探索
  - 1段ずつ調べていく



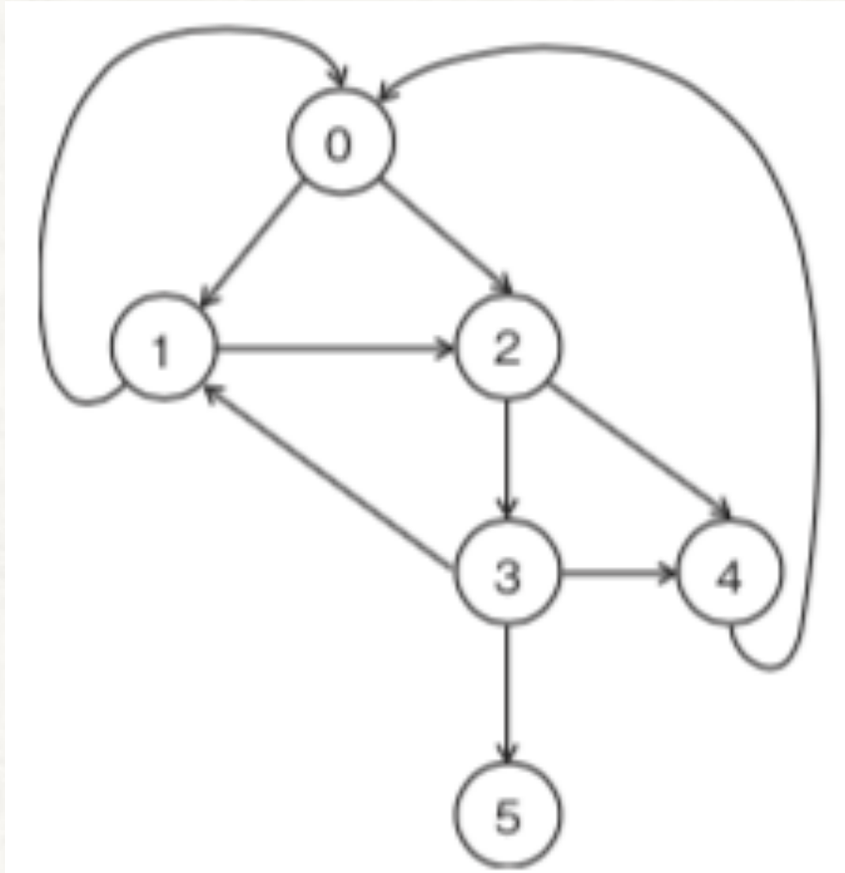
CC 表示-継承 3.0

https://commons.wikimedia.org/w/index.php?curid=354656

有向グラフの例

Python勉強会@HACHINOHE

- 0さんから、5さんまで、最小で何人を介せば到達するか
- この例の場合、深さ優先探索でも幅優先探索でも、答えは一致
- プログラミング上は、経路にノードを追加して調べていく
  - このとき、現在の経路に既にノードが含まれていたら追加しない
    - 閉路(サイクル)があると、いつまでたっても終わらない
    - 既にその先は計算されている



p.276 図17.3

深さ優先探索と幅優先探索の比較

Python勉強会@HACHINOHE

- 深さ優先探索
  - 再帰で実装されることが多い
  - 1本ずつ(?)調べるので最初に見つけた経路が最短とは限らない
  - 本の実装は重みを評価していないので、重み付きの最短経路ではない(そうすることもできる)
- 幅優先探索
  - 繰り返しで実装されることが多い
  - 最初に見つけた経路は最短のうちの一つ ※早く打ち切れる
  - その経路は一般に重み付きの最短経路ではない ※指練習