アルゴリズムとデータ構造のはなし

辻 真吾（@tsjshg）

みんなのPython勉強会＃64 ―師走tapy（しわすたぱい）―

12/9, 2020

https://startpython.connpass.com/event/195410/

おまえ、誰よ？　　辻 真吾（www.tsjshg.info)

- Pythonを使ったデータ解析などを中心にいろいろやってます
- このStart Python Clubをみんなで主宰しています

プログラミング初心者向け
Python入門書

アルゴリズムの教科書だけど
Python力を向上させることを裏目標にして書いた

もくじ兼おわび

- アルゴリズムとは？
- 数あてゲームと計算時間
- グラフを扱うアルゴリズムとデータ構造
- 最近出版された本の話
- （おわび）全体的に小難しい話になってしまいましたので、アンケートに率直なご意見いただけますと幸いです

アルゴリズムとは？

ある問題を解くための一連の手順

複雑な問題を、簡単な計算とその繰り返しを使って、より速く解く

単純な数あてゲーム

コンピュータが事前に決めた1〜16までの数字をあてるゲーム

ルール：コンピュータは質問にYes/Noで応えてくれる

どのような戦略が考えられるでしょうか？

戦略1：思い付いた数字をあたるまで言う（しかも言ったことを忘れる）

戦略2：1から順番にあたりがでるまで聞いていく

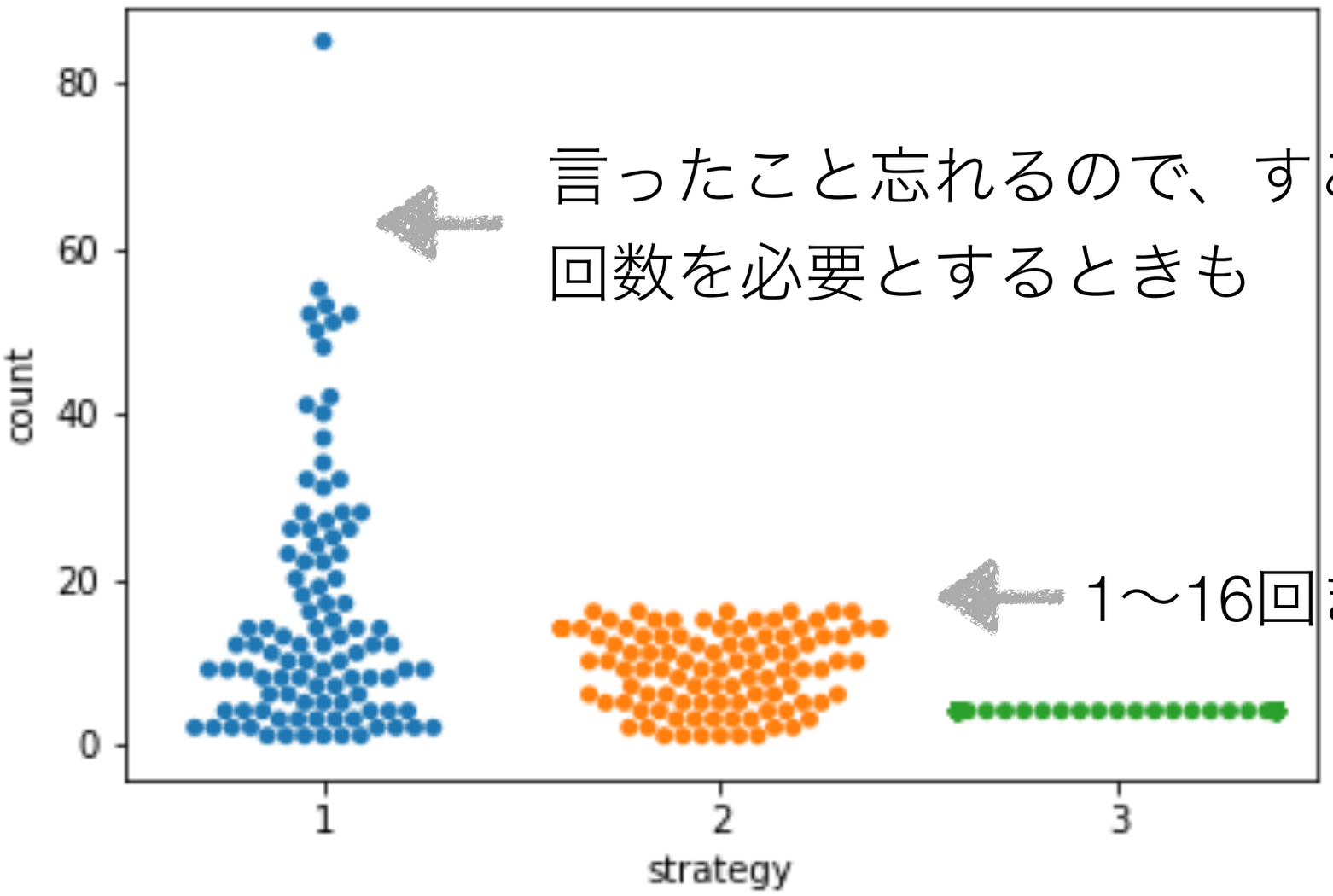
戦略3：あたりの候補を半分に絞っていく

戦略3：あたりの候補を半分に絞っていく（二分探索）

計算時間（問い合わせ回数）

1～16(=2^4)までの数を何回であてられたか

100回試してプロットした図

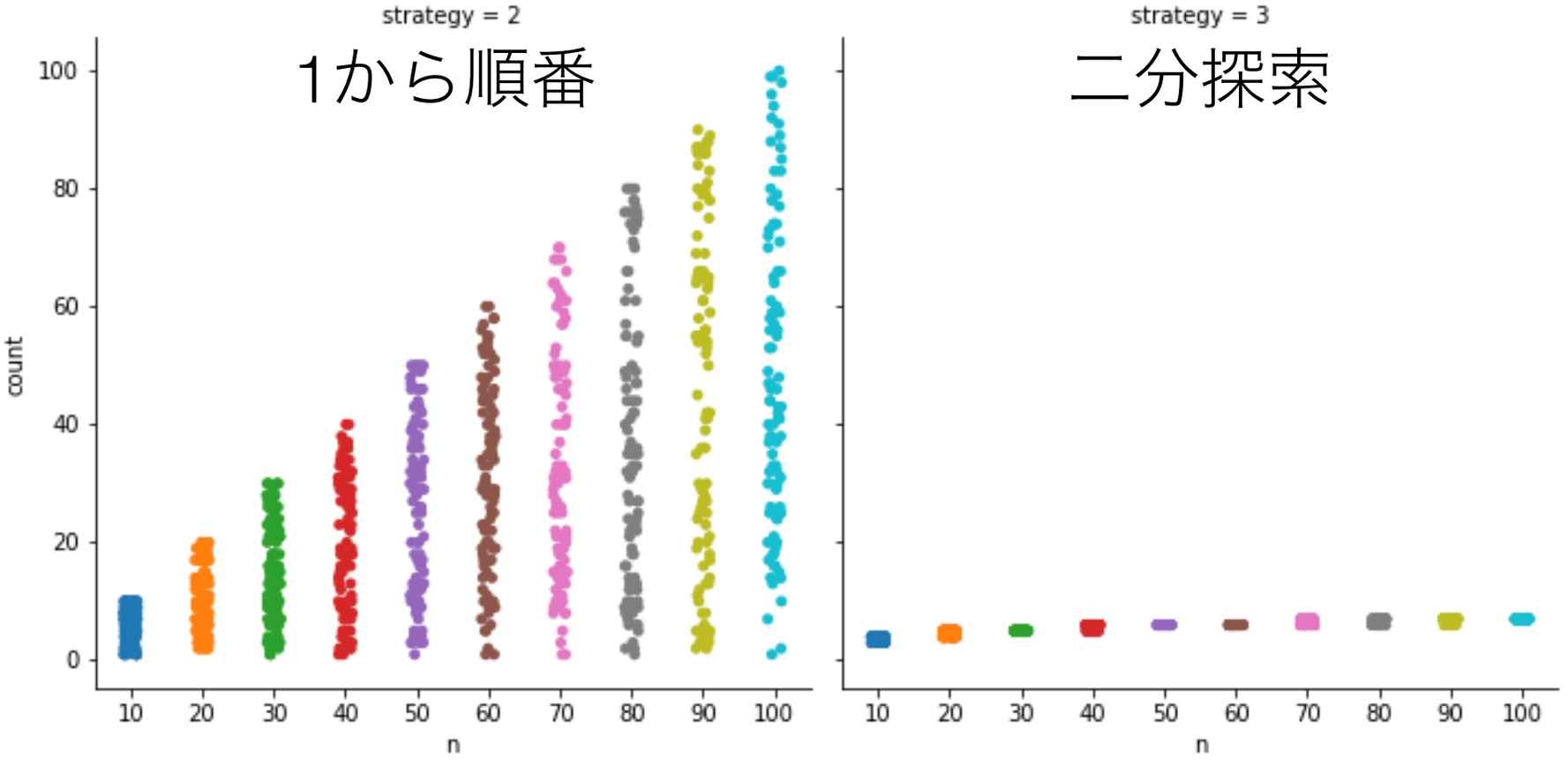


言ったこと忘れるので、すごい回数を必要とするときも

1～16回までだいたい同じ

常に4回

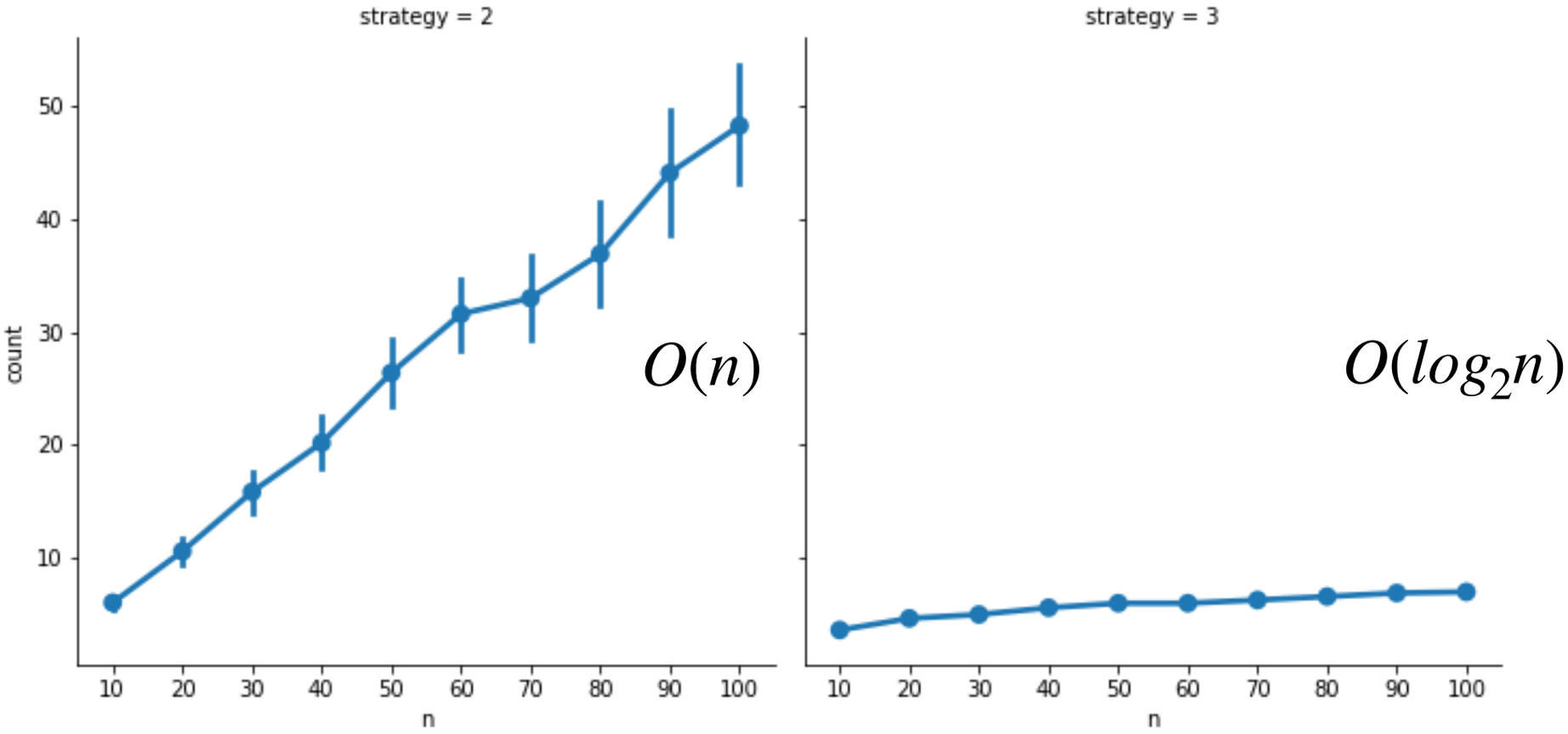
ランダム　　1から順番　　二分探索



1から順番

二分探索

選ぶ数字の
最大値を
10〜100まで
10刻みで
変化させた図

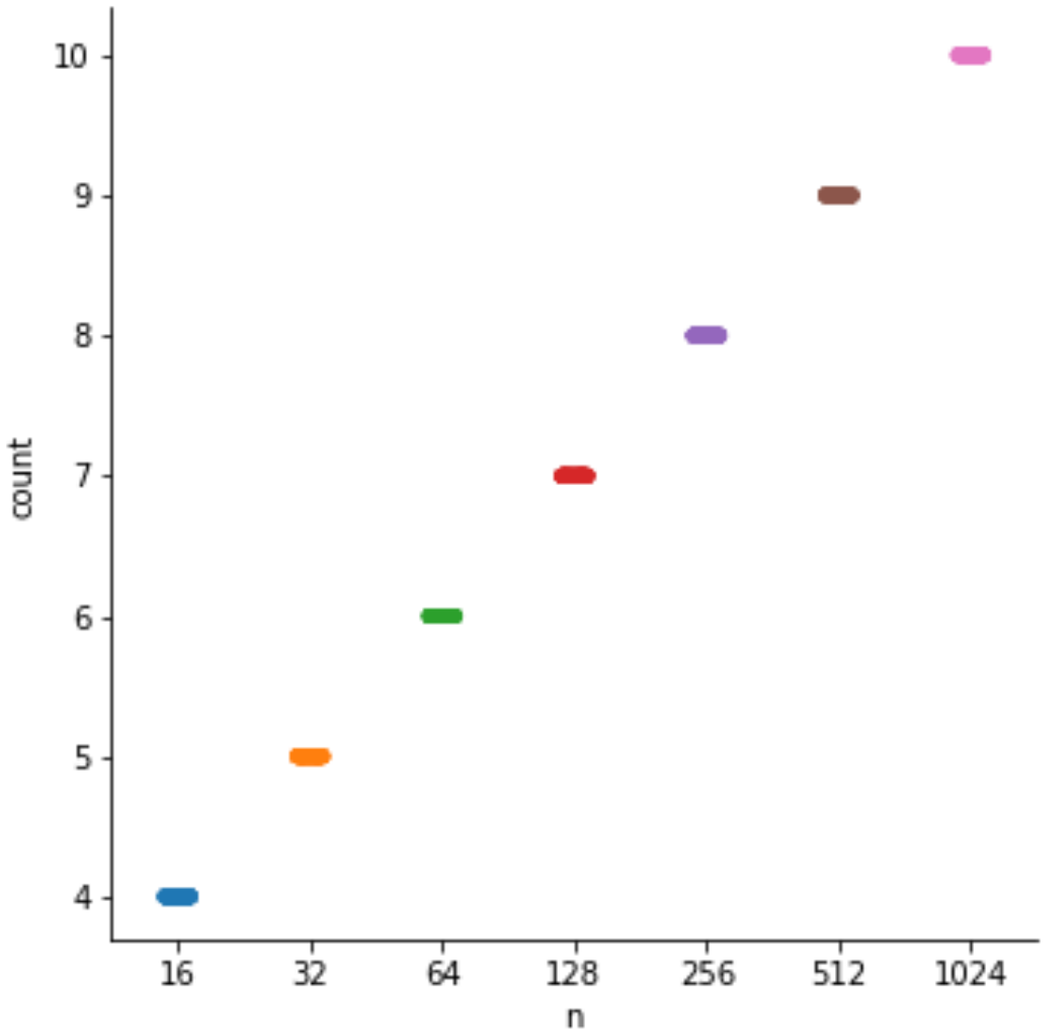


データの
数が増えると
アルゴリズム
の違いが
効いてくる

n を16から2倍ずつ増やしていくと・・・

二分探索

$O(log_2 n)$



対数ってなんだっけ？

2倍になると1増える

4倍になると2増える

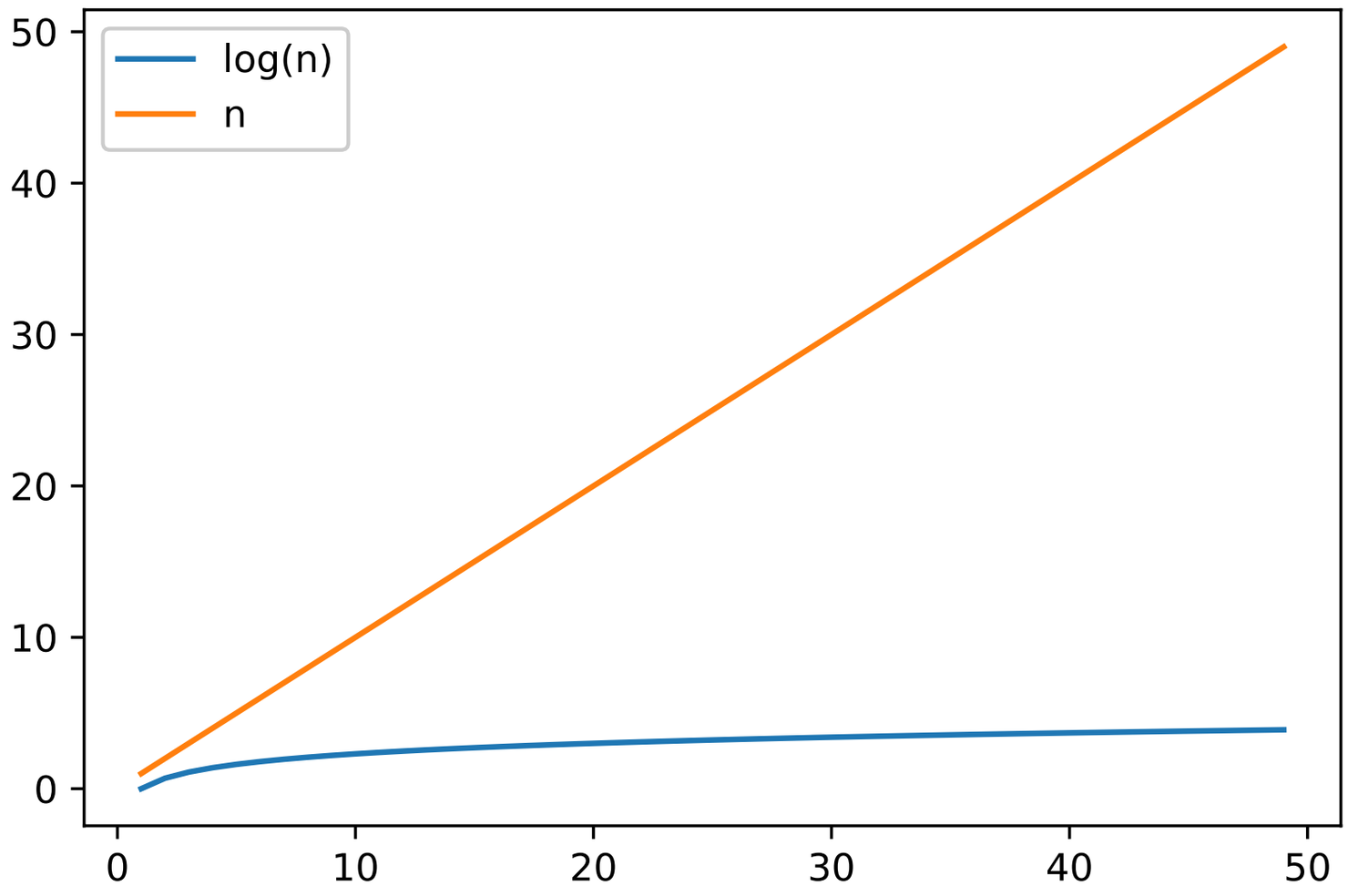
8倍になると3増える

つまり

2^a倍になるとa増える演算

ここでのaを底という

あらためてnと$log_2(n)$を比べてみると



入力が2倍になると出力も2倍になる$O(n)$と

入力が2倍になると1増える$O(log_2 n)$の違いは圧倒的

標準モジュールbisect

ソートされたリストと1つの値を引数に、ソートされた状態が崩れない挿入点が返される

```
import bisect

my_list = list(range(1, 17))
# 挿入する点が返ってくる
bisect.bisect(my_list, 6)
```

6

```
def bisect_right(a, x, lo=0, hi=None):
    """Return the index where to insert item x in list a, assuming a is sorted.

    The return value i is such that all e in a[:i] have e <= x, and all e in
    a[i:] have e > x.  So if x already appears in the list, a.insert(x) will
    insert just after the rightmost x already there.

    Optional args lo (default 0) and hi (default len(a)) bound the
    slice of a to be searched.
    """

    if lo < 0:
        raise ValueError('lo must be non-negative')
    if hi is None:
        hi = len(a)
    while lo < hi:
        mid = (lo+hi)//2
        # Use __lt__ to match the logic in list.sort() and in heapq
        if x < a[mid]: hi = mid
        else: lo = mid+1
    return lo
```

実際のコード

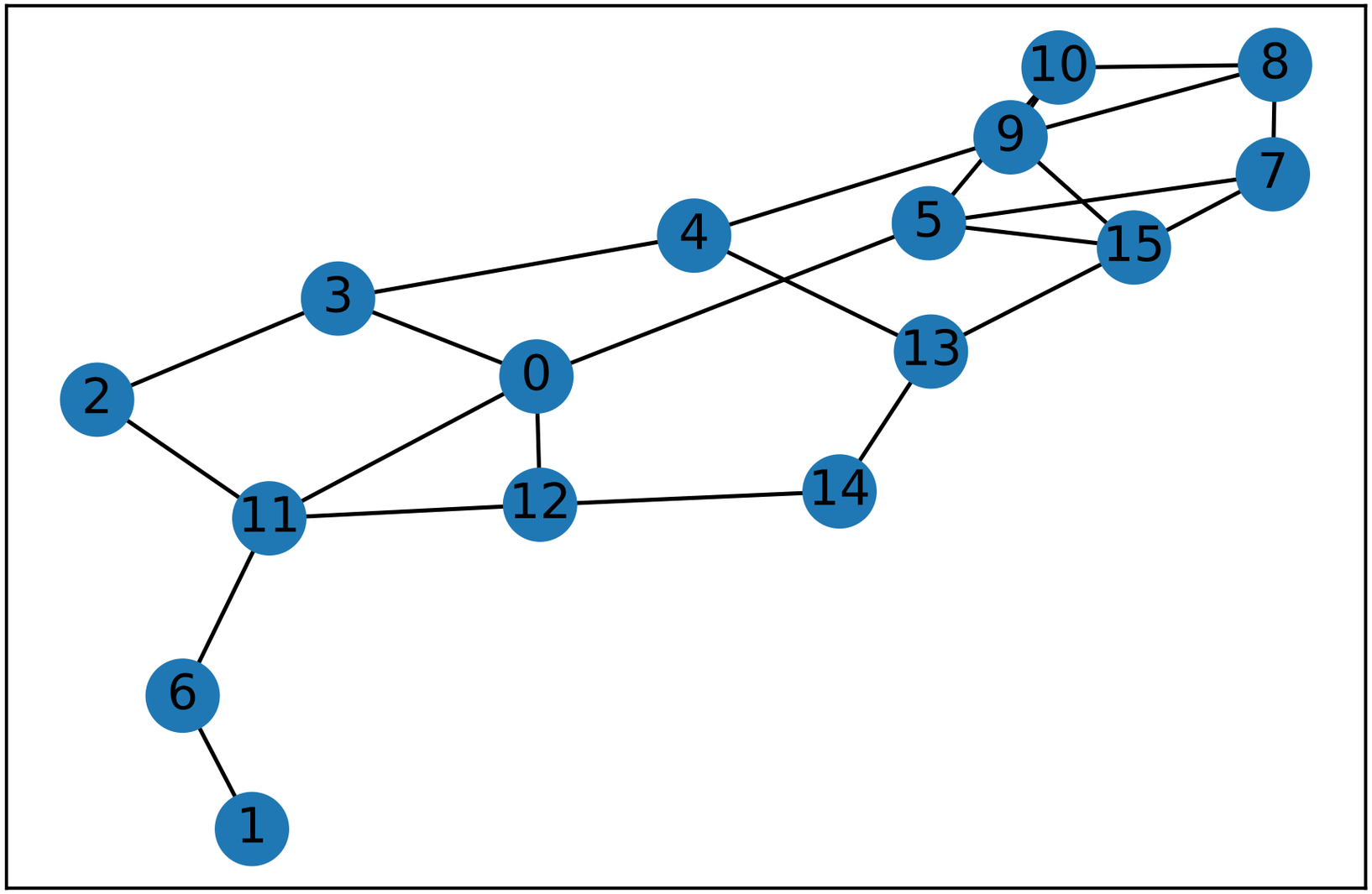
(C言語の実装が使われるのが普通)

アルゴリズムとデータ構造の切っても切れない関係

アルゴリズムと言えばグラフ（ネットワーク）

グラフはノード（頂点）とエッジ（辺）で表現される

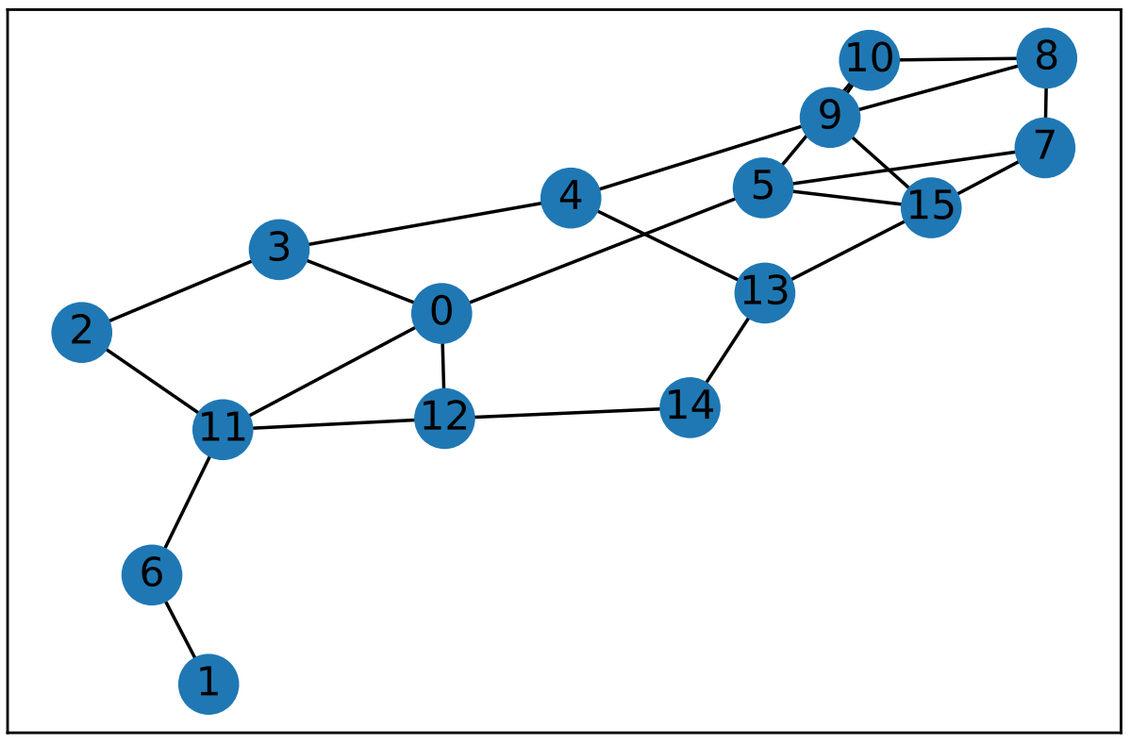
ノード1からスタートして全てのノードをもれなく訪問したい



どのような順番でノードを渡り歩いていったらよいだろうか？

アルゴリズムの大枠

1. いま居るノードを訪問済みとする
2. 訪問済みでないお隣さんをtodoに登録
3. todoリストが空っぽなら終了
4. todoリストのどれかに移動

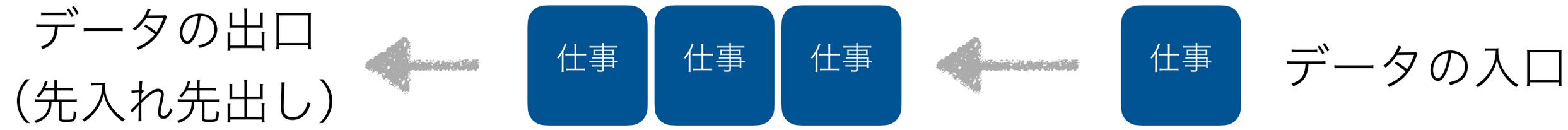


1→6→11のあと、

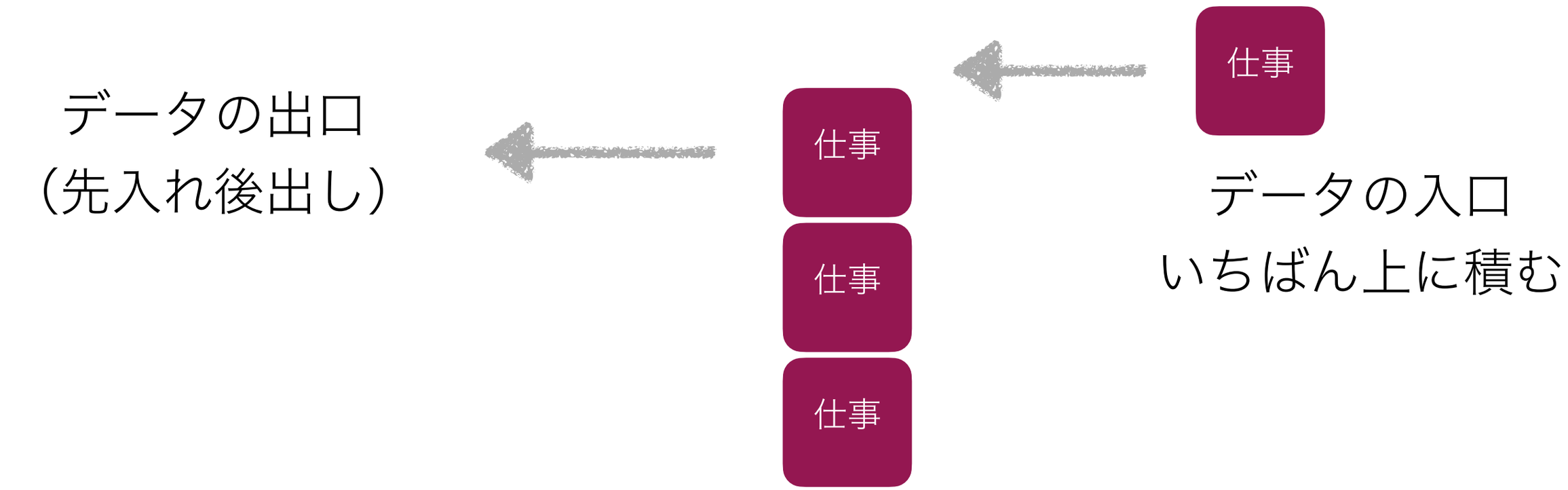
12, 0, 2とあるなかでどこへ行くのか？

たまった仕事をどの順番でやるか？

キュー：来た順番に仕事を並べる（FIFO: First in, first out）

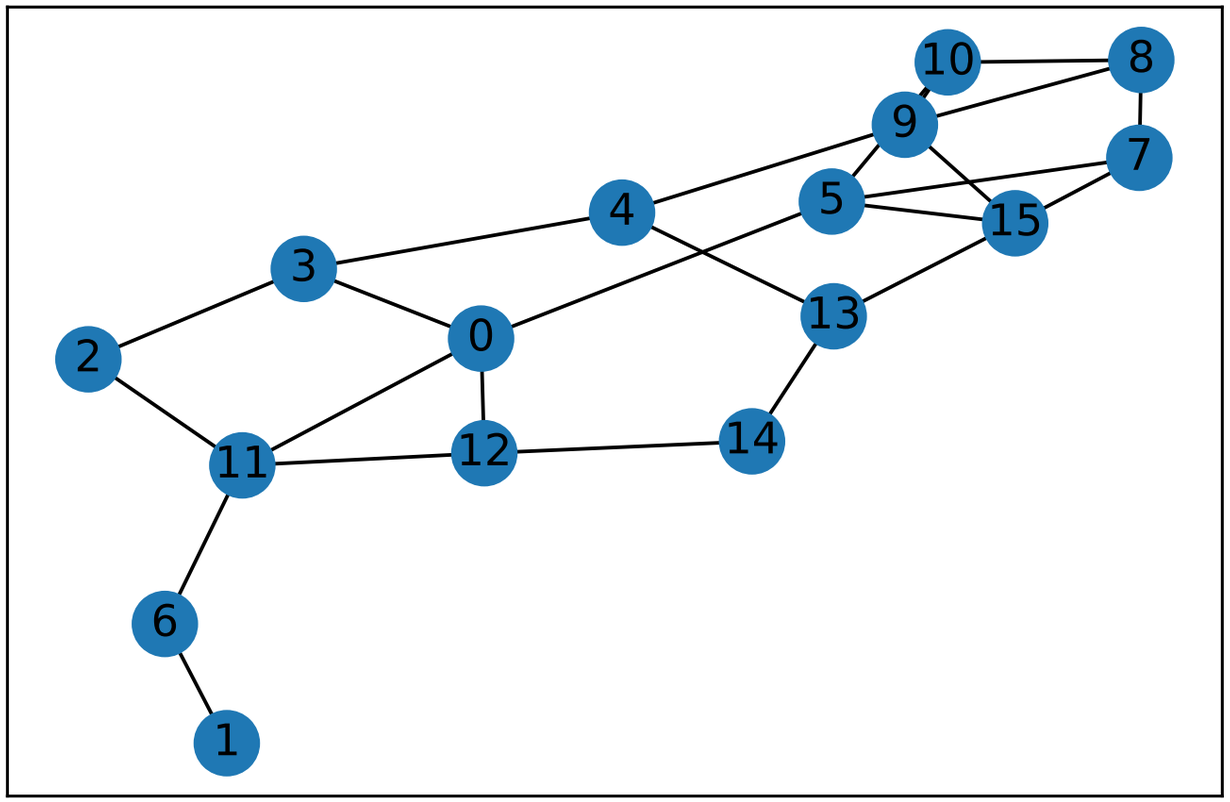


スタック：あたらしい依頼を上に積む（FILO: First in, last out）



深さ優先探索（DFS: depth-first search）

次に訪問するべきノードのtodoリストにスタックを使う方法



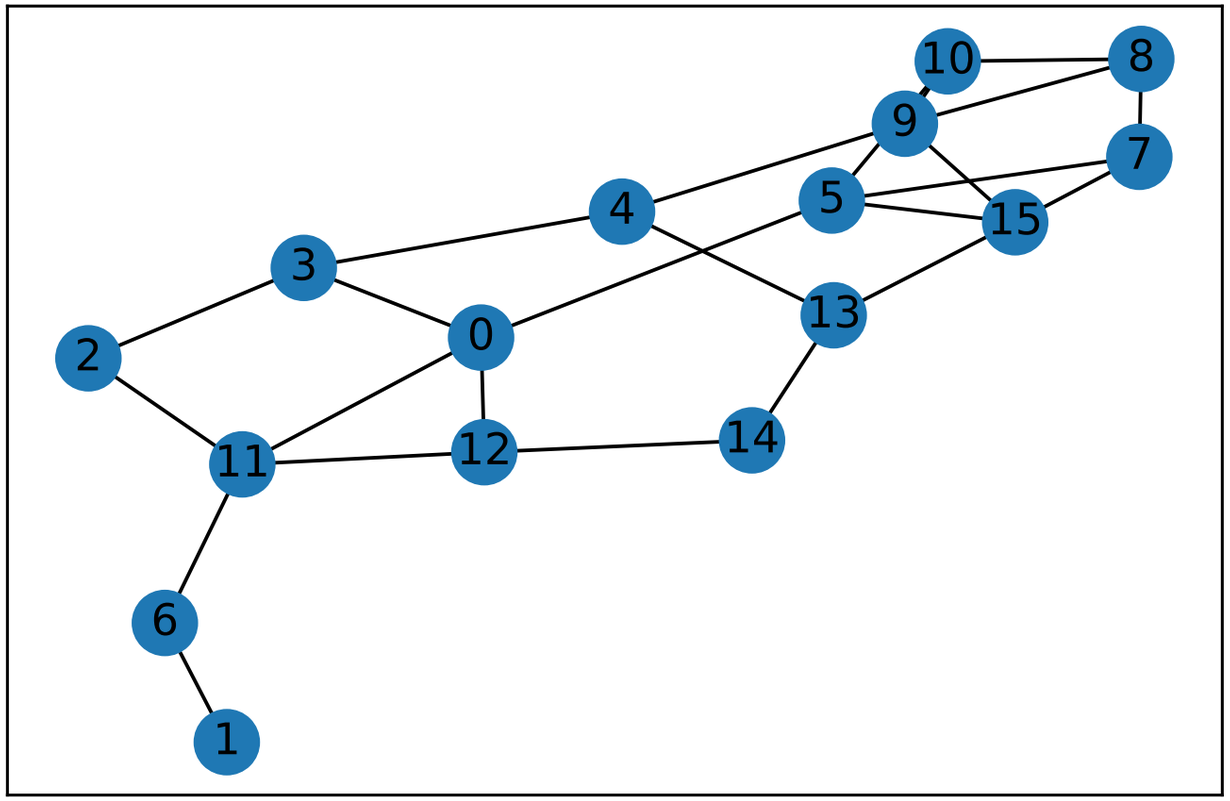
todoリストの中身

```
[1]
[6]
[11]
[0, 2, 12]
[0, 2, 14]
[0, 2, 13]
[0, 2, 4, 15]
[0, 2, 4, 5, 7, 9]
[0, 2, 4, 5, 7, 8, 10]
[0, 2, 4, 5, 7, 8]
[0, 2, 4, 5, 7]
[0, 2, 4, 5]
[0, 2, 4]
[0, 2, 3]
[0, 2]
[0]
```

todoリストの最新が訪問済みへ移動する

幅優先探索（BFS: breadth-first search）

次に訪問するべきノードのtodoリストにキューを使う方法



todoリストの中身

```
deque([1])
deque([6])
deque([11])
deque([0, 2, 12])
deque([2, 12, 3, 5])
deque([12, 3, 5])
deque([3, 5, 14])
deque([5, 14, 4])
deque([14, 4, 7, 10, 15])
deque([4, 7, 10, 15, 13])
deque([7, 10, 15, 13, 9])
deque([10, 15, 13, 9, 8])
deque([15, 13, 9, 8])
deque([13, 9, 8])
deque([9, 8])
deque([8])
```

todoリストの先頭が訪問済みへ移動する

PythonのlistをPythonを使えばどちらも簡単に実装できる

キュー

先入れ先出し

FIFO: First in, first out

```
# キューをlistで実装
my_list = [1, 2, 3]
# appendで末尾に要素を追加
my_list.append(4)
# pop(0)で先頭から要素を取り出す
my_list.pop(0)
```

```
1
```

スタック

先入れ後出し

FILO: First in, last out

```
# スタックをlistで実装
my_list = [1, 2, 3]
# appendで末尾に要素を追加
my_list.append(4)
# popで末尾から要素を取り出す
my_list.pop()
```

```
4
```

この実装にはちょっと問題が・・・

pop()とpop(0)は同じじゃない

1万要素をすべてpopする

```
import timeit

def pop_list(last=True, n=10000):
    my_array = list(range(n))
    while my_array:
        if last:
            my_array.pop()
        else:
            my_array.pop(0)
```

```
timeit.timeit('pop_list()', globals=globals(), number=5000)
```

```
3.496345214021858
```

```
timeit.timeit('pop_list(False)', globals=globals(), number=5000)
```

```
28.618284658994526
```

末尾からやるより、先頭からやる方が遅い

キューの実装にはdeque（デック）を使う

```
from collections import deque

def pop_deque(last=True, n=10000):
    my_array = deque(range(n))
    while my_array:
        if last:
            my_array.pop()
        else:
            my_array.popleft()
```

```
timeit.timeit('pop_deque()', globals=globals(), number=5000)
```

```
3.071630149839446
```

```
timeit.timeit('pop_deque(False)', globals=globals(), number=5000)
```

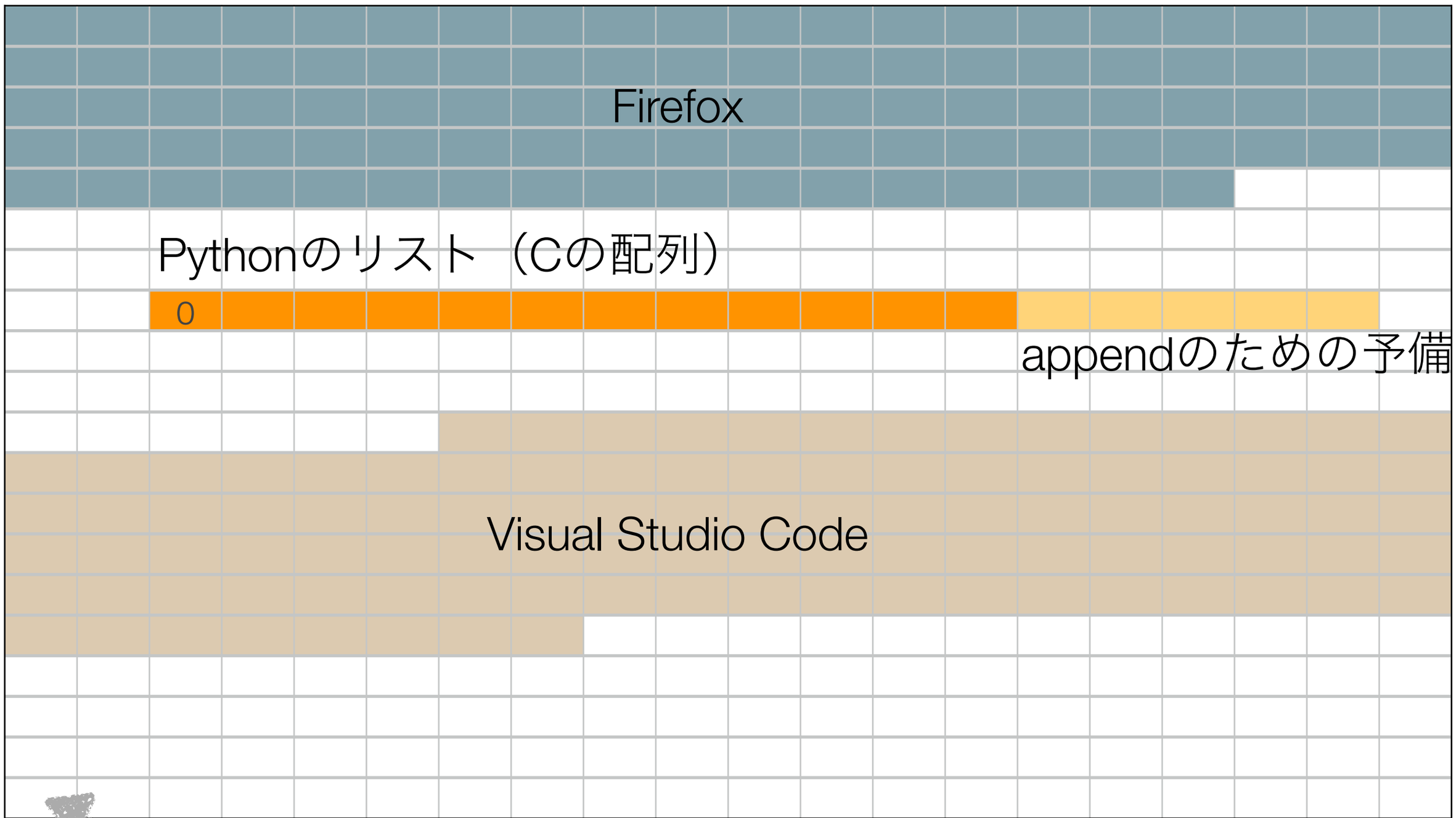
```
2.998329053982161
```

どちらからpopしてもほとんど同じ実行時間になる

グラフのアルゴリズムを実装するときデータ構造の知識が必要

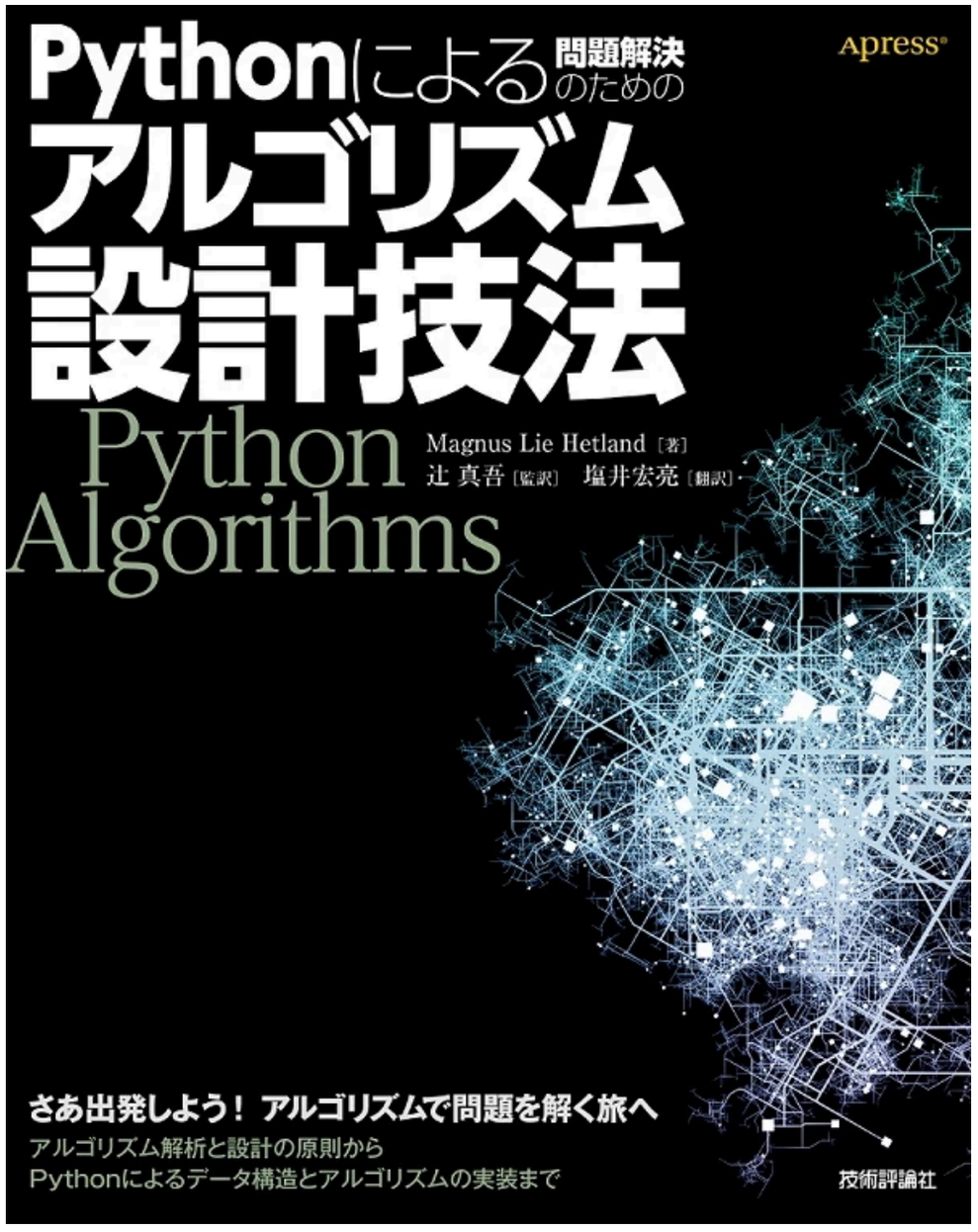
どうしてそうなるのか？

メモリ空間を覗いてみよう（イメージ図です）



Pythonによる問題解決のためのアルゴリズム設計技法

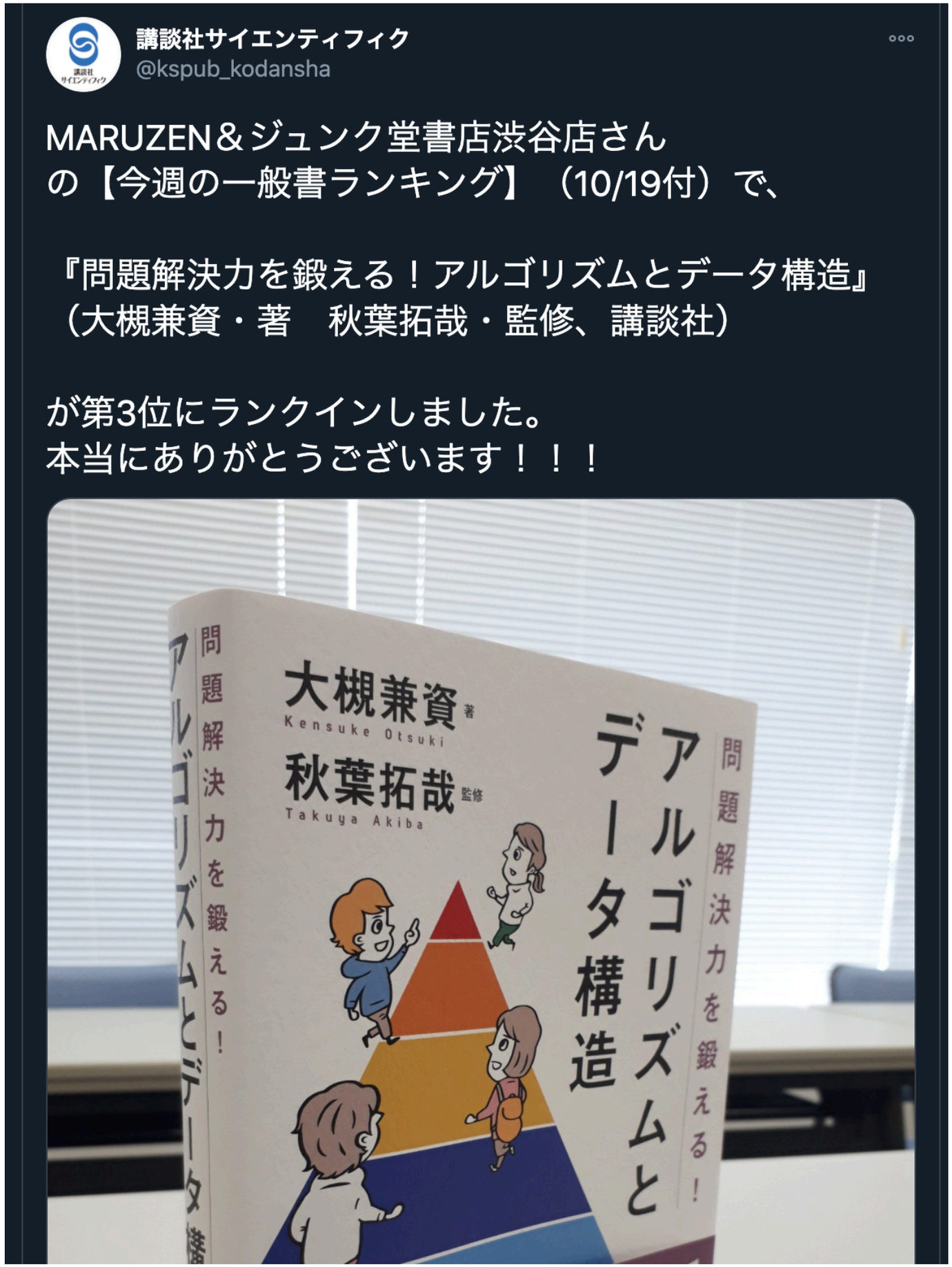
- 監訳とやりました
    - わたしのまえがきとサンプルPDFがgihyo.jpのサイトにあります
- 基本的なアルゴリズムの紹介だけではなく、自分でアルゴリズムを設計できることを目指すための本
    - 結構難しいので入門書のあとがおすすめ
- 翻訳の塩井さんとは、このStapyで知り合いました
- 3冊プレゼント企画あります！



https://gihyo.jp/book/2020/978-4-297-11686-6

アルゴリズムとデータ構造が密かなブーム？

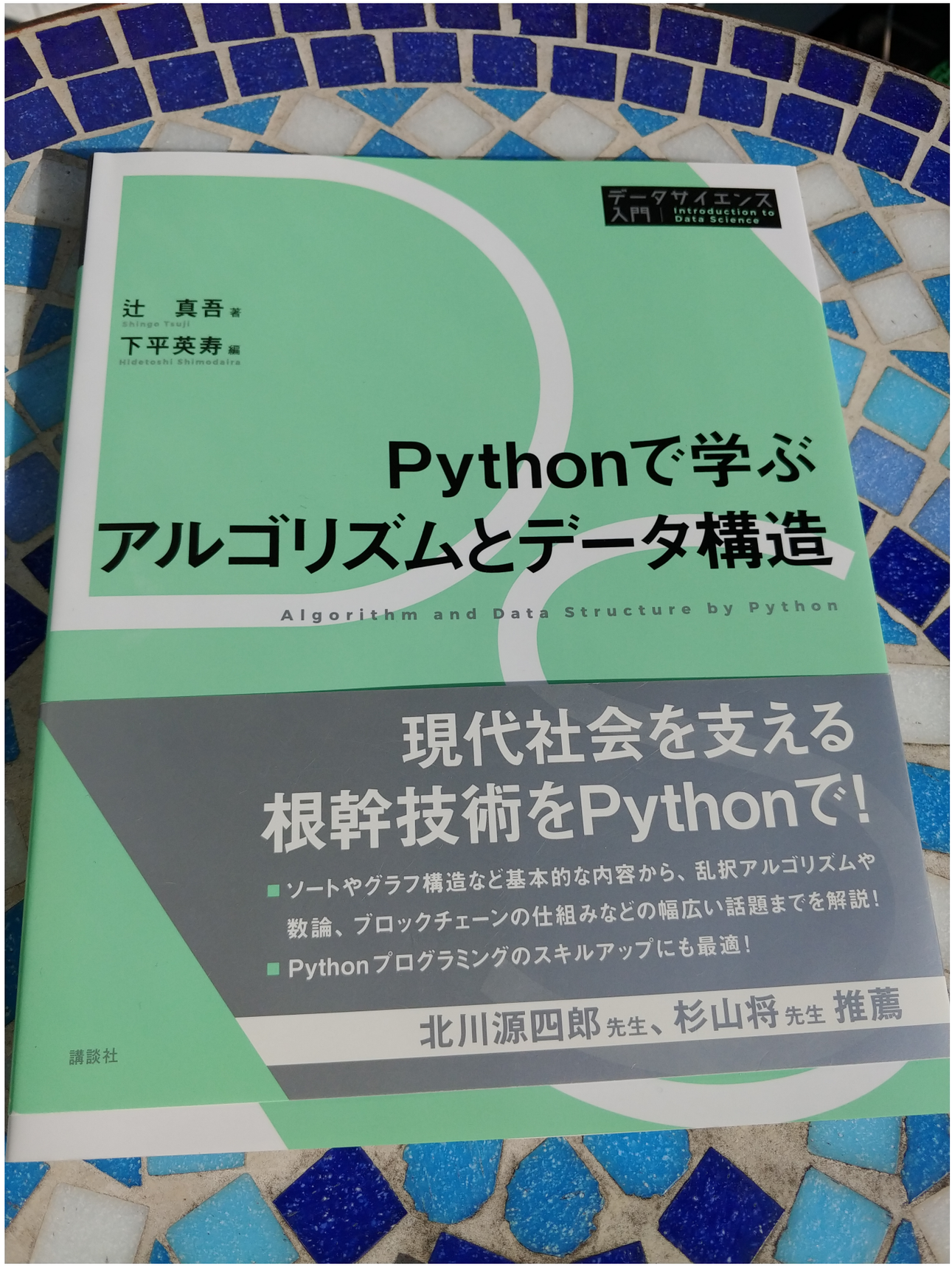
- けんちょんこと大槻兼資さんの本
  - 大型書店の一般書で3位って、売れすぎじゃ・・・
- カラーで見やすいイラストも多くアルゴリズムの入門書としてはおすすめ
  - サンプルコードはC++
- 大槻さんと塩井さんが大学時代の同期という偶然も



https://twitter.com/kspub_kodansha/status/1318344525341888512

でもほんとはこれを買って欲しい

- サンプルコードはもちろんPython！
  - Githubでコードと演習問題の解答公開しています
  - github.com/tsjshg/pyalgdata
- 教科書シリーズの1冊なので、である調で書かれています
  - なので表紙も地味
- 類書は多いので、いろいろ探すとたのしいかも



ご静聴有り難うございました。
また、今年も大変お世話になりました。また来年
もよろしくお願いいたします。