

Python勉強会@HACHINONE

第10章

いくつかの単純な
アルゴリズムとデータ構造

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。

アルゴリズム

Python勉強会@HACHINOHE

- 探索
 - 線形探索
 - リストの実装
 - ソート済みリストの二分探索
- ソート
 - 選択ソート
 - マージ・ソート
- ハッシュ

線形探索

Python勉強会@HACHINOHE

- 線形探索: 前から順番に探す
- L[i]の実行が定数時間なら、線形時間

```
# -*- coding: utf-8 -*-  
def search(L, e):  
    """Lをリスト、eをオブジェクトとする  
    Lにeが含まれていればTrue、そうでなければFalseを返す"""  
    for i in range(len(L)):  
        if L[i] == e:  
            return True  
    return False
```

Lがすべて整数

Python勉強会@HACHINOHE

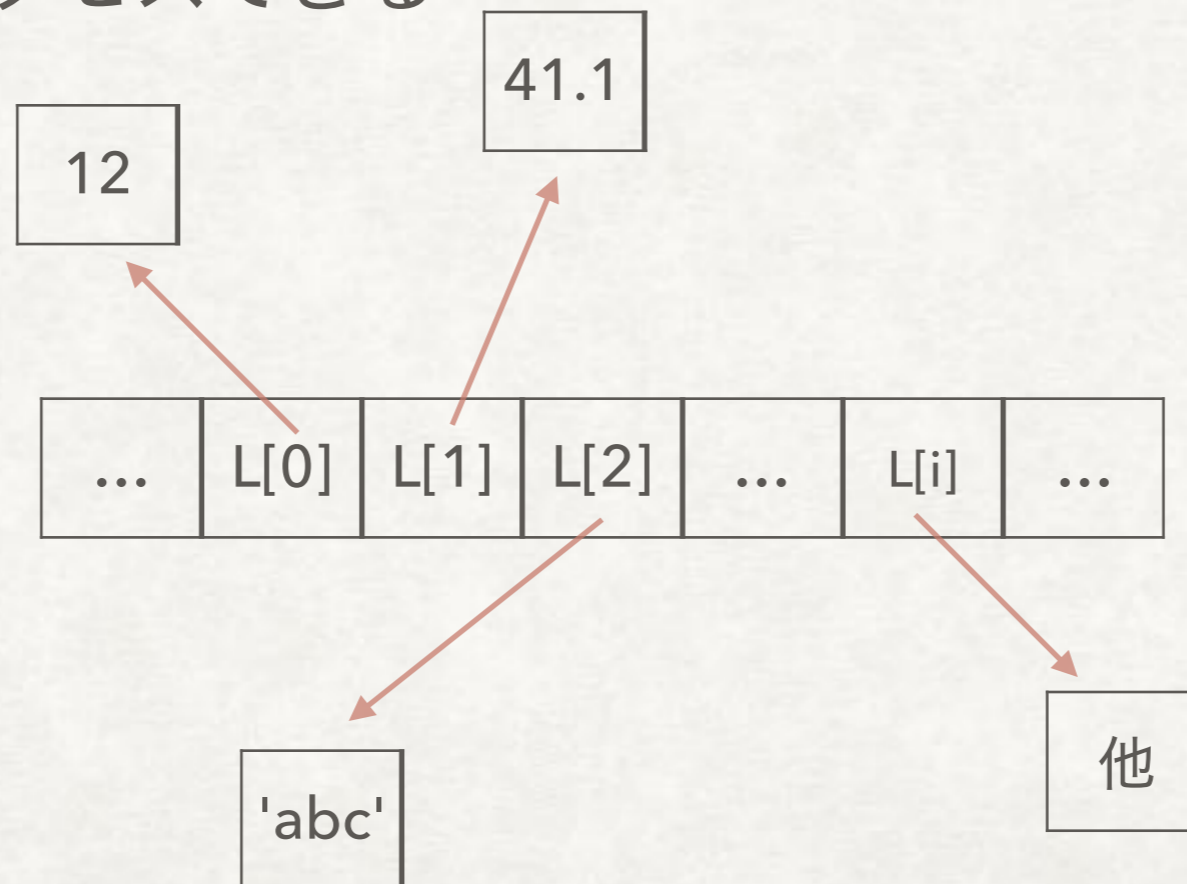
- Lがすべて整数(4バイト)で、順番にメモリに入っているとする
- $L[i]$ は「Lの最初の位置+ $4 \times i$ 」で位置が計算でき、これは定数時間



Pythonのリスト

Python勉強会@HACHINOHE

- Pythonのリストにはさまざまなデータが入れられる
- 実装としては、リストはポインタで、各要素の実体が保存されている位置が入っている(間接参照)
- 定数時間でアクセスできる



要素がソート済みの線形探索

Python勉強会@HACHINOHE

- 要素がソートされていれば、途中で打ち切ることもできる

```
# -*- coding: utf-8 -*-
def search(L, e):
    """Lをリスト、eをオブジェクトとする
    Lにeが含まれていればTrue、そうでなければFalseを返す"""
    for i in range(len(L)):
        if L[i] == e:
            return True
        if L[i] > e:
            return False
    return False
```

要素がソート済みの2分線索

Python勉強会@HACHINOHE

- 要素がソートされていれば、2分探索できる

```
# -*- coding: utf-8 -*-
def search(L, e):
    """Lが昇順にソートされた数値のリスト、eを数値とする
    Lにeが含まれていればTrue、そうでなければFalseを返す"""
    def bSearch(L, e, low, high):
        # high - lowを減少させる
        if high == low:
            return L[low] == e
        mid = (low + high) // 2
        if L[mid] == e:
            return True
        elif L[mid] > e:
            if low == mid: # 探索対象は残っていない
                return False
            else:
                return bSearch(L, e, low, mid - 1)
        else:
            return bSearch(L, e, mid + 1, high)

    if len(L) == 0:
        return False
    else:
        return bSearch(L, e, 0, len(L) - 1)
```

実行は定数時間で
再帰呼び出し回数が
計算量を決める。
半分ずつに減るので
最大 $\log_2(\text{high} - \text{low})$

ソートして二分探索の計算量

Python勉強会@HACHINOHE

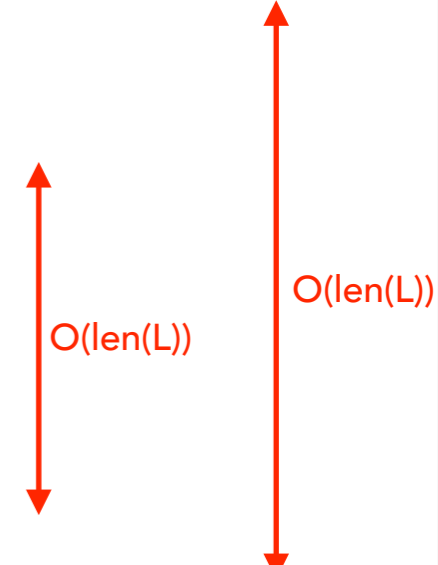
- 線形探索: $\text{len}(L)$
- ソートして二分探索: ソート計算量 + $\log(\text{len}(L))$
 - ソートするには、少なくとも全要素を見る必要があるため、線形探索よりもソート計算量は大きい
- 何回も探索するならば、ソートする意味がある
- Pythonはタイム・ソートで $O(n \log n)$

選択ソート

Python勉強会@HACHINOHE

- 最初に一番小さいのを、2番目に次に小さいのを、と探索して入れていく、 $O(\text{len}(L)^2)$

```
# -*- coding: utf-8 -*-
def selSort(L):
    """Lは「>」で比較できる要素からなるリストとする。
    Lを昇順にソートして返す"""
    suffixStart = 0
    while suffixStart != len(L):
        # サフィックスの各要素を見る
        for i in range(suffixStart, len(L)):
            if L[i] < L[suffixStart]:
                # 各要素の位置を入れ替える
                L[suffixStart], L[i] = L[i], L[suffixStart]
        suffixStart += 1
```



Wikipediaの例

Python勉強会@HACHINOHE

初期データ: 8 4 3 7 6 5 2 1

太字はソート完了した部分

1 4 3 7 6 5 2 8 (1回目のループ終了時)

1 2 3 7 6 5 4 8 (2回目のループ終了時)

1 2 3 7 6 5 4 8 (3回目のループ終了時)

1 2 3 4 6 5 7 8 (4回目のループ終了時)

1 2 3 4 5 6 7 8 (5回目のループ終了時)

<https://ja.wikipedia.org/wiki/選択ソート>

マージ・ソート:1

Python勉強会@HACHINOHE

- マージ関数

```
# -*- coding: utf-8 -*-
def merge(left, right, compare):
    """leftとrightはソート済みのリスト、compareは順序を定義する関数。
    leftとrightを合わせてcompareでソートしたリストを返す"""
    result = []
    i, j = 0, 0
    while i < len(left) and j < len(right):
        if compare(left[i], right[j]):
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    while (i < len(left)):
        result.append(left[i])
        i += 1
    while (j < len(right)):
        result.append(right[j])
        j += 1
    return result
```

マージ・ソート: 2

Python勉強会@HACHINOHE

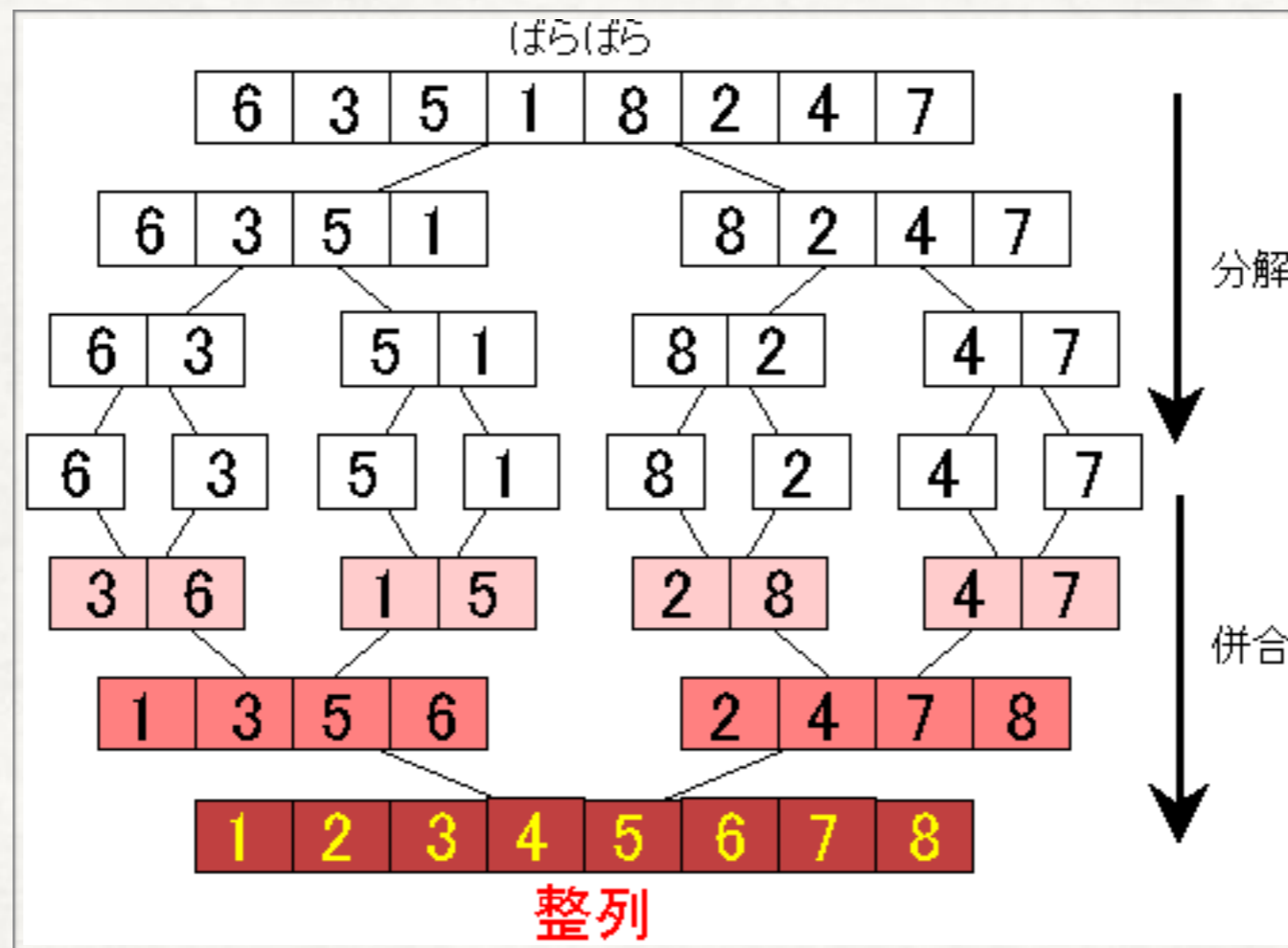
- 本体

```
# -*- coding: utf-8 -*-
import operator
def mergeSort(L, compare = operator.lt):
    """Lをリスト、compareを要素の順序を定義する関数とする。
    ソートされたリストを返す"""
    if len(L) < 2:
        return L[:]
    else:
        middle = len(L) // 2
        left = mergeSort(L[:middle], compare)
        right = mergeSort(L[middle:], compare)
        return merger(left, right, compare)
```

マージ・ソート: アルゴリズム

Python勉強会@HACHINOHE

- ばらして、比較しながら結合していく



<http://www.ics.kagoshima-u.ac.jp/~fuchida/edu/algorithm/sort-algorithm/merge-sort.html>

ハッシュ:1

Python勉強会@HACHINOHE

```
class intDict(object):
    """整数をキーとする辞書"""

    def __init__(self, numBuckets):
        """空の辞書生成する"""
        self.buckets = []
        self.numBuckets = numBuckets
        for i in range(numBuckets):
            self.buckets.append([])

    def addEntry(self, dictKey, dictVal):
        """dictKeyをint型とし、エントリを追加する"""
        hashBucket = self.buckets[dictKey%self.numBuckets]
        for i in range(len(hashBucket)):
            if hashBucket[i][0] == dictKey:
                hashBucket[i] = (dictKey, dictVal)
                return
        hashBucket.append((dictKey, dictVal))

    def getValue(self, dictKey):
        """dictKeyをint型とする
        キーdictKeyに関連付けられたエントリを返す"""
        hashBucket = self.buckets[dictKey%self.numBuckets]
        for e in hashBucket:
            if e[0] == dictKey:
                return e[1]
        return None

    def __str__(self):
        result = '{'
        for b in self.buckets:
            for e in b:
                result = result + str(e[0]) + ':' + str(e[1]) + ','
        return result[:-1] + '}' #result[:-1]により最後のカンマを省く
```

ハッシュ:2

Python勉強会@HACHINOHE

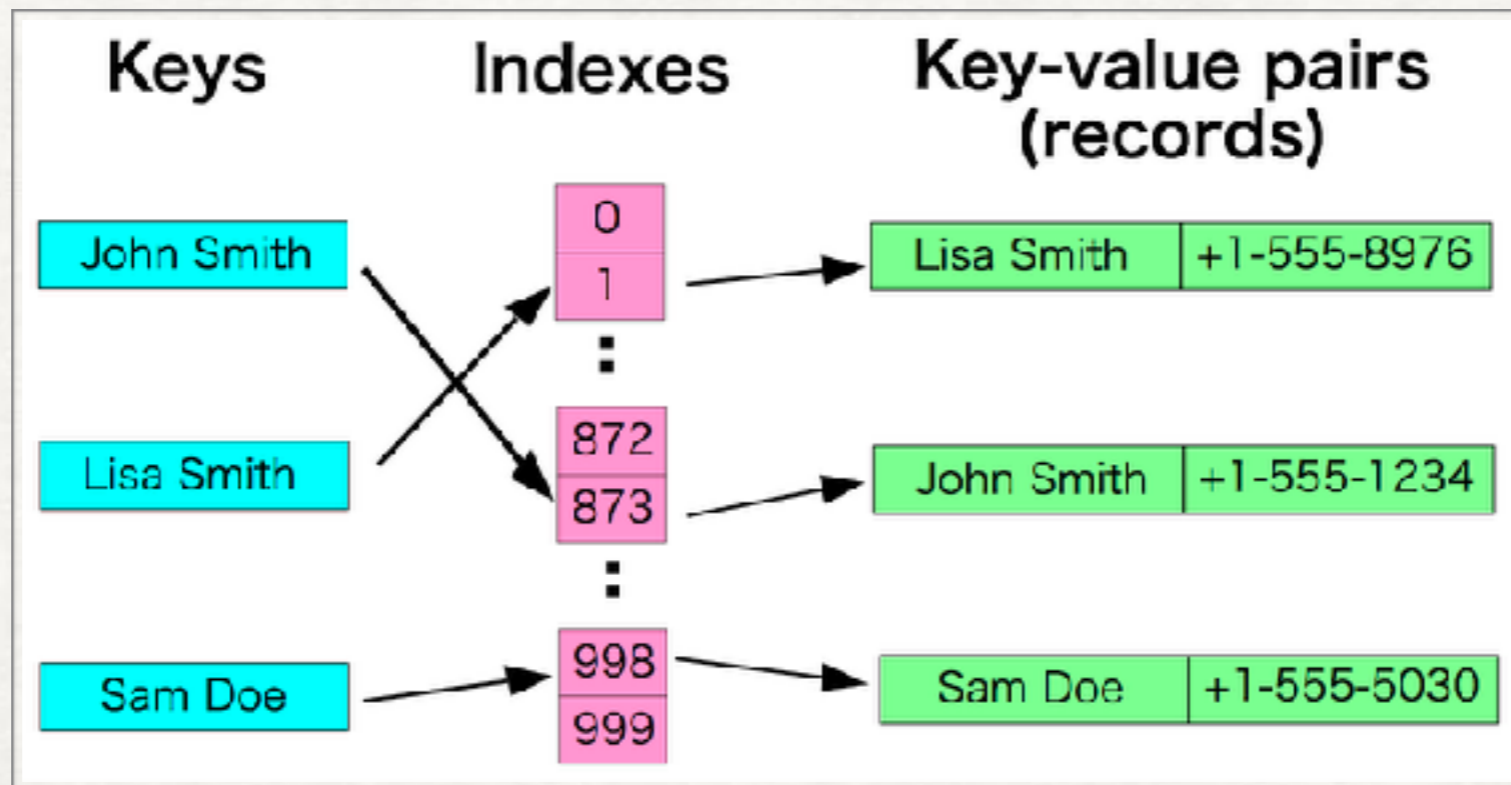
```
import random #標準ライブラリモジュール

D = intDict(29)
for i in range(20):
    #0から10**5までの整数をランダムに選ぶ
    key = random.randint(0, 10**5)
    D.addEntry(key, i)
print 'intDictの値:'
print D
print '\n', 'バケット:'
for hashBucket in D.buckets: #抽象化の壁を侵す
    print ' ', hashBucket
```


ハッシュ表: アルゴリズム

Python勉強会@HACHINOHE

- キーからハッシュ値を計算
- ハッシュ値を添字とするリストを作り、そこに値を格納
- キーからハッシュ値を計算するのが簡単で、異なるキーから異なるハッシュ値が生成されるかがポイント



パブリック・ドメイン, <https://commons.wikimedia.org/w/index.php?curid=441890>