Algorithms (2020 Summer)
#1：イントロダクション，計算量

矢谷 浩司

Welcome!

今年度よりこの講義を担当する矢谷浩司です.

この講義は相田先生と共同で担当になりますが，授業は主に私が担当します.

アルゴリズムとは？

アルゴリズムとは

あるタスクを達成するために設計された有限回の計算手順（ソート，サーチ，最適化などなど）．

アルゴリズムは正しい，あるいは「最適な」解を導くように設計されている．（ただし，最適と言っても，あくまでアルゴリズム内で設定された評価基準による）．

多くの場合は（時間コスト，メモリコスト，通信コストにおいて）効率的な計算手順を意味する．

アルゴリズムの例

文章の中から所望の単語がある場所を探し出す.

たくさんの数字を小さいもの順に並べる.

A駅からB駅に行くまでの電車でのルートを検索し,
運賃の安い順に並べる.

進学選択で各学生さんの配属学科を決める.

アルゴリズムがわかると

効率的な処理を設計できる.

与えられた処理がどの程度計算を要するものなのかを見積もることができる.

ボトルネックになっているコードを解析してそこを改善できる.

例えるなら

コードが書ける　＝　調理ができる

アルゴリズムがわかる　＝　手早く美味しい料理が作れる

どうせなら美味しいもの食べたいよね！

この講義でやること

データ構造やアルゴリズムに関する基礎知識の学習

それらをpythonで実装し，実際に体験

さらにそれらを利用して，演習課題に取り組む

この講義で学んでほしいこと:

頭で考えた処理手順をコードに落とし込む

この講義を終えた後にはこうなってほしい

データ構造やアルゴリズムに関する基礎的な話が
わかっている

それらをpythonで実装することができる

それらを応用するような課題に自分で取り組める
（例えば，競技プログラミングなど）

今年度から新しくなりました！

矢谷が新しく担当．よろしくお願いします．

手を動かすことを重視したスタイルに大幅変更．

一般的なアルゴリズムの授業でカバーする内容に加えて，
競技プログラミングも少し意識した内容に変更．

この講義の新モットー:

Let’s code! 

講義内容を刷新するにあたって

アルゴリズムを初めて学ぶ人が大部分，という前提です.

コーディングを得意としない人にもある程度は理解してもらえるように噛み砕いてスライドを作ったつもりです.

力のある方には少し物足りない授業かもしれませんが，その分コードチャレンジ（後で説明）で遊んでもらえるようになっていると思います.

講義体制

講義担当者：相田 仁，矢谷 浩司（私）
TA：杉山 悠司，鈴木 凌斗

ホームページ：https://iis-lab.org/algorithms
メールアドレス：algorithms@iis-lab.org（矢谷＆TA）

基本は講義のslackにてコミュニケーションをお願いします．

講義の前提条件

使用言語：Python3
コーディング環境：track（後で説明します）

学科PCか自前のPCを持参し，ネットワークに接続できる状態にしておいてください．

ローカル環境でもpythonを実行できるようにしておいてもらえるとよいです（必須ではありません）．

講義の前提条件

プログラミング，pythonの基礎的内容は身についているものとします.

変数の型，配列，条件分岐，ループ，関数，
再帰，入出力などなど.

このあたりが不安な人は今のうちに自習をお願いします.

授業の構成

50 – 60分：データ構造，アルゴリズムに関する講義形式の説明

45 – 55分：コードチャレンジ（ハンズオン形式）

全てオンラインで行いますので，講義視聴およびコーディングに必要なPC環境とネットワーク環境を確保してください．

講義パート

スライドを使って説明します．スライドは授業のホームページにもアップロードしておきます．

授業開始時刻にZoom webinarを開始します．終了後も後から視聴できるようにしておきますので，見逃した方はお時間のある時に自習してください．

環境によりZoom webinarを視聴できない人はTAさんまで連絡してください．

コードチャレンジ

基本課題：全員できてほしい課題

講義内で紹介された擬似コードの実装，変更.
授業の内容を追えばできる（はず）.

Extra**課題：頑張ればできる腕試し的課題**

授業の内容を踏まえた発展的な課題.
競技プログラミングとかにも出てきそうな内容.

Class structure

#0: プレイントロ（4/8）
#1: イントロダクション，計算量
#2: 累積和，データ構造
#3: 探索（サーチ）
#4: 文字列照合
#5: 整列（ソート）
#6: 動的計画法1
#7: 動的計画法2

Class structure

#8: 整数関連
#9: BFS，DFS
#10: グラフアルゴリズム1
#11: グラフアルゴリズム2
#12: グラフアルゴリズム3

（特別イベントによるスケジュール変更の可能性あり．
その場合は，別途お知らせいたします．）

成績評価方針

コードチャレンジ基本課題：35点

コードチャレンジExtra課題：35点

期末試験：30点

（ただし，状況に応じて点数配分を数点程度変更することがあります．また期末試験はレポート等に変更する可能性があります．）

成績評価方針

基本課題をこなして期末試験をそこそこ -> 良
Extraまである程度こなして期末試験も上出来 -> 優
Extraもしっかりこなして期末試験も立派 -> 優上

（あくまで目安です．相対評価なので．．．）

コーディングに自信のない人も基本課題はしっかり取り組んでください．そうすれば単位を落とすことはない（はず）．

出席

取りません. 😆

授業を視聴することは必須ではありませんが，頑張って配信しますので，見ていただけると嬉しいです．．．

出席せずとも，課題の提出はしてください.

病欠，公欠

以下の情報を**メールにて**，algorithms@iis-lab.orgに送ってください．認められた場合，課題提出期限をこちらが指定する日時まで延長します．

- 病欠，公欠を希望する授業日
- 病欠，公欠の理由
- 理由が正当なものであることを裏付けるもの（診断書，処方箋，出席する学会のプログラム等）

可能な限り事前に，事後でも速やかにお願いします．大きな怪我や病気などの例外を除き，病欠，公欠を希望する授業日から1週間以上を経過した場合は，認めないものとします．

質問

Zoom内のQ&Aを利用してください.

代表的な質問は次の回でも紹介したいと思います.

slack（#2020s-アルゴリズム）

講義のアナウンス等に利用します.

EEICのworkspace内にチャネルがありますので，そこにjoinしてください.

学科外の方は私から個別に招待を行っています．まだ登録されていない方はリマインドしてください.

講義の質問はここでは行わなず，zoomでお願いいたします.

Academic misconduct

この講義でも厳しく対処します.

剽窃，無断流用などが発覚した場合，その度合いに
応じて以下のような処置が取られます.

- 提出された課題，試験を0点とする.
- 提出された課題，試験を0点とし，以降の提出を認めない.
- 全課題，試験に遡って0点とし，以降の提出を認めない.

Academic misconduct

コードチャレンジでは皆さんに課題に対するコードを提出してもらいます.

Extra課題に関しては，提出されたコードに対して後日類似度チェック等を行い，明らかに類似したコードがあった場合は，その全てに対してペナルティを課します.

基本課題に関してはほぼ同じようなコードになるはずですので，このルールを直接は適用しませんが，チェックは行います.

Academic misconduct

講義で使うシステムではコードをテストケースにかけた時点で，自動でバージョン管理します．

コピペ等を行なった場合には，不自然にコード行数が増えたり，テストケースにパスする回数が突然増えたりするので，その辺りもチェックします．

Academic misconduct

本やWeb上の情報を参考にするのは構いません．ただし，そのまま使うことのないようにしてください．

友達同士で教え合うのも積極的にどうぞ．ただし，コードを直接見せ合うのではなく，考え方だけを共有するようにしてください．

「自分で手を動かす」ことをご自身で大切にしてください．

パケ死しそうな時は

講義のスライドは各回の開始前にアップロードします.

（一応）丁寧に作ったつもりですので，最悪講義の動画が視聴できなくても，ゆっくり読んでもらえれば理解してもらえるのではないかと思っています.

今後講義の視聴環境が制限され，受講していくことが非常に困難になることが予想される人は，私とTAさんまで，メール等にてご連絡ください.

デバッグにお付き合いください

今年度初めて取り入れたことがいっぱいあります.

- 授業構成・内容
- コードチャレンジ
- 完全オンライン化

皆さんにとって不利にならないように適宜配慮します.

随時皆さんに授業に関するアンケートをお願いします.
ぜひご協力ください. 🙇

ご登録をお願いします！

https://iis-lab.org/algorithms-entry

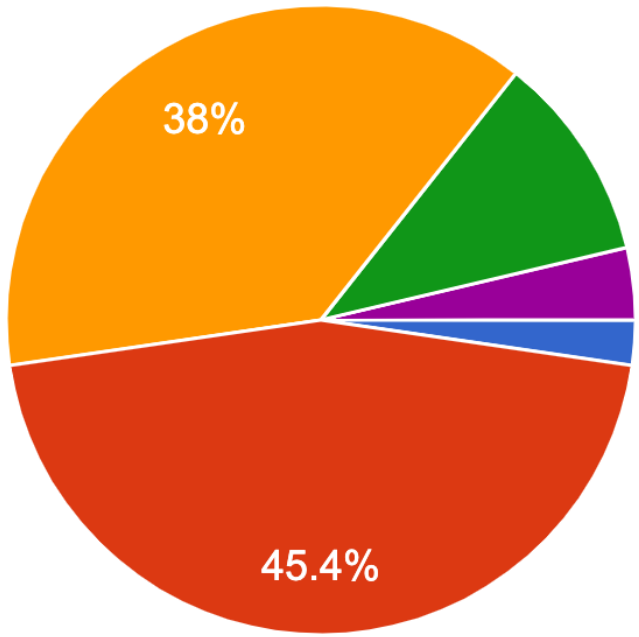
まだの方は，今すぐお願いします！

講義のslackやコードチャレンジで使うシステムへの招待を行う他，皆さんのバックグラウンドを把握するため，ご協力をお願いします．

単位の取得が必要な人は，別途UTASでの登録を忘れないようにお願いします．

Pythonでのプログラミングにどのくらい自信がありますか？

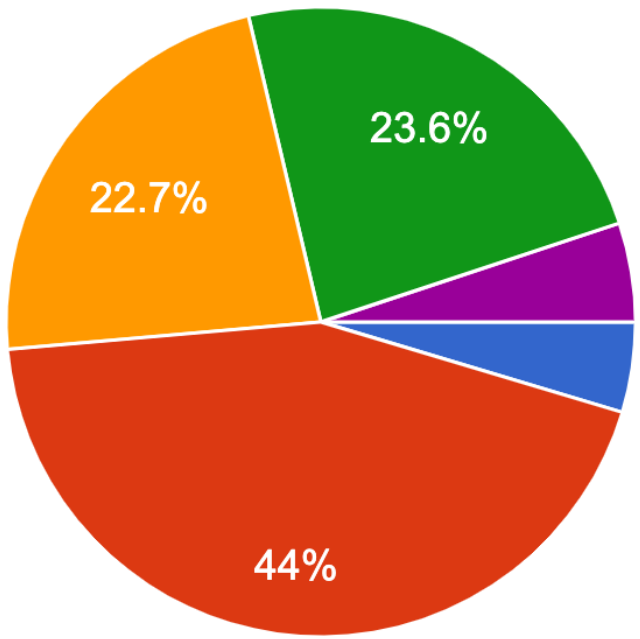
216 responses



- まったくない，Pythonは今回が初めて．
- 少しだけある．授業でちょっとかじった程度．
- ある程度ある．授業＋αで使ったことがある．
- 結構ある．バイトなどでも使っている．
- かなりある．ある程度のことは自力で解決し，実装できる．

競技プログラミング等に参加したことはありますか？

216 responses



- 参加したこともないし，興味もない.
- 興味はあるが，参加したことはない.
- 参加したことはあるけど，続かなかった.
- 参加したことがあり，今でも時々参加している.
- 常連．もしくは，競技プログラミングに取り憑かれている．:)

登録フォームにあったコメント

「所属学科と授業開始がずれているため，期末試験の日程などにおいて障害が生じる可能性がある」

→スケジュールが分かり次第教えてください.

「PEAK生なんですが、レポートを書く場合は英語を使っても大丈夫でしょうか？」

→Yes, of course!

登録フォームにあったコメント

「現在は聴講予定ですが、課題等の負担が想定より小さい場合は履修訂正期間内に履修登録するかもしれません。」

→負担が大きくてもぜひ！

「午後の実験が無期限で延期となっており、再開の目処が立つまでは聴講させていただきたいと思っております。」

→その後もどうぞお時間のあるときに視聴してください！

登録フォームにあったコメント

「非常にアルゴリズムに興味があり、自分で書けるようになりたいのですが、文系で、しかも、まだ２年生で、ついていけるかどうか正直不安です。コードとかで、わからないこととかあったら、質問は、授業時間中はzoomで、授業時間外は、slack,それとも、矢谷さんのめーるに送ればいいのでしょうか？？」

→授業時間外の場合は，algorithms@iis-lab.org，もしくは私とTAさんにslackしてください．

登録フォームにあったコメント

「くだらないことですが、プログラミング能力が学科内最弱レベルの自信があり、どんなに簡単なプログラムでも読んだり書いたりするのにかなりの時間がかかります。しょうもないミスで無限に時間が溶けることが多いし、人が難なくクリアしている小さなことにもよくつまずいています。」

→コーディングは慣れで結構何とでもなります.
いっぱい書くこと，いっぱい書いて自分の癖を知る事が重要だと思います．あとはいっぱいテストすることです.

登録フォームにあったコメント

「同じ時間帯に他の授業を履修します。この授業も興味があるので授業後アップロードされた録画を見ながら課題を解きたいなぁと考えています。単位はいりません。」

→どうぞー．お気に召していただけたらお友達にも紹介してください！

「完全他学部なのですが、がんばります。」

→はい，その心意気が重要です！

では，早速！

アルゴリズムの最低条件

有限実行時間で必ず止まる（停止性）.

止まった時，正しい結果が得られる.

アルゴリズムの最低条件

有限実行時間で必ず止まる（停止性）.

止まった時，正しい結果が得られる.

まあ，そりゃそうですね.

気にしたいのは「いつ止まる」のか.

計算量

与えられたアルゴリズムがどの程度の時間的・メモリ的コストで実行できるかの目安.

具体的な時間やメモリ量を表すものではなく，必要なコストを半定量的に表す指標.

大雑把な見積もり，という感じ.

計算量

与えられたアルゴリズムがどの程度の時間的・メモリ的コストで実行できるかの目安.

具体的な時間やメモリ量を表すものではなく，必要なコストを半定量的に表す指標.

$O(n)$ とか $O(n^2)$ という形で表す.（「オーダーn」というふうに読む.）ビックオー記法と呼ばれる.

時間計算量の例（線形探索）

ある配列の中から，別に与えられた値に一致する要素を探し出す．

配列の先頭から順にチェックして，一致する要素があればその時のindexを返す．ない場合は-1を返す．

例）
入力：[8, 3, 4, 1, 6, 9, 2]で6の場所を探す
出力：4

時間計算量の例（線形探索）

```
def search(sequence, key):
    i = 0
    while i < len(sequence):
        if sequence[i] == key:
            return i
        i += 1
    return -1
```

時間計算量の例（線形探索）

```
def search(sequence, key):
    i = 0
    while i < len(sequence):
        if sequence[i] == key:
            return i
        i += 1
    return -1
```

平均的な実行回数

1回
n/2回
n/2回
1回
n/2回
(1回)

時間計算量の例（線形探索）

```
def search(sequence, key):
    i =0
    while i < len(sequence):
        if sequence[i] == key:
            return i
        i += 1
    return -1
```

$O(n)$

一番支配的な項のみで計算量を表す．

計算量

入力が$O(n)$規模である想定（n個の要素の配列とか）.

平均的な場合と最悪の場合で計算量が変わることもある.

両方の場合で分けてたり，最悪の場合だけ考えたり，
とその時々で違う.

表されている計算量がどんなケースを考えているもの
なのか，正しく理解してから前に進んでください.

オーダーの感覚

$O(1)$
$O(\log n)$
$O(n)$
$O(n \log n)$
$O(nm)$
$O(n^2)$
$O(2^n)$
$O(n!)$

(データの大きさによります)

オーダーの感覚

$O(1)$
$O(\log n)$
$O(n)$
$O(n \log n)$
$O(nm)$
$O(n^2)$
$O(2^n)$
$O(n!)$

(データの大きさによります)

オーダーの感覚

$O(1)$ 😍
$O(\log n)$ 😄
$O(n)$
$O(n \log n)$
$O(nm)$
$O(n^2)$
$O(2^n)$
$O(n!)$

(データの大きさによります)

オーダーの感覚

$O(1)$ 😍
$O(\log n)$ 😄
$O(n)$ ☺️
$O(n \log n)$
$O(nm)$
$O(n^2)$
$O(2^n)$
$O(n!)$

(データの大きさによります)

オーダーの感覚

$O(1)$ 😍
$O(\log n)$ 😄
$O(n)$ ☺️
$O(n \log n)$ 😌
$O(nm)$
$O(n^2)$
$O(2^n)$
$O(n!)$

(データの大きさによります)

オーダーの感覚

$O(1)$ 😍
$O(\log n)$ 😄
$O(n)$ ☺️
$O(n \log n)$ 😌
$O(nm)$ 😑
$O(n^2)$
$O(2^n)$
$O(n!)$

(データの大きさによります)

オーダーの感覚

$O(1)$ 😍
$O(\log n)$ 😄
$O(n)$ ☺️
$O(n \log n)$ 😌
$O(nm)$ 😑
$O(n^2)$ 😩
$O(2^n)$
$O(n!)$

(データの大きさによります)

オーダーの感覚

$O(1)$ 😍
$O(\log n)$ 😄
$O(n)$ ☺️
$O(n \log n)$ 😌
$O(nm)$ 😑
$O(n^2)$ 😩
$O(2^n)$ 😱
$O(n!)$

(データの大きさによります)

オーダーの感覚

$O(1)$ 😍
$O(\log n)$ 😄
$O(n)$ ☺️
$O(n \log n)$ 😌
$O(nm)$ 😑
$O(n^2)$ 😩
$O(2^n)$ 😱
$O(n!)$ 😇

(データの大きさによります)

オーダーの感覚

$$O(n^2)$$

$$O(n \log n)$$

$$O(n)$$

オーダーの感覚

$O(n^2)$

$n = 10^6$

$O(n \log n)$

10^8ステップ/秒

$O(n)$

オーダーの感覚

$O(n^2)$ 2.7時間

$n = 10^6$

$O(n \log n)$ 200ミリ秒

10^8ステップ/秒

$O(n)$ 10ミリ秒

空間計算量（領域計算量）

単純には，メモリの消費量.

IoTデバイスのようなメモリが限られる環境や，超大規模なデータを処理する場合などにはよく考える必要あり.

時間計算量とトレードオフになることもある.

アルゴリズムの授業では時間計算量の方が重要視される（ことが多いと思われる）.

そのほかの計算量・計算コスト

通信コストなど.

Computational offloading（モバイルやウェアラブル端末において重い処理をクラウドなどに投げてしまう）などの場合には重要.

時間計算量を体験しよう！

以下のタスクを解くようなアルゴリズムを考えよう.

「ランダムな整数が格納されている長さNの配列の中で,
m個の隣接する要素の和が最大となる部分を1つ求めよ.」

入力：配列（長さN，0からN-1の重複のない整数）とm

出力：m個の隣接する要素の和が最大値と，そのm個の部分列の一番最初のindex.

時間計算量を体験しよう！

例）「1 1 3 4 2」で隣り合う3つの要素の和が最大になるものはどれか？

「1 1 3 4 2」で，9.

考え方（今日のところはナイーブに）

配列の1番目からm番目までを足し合わせ，その値を最大値として記録．indexは0を記録.

次に，配列の2番目からm+1番目までを足し合わせ，この値が今の最大値より大きければ，最大値とindexを更新.

以降，同じ処理をN-m+1番目からN番目の要素の部分和をチェックするまで繰り返す.

部分和が同じになる場合，indexの若い方を優先する.

コードチャンレジ：基本課題#1

1つ前にあるスライドに説明されているアルゴリズムを実装してください.

課題の詳細はこれから説明するtrackの上で公開されています.

実装できたら

自分のローカル環境で，配列の長さやmの値を適当に変化させてみて，計算が完了するまでにどのくらい時間がかかるかを試してみてください．（例えば，配列の長さを10万くらいにするとどうでしょうか？）

このアルゴリズムの計算量がいくらか，考えてみてください．

実装できたら

ランダムに非負整数が並んだ配列は以下のコードで
作れます.

```
import random

def RandomIntSeq(length):
        seq = random.sample(list(range(0, length)), k=length)
        return seq
```

コードチャンレジ：Extra課題#1

今日はExtra課題はありません.

コードチャンレジで使うシステムに慣れておいてください.

実装してみよう！

ここから，コードチャレンジで使うシステムの説明.

システムのwalkthroughなどを残りの時間で行います.