



コンピュータ・サイエンス2

第12回 データ構造とアルゴリズム

人間科学科コミュニケーション専攻
白銀 純子





今回の内容

- データ構造とアルゴリズム





設問1

- 下記のページについて、httpsであるべきか、httpでも良いかを答えなさい。
 1. WebClassのログインページ
 2. 東女の図書館のトップページ
 3. 学長メッセージが掲載されたページ
 4. 情報処理教室のパスワード変更ページ

解答:

- httpsであるべきもの: 1, 4
- httpでも良いもの: 2, 3



設問2

- コンピュータ名からIPアドレスを調べた結果を、2件以上報告しなさい。

解答例1 - <https://webclass.twcu.ac.jp/webclass/login.php>

➤ 「host webclass.twcu.ac.jp」で引く → 202.11.168.73

解答例2 - <http://weather.yahoo.co.jp/weather/jp/3.html?day=1>

➤ 「host weather.yahoo.co.jp」で引く → 182.22.24.124



Question!





前回の質問の回答



データ構造とアルゴリズム



データ構造(p. 116)

- データ構造：データをメインメモリに格納するときの形式
 - コンピュータはデータをメインメモリに記憶させて処理
 - コンピュータが扱いやすい形式で格納する必要
 - データの形式によって、プログラムの効率に大きく影響

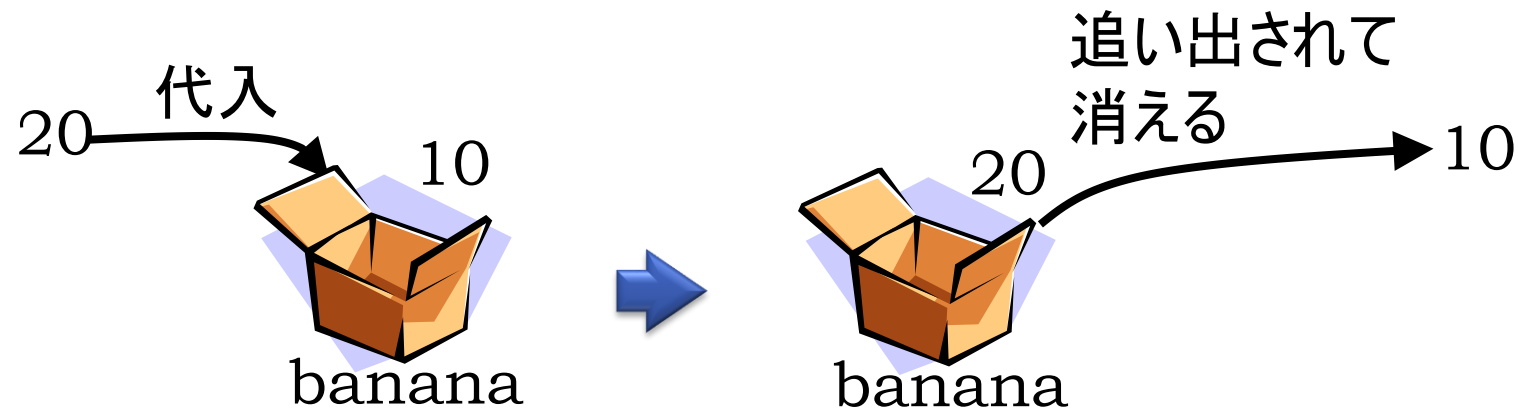
変数(p. 116)

- **変数**: 計算の対象や処理結果などのデータを記憶しておくための場所
 - データを入れておく「箱」と言うことができる
 - 変数には名前をつける
 - 変数にデータを入れることを「**代入**」と呼ぶ
 - 変数に入れられたデータのことを「**値**」と呼ぶ
 - 変数の名前を指定するとデータを取り出すことができる
 - 変数は扱うデータの個数分だけ用意する



変数の特徴[1](p. 116)

- 変数の中のデータを取り出しても、変数の中にデータは残っている
 - 「変数の値を参照する」とは、「箱の中のデータを見る」という意味
- すでにデータが入っている変数に別のデータを入れると、元のデータは消えてしまう
 - 1つの変数に入れておくことができるデータは1つだけ

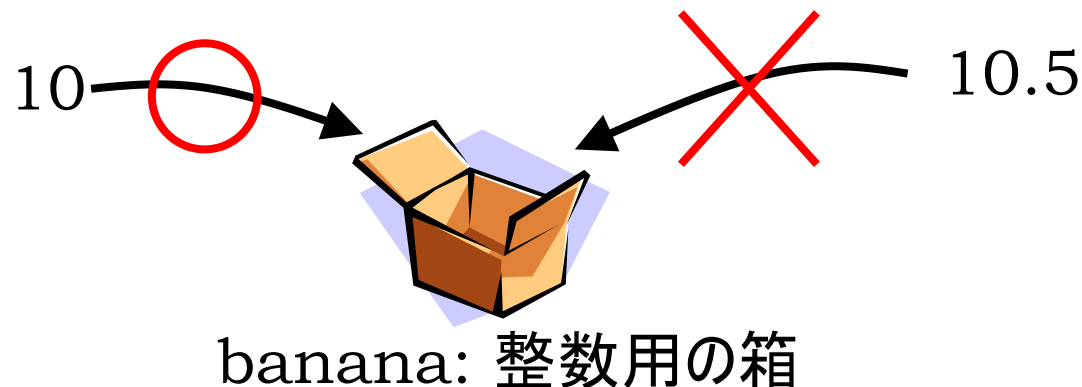


変数の特徴[2](p. 116)

- データの種類によって、違う箱を使う必要がある

- 整数用の箱
 - 実数用の箱
 - 文字列用の箱
 - etc.
- 整数用の箱に実数を入れることはできない
 - 文字列用の箱に整数や実数を入れることはできない

あらかじめ、「この名前の箱は整数用の箱」と、決めてコンピュータに知らせた上で箱を使い始める



「配列」って?[1](p. 116)

- データの種類(整数や小数)が同じで、処理方法も同じ変数をたくさん扱うときに利用する変数
 - たくさんの変数を1度にまとめて利用する方法

例えば...生徒の英語の成績を扱うプログラム(30人分)

出席番号1番の生徒の成績

出席番号2番の生徒の成績

...

出席番号30番の生徒の成績

30個の変数が必要!

➡ english1, english2, english3, ..., english30
のように用意して使うのは大変!

➡ 「配列」を利用



配列って?[2](p. 116)

- 配列を利用するには...
 - 扱うデータのグループに名前を付ける(配列名)
 - Ex. 生徒の英語の成績: english
 - 配列名に番号(添え字)を「[]」でつけて、個々のデータを扱う
 - Ex. 生徒の英語の成績
 - 出席番号1番の生徒の成績: english[0]
 - 出席番号2番の生徒の成績: english[1]
 -
 - 出席番号30番の生徒の成績: english[29]



アルゴリズム





アルゴリズムとは[1](p. 118)

- **アルゴリズム**: ある問題を解決するときに必要な処理手順
- プログラムでの処理の方法を記述したもの
 - 何をどのように行うかを記述
 - コンピュータには手順を1つ1つ詳細に指示する必要
 - 人間には一言ですむような処理でも、コンピュータがその処理をこなすには、たくさんの手順が必要



アルゴリズムの書き方(p. 119)

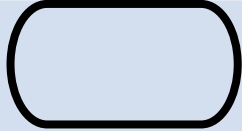
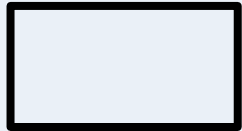
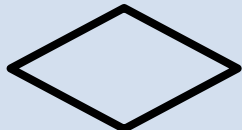
- 文章で書く
 - 箇条書きで書くことも多い
- プログラミング言語に自然言語を混ぜて書く(疑似言語)
 - 自然言語: 普段人間が話したり書いたりしている言葉
- 図で描く
 - フローチャート(流れ図)を使うことが多い



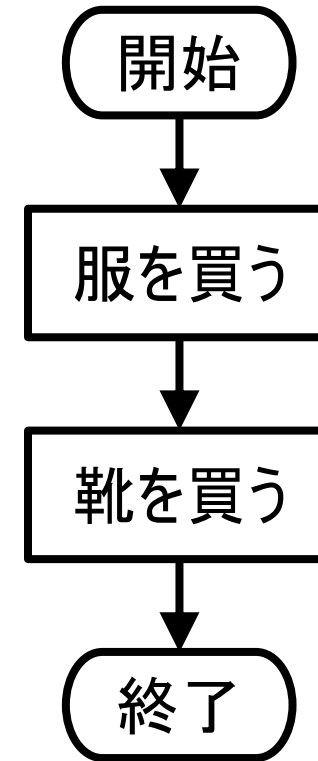
フローチャート[1](p. 119)

- 記号や矢印などを使って処理の流れを描いた図
- 順次処理、条件分岐、反復処理が基本
 - **順次処理**: プログラム中に書いてある命令を、上から順に1つずつ処理すること
 - **条件分岐**: ある条件を満たしたときとそうでないときで、処理内容が変わること
 - **反復処理**: ある条件が満たされている限り、処理を繰り返すこと

フローチャート[2](p. 119)

記号	意味
	開始と終了
	処理
	条件判断
	処理の流れ

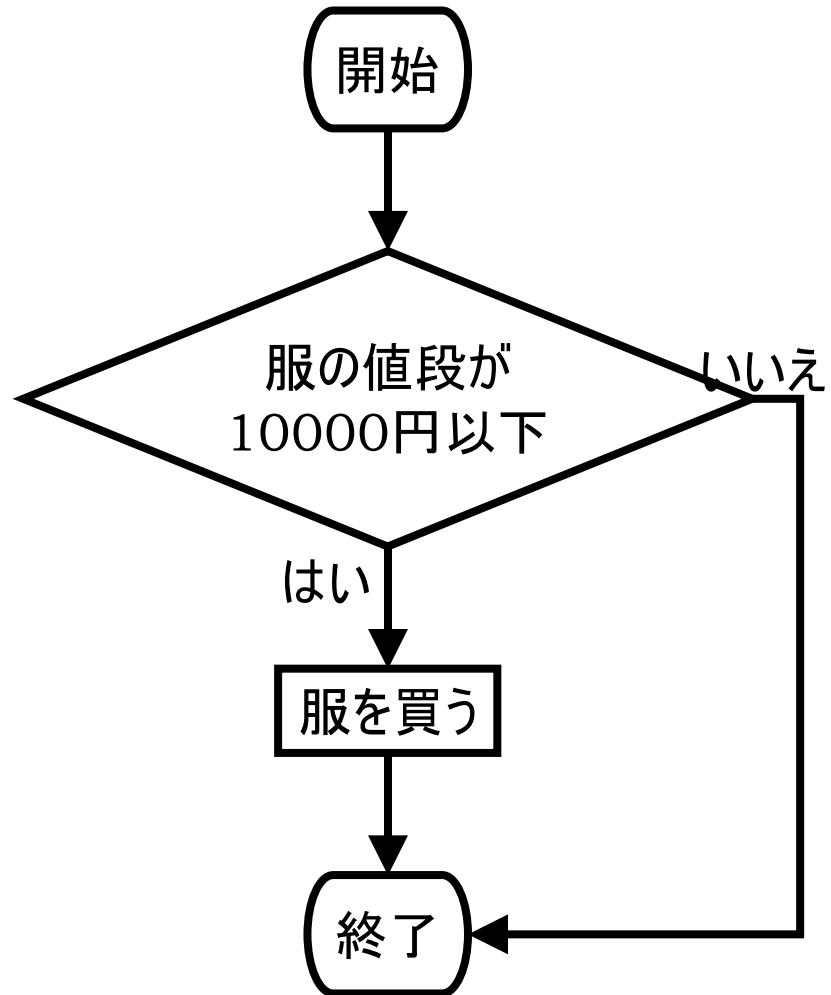
順次処理の例



フローチャート[3](p. 119)

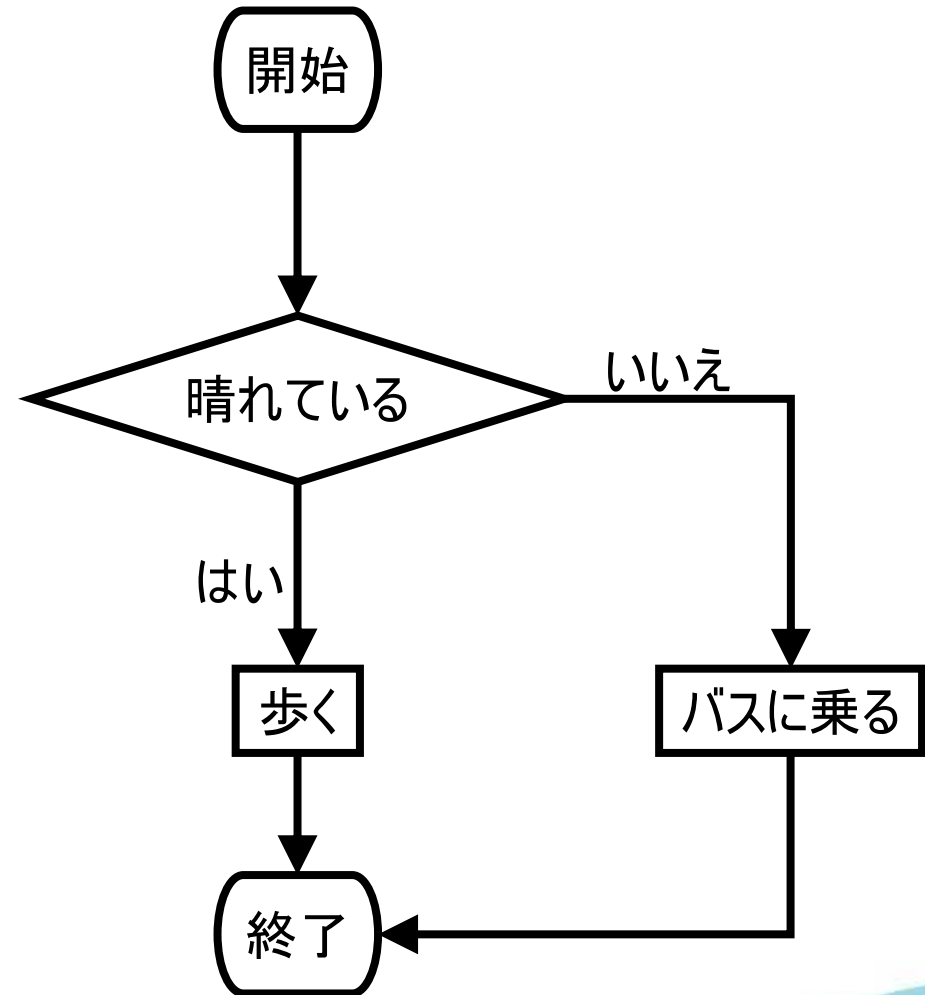
条件分岐の例1

(条件判断が「いいえ」の場合何もしない)



条件分岐の例2

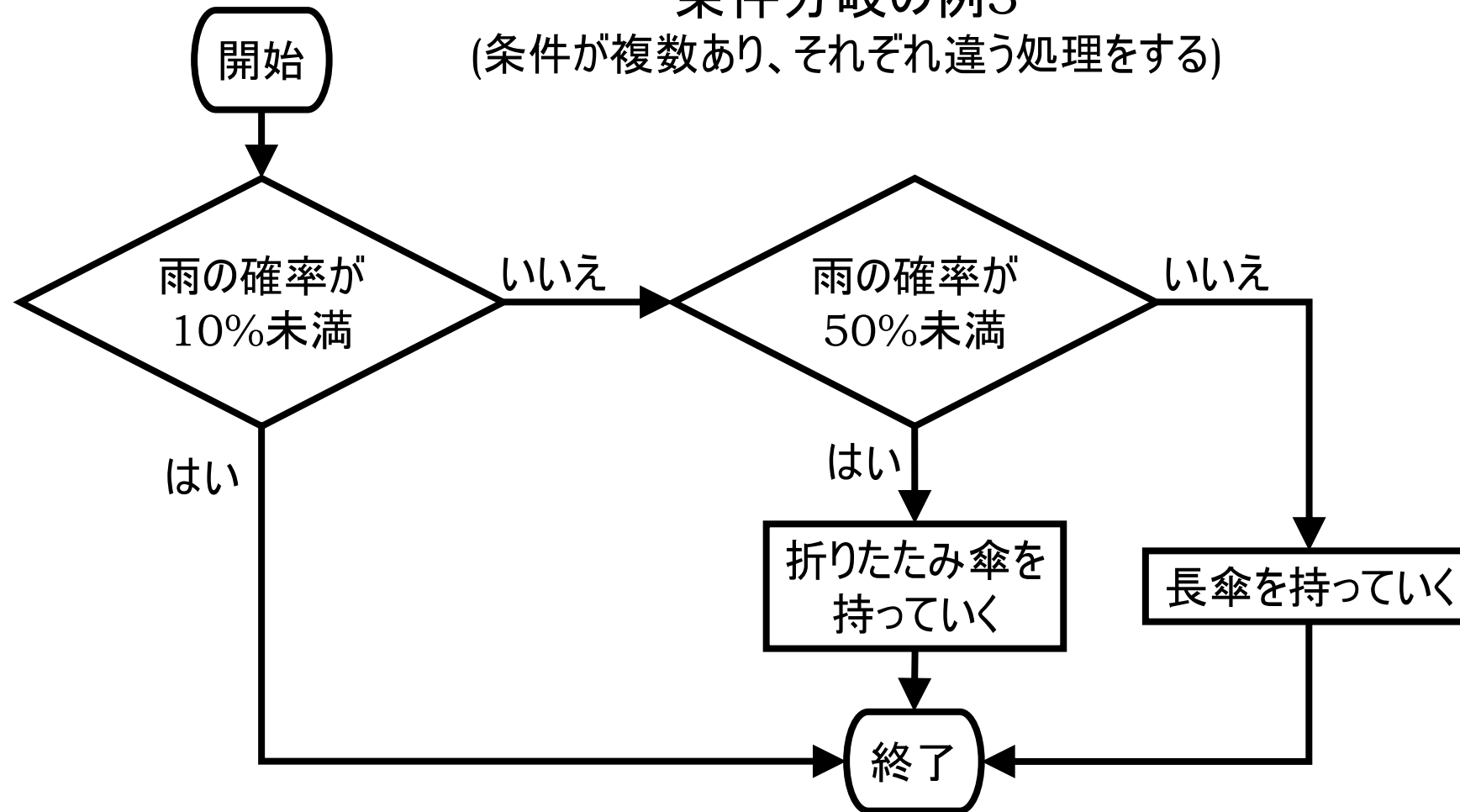
(条件判断が「いいえ」の場合別のことをする)



フローチャート[4](p. 119)

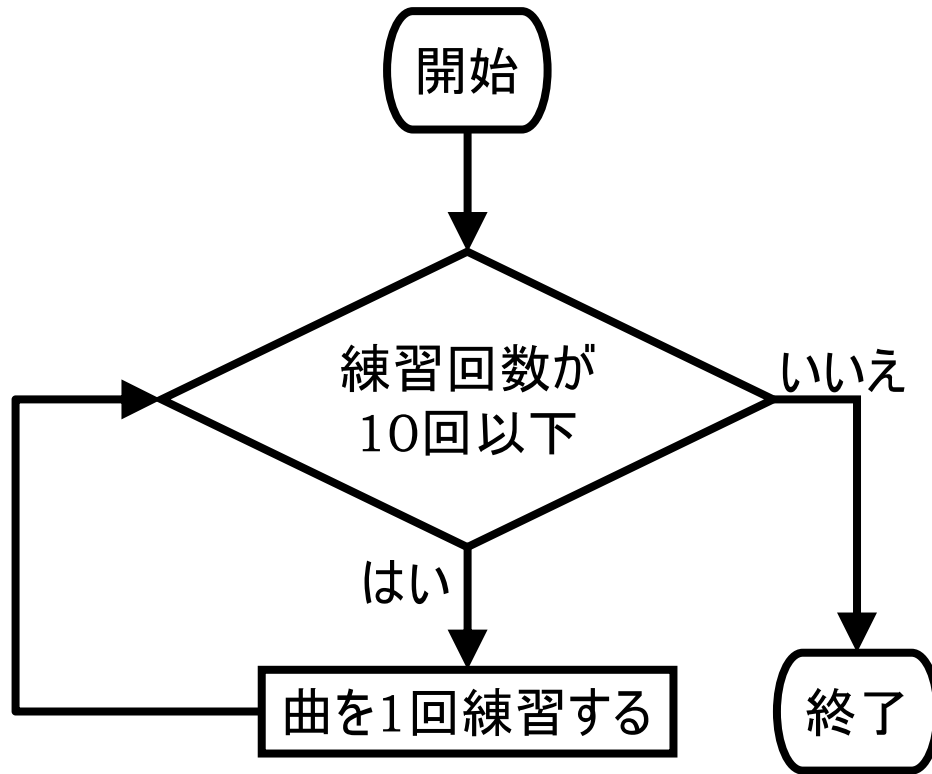
条件分岐の例3

(条件が複数あり、それぞれ違う処理をする)

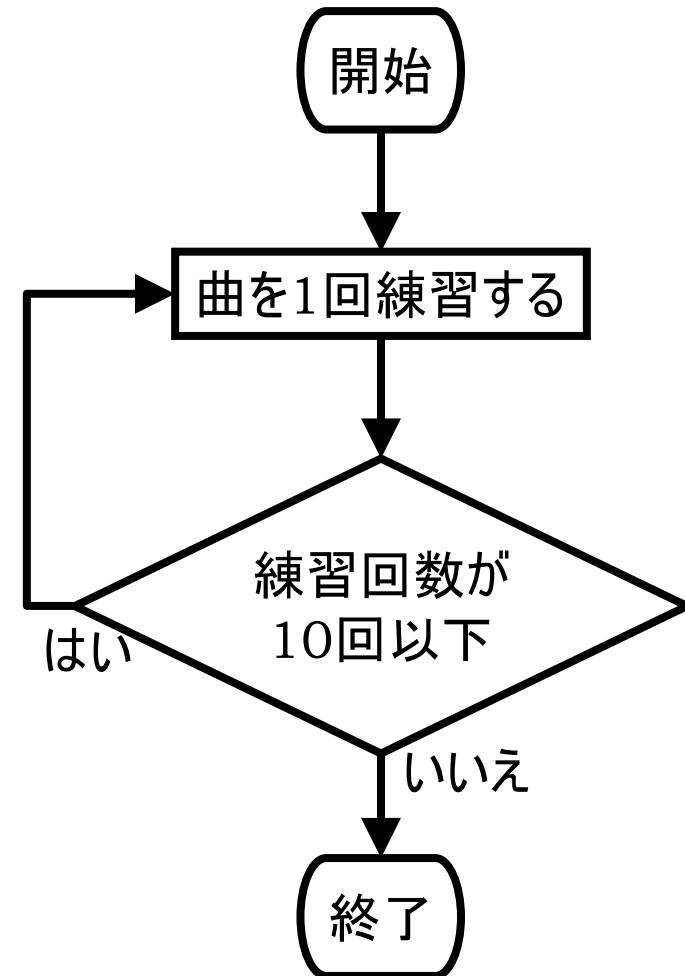


フローチャート[5](p. 119)

反復処理の例
(最初に条件を判断)



反復処理の例
(最初に処理をしてから条件を判断)





Question!



探索アルゴリズム



探索アルゴリズム[1](p. 121)

- **探索**: たくさんのデータから目的のデータを見つけること

Ex. 高校の生徒の得点を管理するプログラム

- 出席番号5番の生徒の英語の点数を知りたい
→ 高校の生徒の配列から、出席番号が「5」というものを探す
- 「東京子」という生徒の国語の点数を知りたい
→ 高校の生徒の配列から、名前が「東京子」というものを探す



探索アルゴリズム[2](p. 121)

- 様々な探索アルゴリズム
 - 逐次探索
 - 2分探索
 - 自己組織化探索
 - 2次元探索
 - 補間探索
 - ハッシュ法

逐次探索[線形探索](p. 121)

- データを前から順番に比較して探していく方法

データの中から「98」を見つけたい!

添え字	0	1	2	3	4	5	6	7
データ	21	13	98	31	44	87	72	50

1. 添え字「0」のデータをチェック: 違う

添え字	0	1	2	3	4	5	6	7
データ	21	13	98	31	44	87	72	50

2. 添え字「1」のデータをチェック: 違う

添え字	0	1	2	3	4	5	6	7
データ	21	13	98	31	44	87	72	50

3. 添え字「2」のデータをチェック: 見つかった!

添え字	0	1	2	3	4	5	6	7
データ	21	13	98	31	44	87	72	50

二分探索[1](p. 121)

- 小さい順に並べられたデータの中から、中央のデータと目的のデータを比較することで、探す方法
 1. 中央のデータと探したいデータを比較する
 2. 1. の結果、中央のデータが大きければ、探したいデータは右半分、そうでなければ左半分を、探索対象とする
- 中央のデータと探したいデータが同じになるまで繰り返す

二分探索[2](p. 121)

データの中から「98」を見つけたい!

添え字	0	1	2	3	4	5	6	7
データ	13	21	31	44	50	72	87	98

1. 中央のデータ(44)と目的のデータ(98)を比べる
→ 目的のデータ(98)の方が大きいので、右半分を探索対象にする

添え字	0	1	2	3	4	5	6	7
データ	13	21	31	44	50	72	87	98

2. 中央のデータ(72)と目的のデータ(98)を比べる
→ 目的のデータ(98)の方が大きいので、右半分を探索対象にする

添え字	4	5	6	7
データ	50	72	87	98

二分探索[3](p. 121)

3. 中央のデータ(87)と目的のデータ(98)を比べる
→目的のデータ(98)の方が大きいので、右半分を探索対象にする

添え字	6	7
データ	87	98

4. 中央のデータ(98)と目的のデータ(98)を比べる
→同じなので見つかった!

添え字	7
データ	98



整列(ソート)アルゴリズム(p.123)





整列[ソート](p. 123)

- ソート: 複数の数を小さい順or大きい順に並べること
 - 選択ソート
 - バブルソート
 - 挿入ソート
 - クイックソート
 - etc.



選択ソート[1](p. 123)

- 変数 t : 並べ替えをする数の中で最も小さい数を入れておく変数
- 変数 i : 並べ替えをする数の中で、 t が最初から何番目の位置にあるかを表す変数
- 変数 j : 何回繰り返したかを数えるための変数
- 並べ替えをする数は n 個とする



選択ソート[2](p. 123)

1. t に並べ替えをする数の一番最初の数を代入する
2. i に1を代入する
 - 1: 並べ替えをする数の一番最初の数の位置
3. j に2を代入し、1ずつ増やしながら n になるまで以下を繰り返す
 - t が j 番目の数より大きいならば、 j 番目の数を t に代入し、 i に j の値を代入する
 - この処理を1回することにより j の値を1増やす
 - j の値は、現在調べている数の位置になる
4. 並べ替えをする数の一番最後の数と t を入れ替える

選択ソート[2](p. 123)

4 8 2 5

ステップ1:

- tに4を代入する
- iに1を代入する

ステップ2:

- jに2を代入する
- tの値(4)と8を比べる
- tの値の方が小さいのでそのまま

ステップ3:

- jの値を1増やす(3になる)
- tの値(4)と2を比べる
- tの値の方が大きいのでtに2を代入し、iにjの値(3)を代入する

ステップ4:

- jの値を1増やす
- tの値(2)と5を比べる
- tの値の方が小さいのでそのまま

ステップ5:

- tの値(2)を最後に置く
- これまで最後だった数(5)をi番目に入れる



4 8 5 2

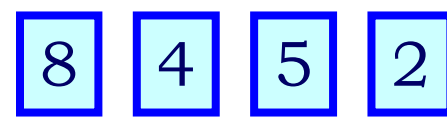
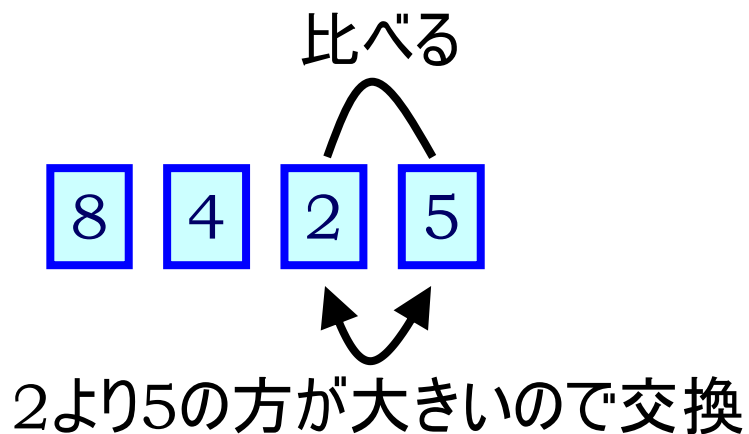
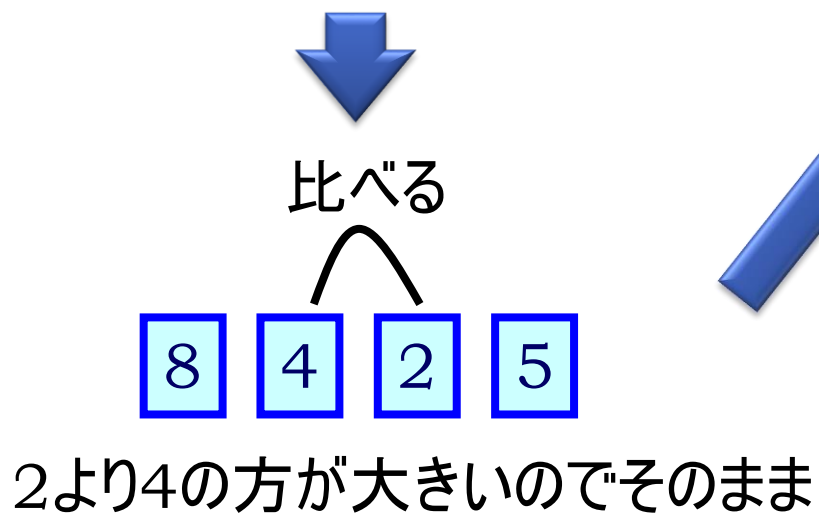
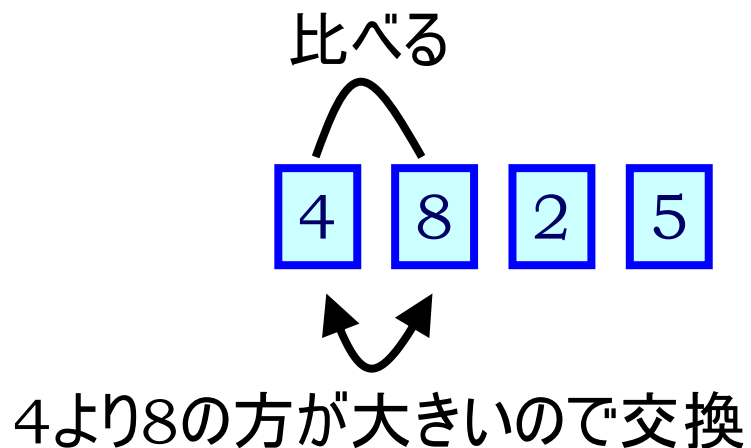
- 最も小さな数が一番後ろに来る
- 次は、一番後ろの1つ前まで(4, 8, 5)で同じようにする



バブルソート[1](p. 124)

- 前から2つずつ、数の大きさを比較して、小さい数を後ろに送っていく
 - 最後まで調べると、最も小さな数が一番後ろにある
 - この作業を、並べ替える数の個数だけ繰り返すと、数が大きい順に並ぶ

バブルソート[2](p. 124)



- 最も小さな数が一番後ろに来る
- 次は、一番後ろの1つ前まで(8, 4, 5)で同じようにする



その他

- 挿入ソートのアルゴリズム(p. 125)は、教科書をよく読んでおくこと



やってみよう!

- 配ったカードを、順番をバラバラにして...
 1. バブルソートで昇順に並べ替え(練習)
 2. バブルソートで昇順に並べ替え、かかった時間を計測(本番)
 3. 自分のやりやすい方法で昇順に並べ替え、かかった時間を計測
 4. 2. と3. の時間を比較(どちらが時間がかかっているか??)



Question!





アルゴリズムのよしあし





良いアルゴリズム(p. 126)

- そのアルゴリズムを使ったプログラムをコンピュータで実行するときの処理時間や記憶領域の使用量
- アルゴリズムのわかりやすさ・作りやすさ・修正の容易さ



アルゴリズムの計算時間(p. 126)

- アルゴリズムをコンピュータで実行したときの処理時間

- CPUそのものの速さ
- CPUとメインメモリとの間のアクセスの速さ
- etc.

これらを除いても、同じ結果を出す複数のアルゴリズムで計算時間に違いが出る

➡ アルゴリズムの計算量

※同じコンピュータで同じアルゴリズムの処理をしても、そのときどきで処理に必要な時間が異なる



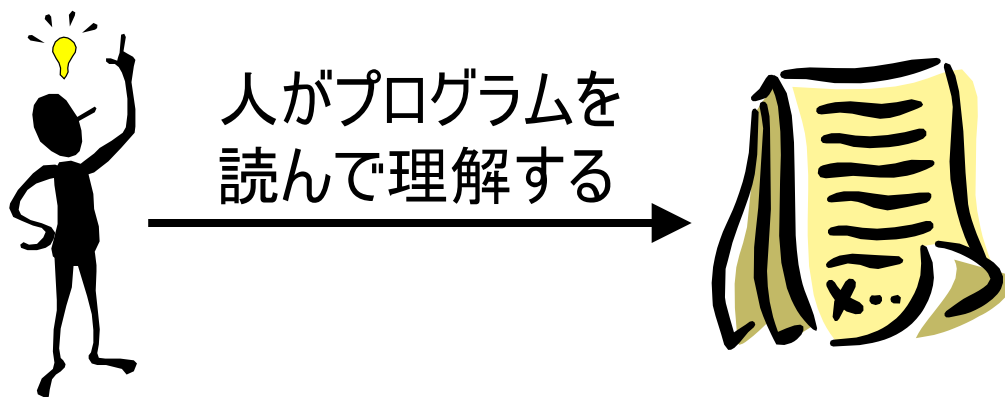
アルゴリズムの計算量(p. 126)

- CPUの速さなど、アルゴリズムには関係ない要因を除いた、アルゴリズムそのものの計算時間
 - アルゴリズムそのものの計算(処理)の速さ
- アルゴリズムでの計算の複雑さ

アルゴリズムのわかりやすさ(p. 126)

- 一旦完成したプログラム: 機能の追加などのためにプログラムの修正が必要なことも多い

アルゴリズムの修正



プログラムを読んで理解する = 書かれてあるアルゴリズムの理解が必要

- アルゴリズムが難解 ⇨ 修正が難しい
- アルゴリズムが簡単 ⇨ 修正が容易



アルゴリズムを使う状況(p. 126)

- 多くの場合、計算量の少ない(処理時間の速い)アルゴリズムとわかりやすいアルゴリズムは対立関係

- 計算量が少なければ、わかりにくいアルゴリズム
- わかりやすければ、計算量が多いアルゴリズム

速さをとるか、わかりやすさをとるかは、状況に応じて判断

- めったに使わないプログラムや頻繁に修正するプログラム: わかりやすいアルゴリズム
- よく使うプログラムや計算時間に制約があるプログラム: 速いアルゴリズム

ソートアルゴリズムの比較[1](p. 126)

- 結果が出るまでの基本処理の回数(アルゴリズムの計算量)

- バブルソート: $N(N-1)/2$
 - 併合ソート: $N/2 + (N-1)\log_2 N$
- ※ $\log_2 N$: N を 2^k としたときの「 k 」
- } N : 並べ替える数の個数

N	$N(N-1)/2$ (バブルソート)	$N/2 + (N-1)\log_2 N$ (併合ソート)
8	28	25
32	496	171
64	2016	410
128	8128	953

➡ N が大きければ大きいほど、併合ソートの方が速い

ソートアルゴリズムの比較[2](p. 126)

- **計算量**: 入力(N: 並べ替えの場合は数の個数)に対して行われる基本処理の回数

- Nが十分に大きなとき: 計算式の中の最も大きな項だけに着目して、大まかに計算

= 各項の比例定数や次数の低い項は無視

- バブルソート: $N(N-1)/2 = N^2/2 - N/2$
→ N^2 のみに注目
- 併合ソート: $N/2 + (N-1)\log_2 N = N/2 + N\log_2 N - \log_2 N$
→ $N\log_2 N$ のみに注目

アルゴリズムの計算量は、正確な計算量ではなく、Nが大きくなればどの程度の割合で計算量が増えるかを大まかに知ることが重要なため

➡ 注目する項を取り出して、 $O(\dots)$ と表記
➤ $O(N^2)$ や $O(N\log_2 N)$ など



ソートアルゴリズムの比較[3](p. 126)

- 計算時間の速いアルゴリズム: N や $\log N$ などのみで計算量が計算できるアルゴリズム
- 計算時間の遅いアルゴリズム: N^2 , N^3 , ..., N^k や $N!$ (1から N までを掛けあわせた数), 2^N など、多くのかけ算を計算に必要とするアルゴリズム
 - N^2 , N^3 などの計算を必要とするアルゴリズム: 多項式時間アルゴリズム
 - $N!$ や 2^N などの計算を必要とするアルゴリズム: 指数時間アルゴリズム



扱いにくい問題



扱いにくい問題(p. 127)

- コンピュータでの処理が難しい問題も存在

データの個数に対する処理時間
(1回の処理に0.00000001秒かかるコンピュータ)

n	logn	n	nlogn	n ²	n ³	2 ⁿ	n!
10	0.00000003	0.0000001	0.0000003	0.000001	0.0001	0.0001024	0.36
20	0.00000004	0.0000002	0.0000009	0.00004	0.0008	0.1048576	7700年
30	0.00000005	0.0000003	0.000015	0.00009	0.0027	107	8 × 10 ¹⁸ 年
50	0.00000006	0.0000005	0.000028	0.00025	0.012	3.4年	
100	0.00000007	0.000001	0.000066	0.001	0.1	4 × 10 ¹⁵ 年	
10000	0.00000013	0.001	0.0133	10	1.2日		
1000000	0.00000023	1.0	23	116日	3200000年		

※「日」や「年」の書いていない数の単位は「秒」

扱いにくい問題[ナップザック](p. 127)

- 重さと値段のわかっているN個の荷物をナップザックに詰め込むとき、合計金額を最大いくらにできるか
 - 荷物の重さの合計はWを超えてはならない

解答例: 荷物の全ての組み合わせを作って
重さの合計と金額の合計を計算

荷物がN個の場合、荷物の組み合わせは 2^N 通り
= 重さの合計と金額の合計を 2^N 回計算する必要

例えば...

荷物が60個、計算の基本処理1回分が1000万分の1秒の場合:

$1 / 1000万 \times 2^{60}$ 秒 $\div$ 3000年

➡ アルゴリズムを作ることはできるが、
計算時間が非現実的!

扱いにくい問題[セールス](p. 127)

- セールスパーソンが、 A_0 町(駅)からN個の町(駅)を回って A_0 町(駅)に帰るまでに最小のコスト(交通費)の経路を求める

解答例: A_0 から始まって、N個の町を回って帰ってくる事ができる全ての経路を考え、それぞれの経路のコストの合計の中で最小のものを求める

町がN個の場合、経路の組み合わせは $(N-1)!$ 通り
= 調べなければならない経路が $(N-1)!$ 通り

例えば...

町が30個、計算の基本処理1回分が100万分の1秒の場合

$(30 - 1)! \times 1 / 100万 = 8.8 \times 10^{30} \text{回} \times 1 / 100万 \div 3800 \text{年}$

➡ アルゴリズムを作ることはできるが、
計算時間が非現実的!



扱いにくい問題への対応(p. 127)

- 入力が特殊な条件を満たす場合は、扱いやすくなることもある
- 最適解でなく、近似解で良ければ、扱いやすくなる
 - 最適解: 最も良い答え
 - 近似解: 最も良いわけではないかもしれないが、他の多くの答えよりは良い答え



次回(12/18)

- 実習のため、24102教室に集合

