コンピュータ・サイエンス2

第11回
データ構造とアルゴリズム(続き)

人間科学科コミュニケーション専攻
白銀 純子

第11回の内容

- データ構造とアルゴリズム(続き)

前回の復習

情報の隔離[1](p. 110)

- 重要な情報を守るために…
  - 守るべきものを隔離する
  - 必要最低限の人や機器だけが利用可能にする
    - 安全性と利便性は常に対立関係
      - 安全性を上げれば利便性が下がり、利便性を上げれば安全性が下がり…という関係

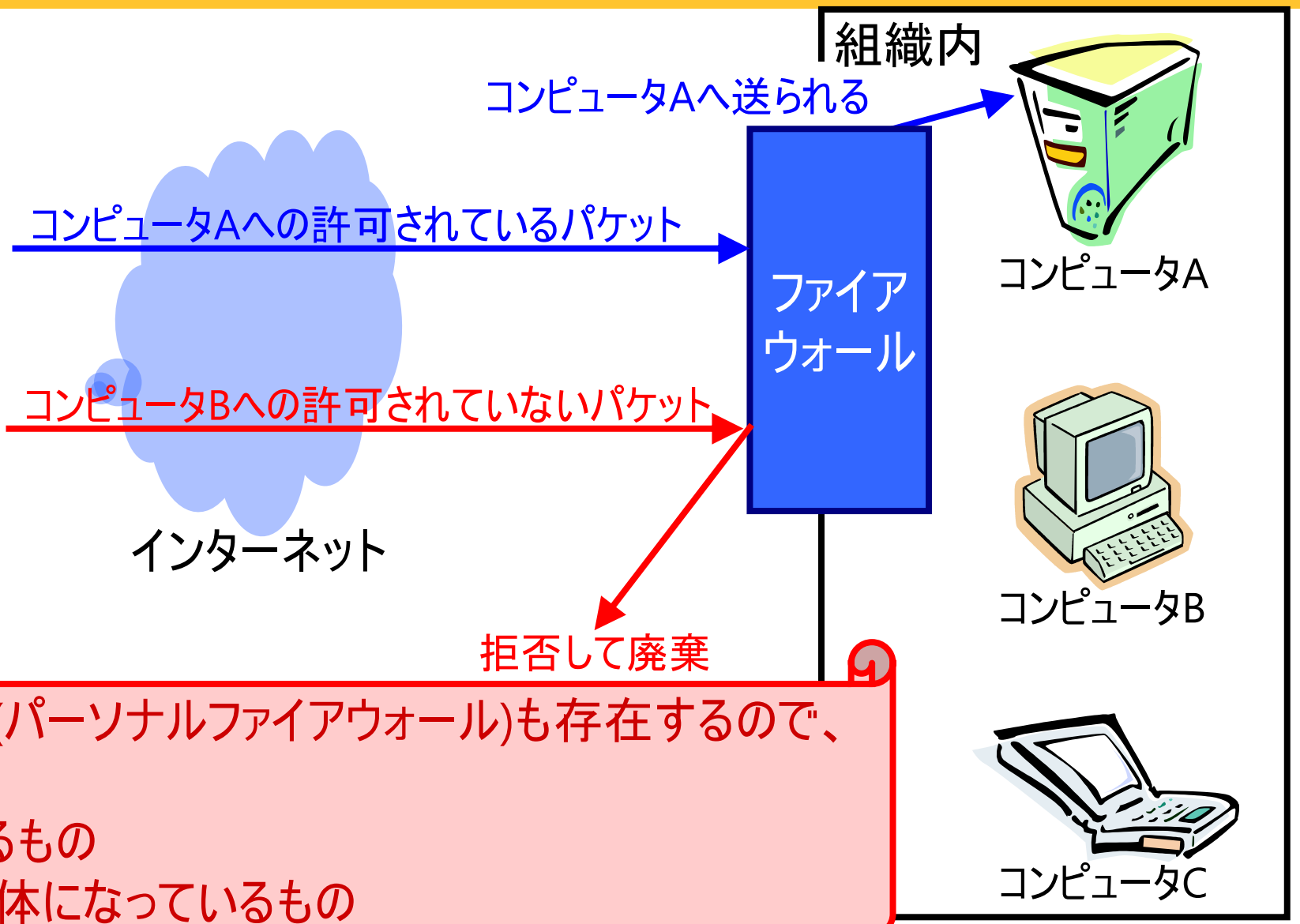
安全性と利便性のバランスを考慮してセキュリティポリシーで情報保護の方針を決定

情報の隔離[2](p. 110)

- ファイアウォールの設置
  - ファイアウォール: 組織内と外部との間に設置して組織内に不正にアクセスされないように監視するコンピュータ
  - 外部からのパケットの監視(アクセス制御)
    - 許可されていないIPアドレス(インターネット上の住所)からパケットが送信されていないか?
    - 許可されていないポート(データの出入り口)にパケットが送信されてきていないか?

許可されていないアクセスを遮断(フィルタリングと呼ぶ)

情報の隔離[3](p. 110)



個人用のファイアウォール(パーソナルファイアウォール)も存在するので、利用して情報を守ろう!

- OSに付属しているもの
- ウィルスソフトと一体になっているもの

情報の隠蔽[1](p. 111)

- インターネット上での通信(メール, Web, etc.)
  - データがそのままの形で送受信される
    = パスワードなどの個人情報がそのままインターネット上に流される

    = 途中で盗聴されてデータが盗まれる可能性もある

    ➡ インターネット上での盗聴は、仕組み上防ぐことは難しい

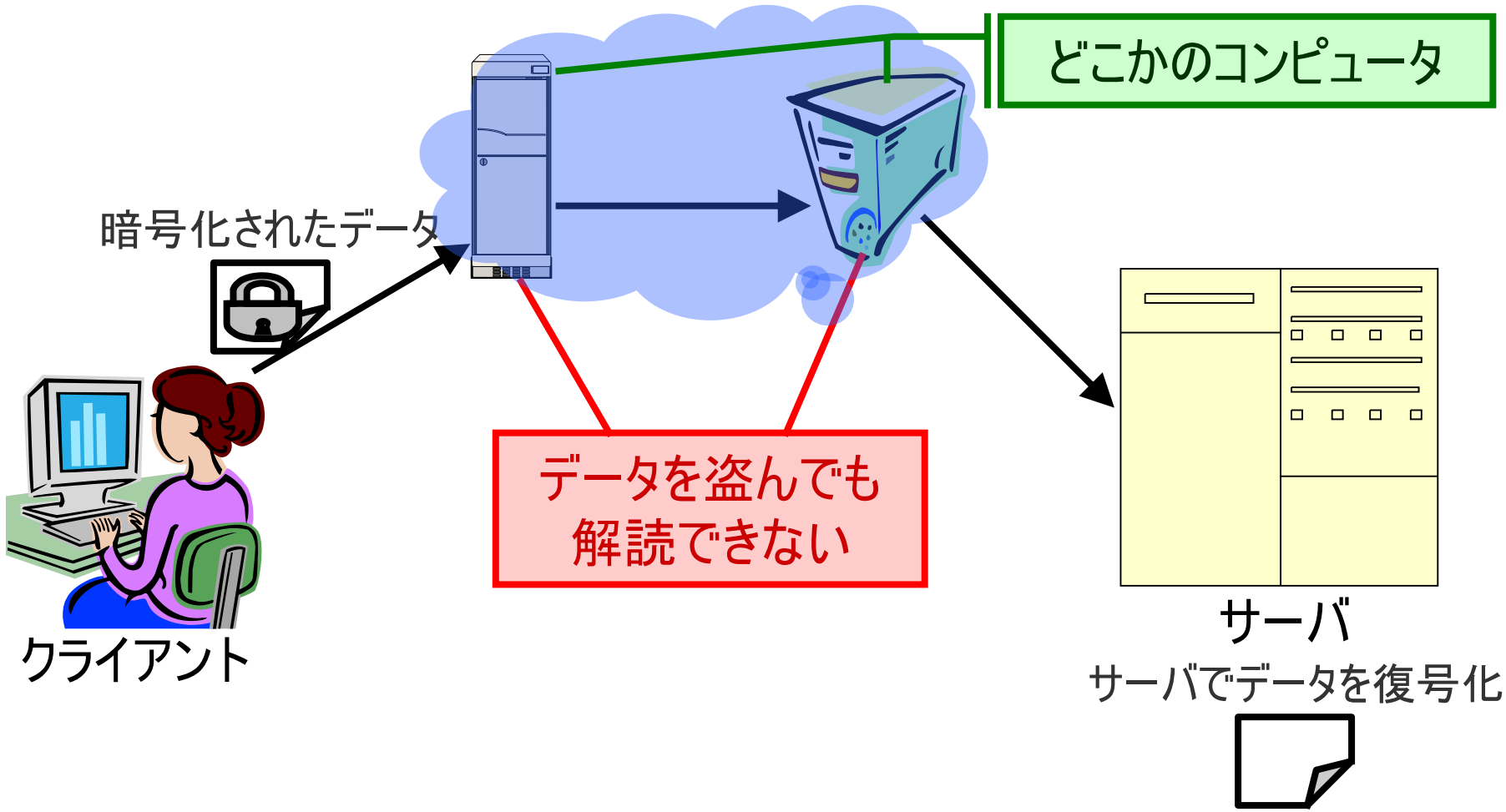
    - データを暗号化し、盗まれても中身を理解不能にする
    - ➡ 正当な受け取り主は、暗号を解読して本来のデータを見ることができるようにする

情報の隠蔽[2](p. 111)

- **暗号化**: データを別の形に加工すること
  - データが元の形と違っているので、データを見ても内容がわからない
  - Ex. This is a pen. → Uijt jt b qfo.
    - 暗号化の方法: アルファベットを1文字後ろにずらす
- **復号化**: 暗号化されたデータをもとの形に戻すこと
  - 復号化する方法を知らなければ、もとのデータの内容がわからない
  - Ex. Uijt jt b qfo. → This is a pen.
    - 復号化の方法: アルファベットを1文字前にずらす

情報の隠蔽[3](p. 111)

- **暗号化通信**: 利用者の使っているコンピュータで暗号化をして送り、サーバ側で復号化する通信方法



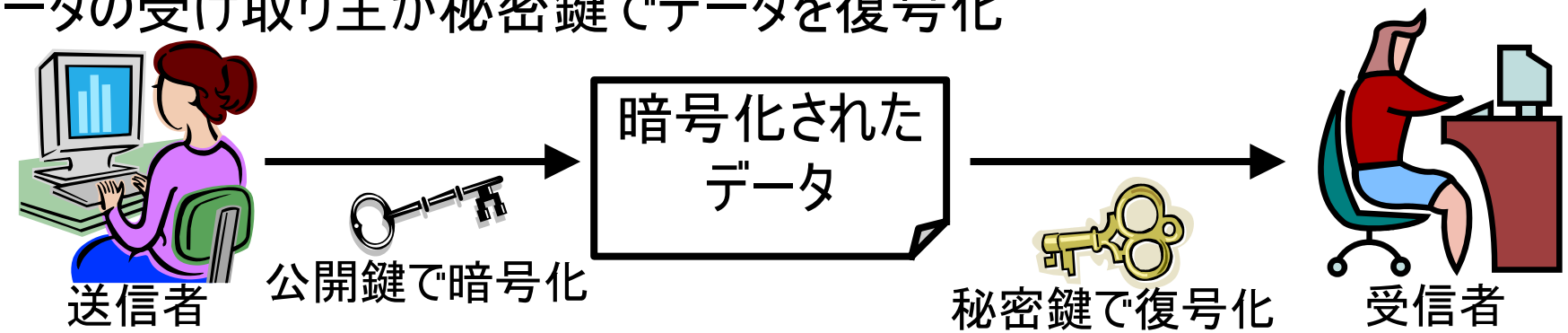
情報の隠蔽[4](p. 111)

- 共通鍵暗号方式(秘密鍵暗号方式とも呼ぶ)
  - データを暗号化するために「暗号鍵」を使う
    - 暗号鍵: データを暗号化するために使うキーワード(キーワードが長ければ長いほど、暗号が解読されにくい)
  - データを暗号化するときと復号化するときで、同じ暗号鍵を使う
  - 欠点1: データを送る側と受け取る側で暗号鍵を受け渡しする方法が難しい
    - 下手な方法では、途中で盗まれてしまう
  - 欠点2: 相手ごとに暗号鍵を用意する必要がある

情報の隠蔽[5](p. 111)

公開鍵暗号方式

- 「公開鍵」と「秘密鍵」という2種類の暗号鍵を使う方法
  - 公開鍵: データを暗号化するための暗号鍵
  - 秘密鍵: データを復号化するための暗号鍵
- データのやりとりの方法
  1. データの受け取り主が公開鍵と秘密鍵を作成
  2. データの受け取り主が公開鍵をデータの送信者に受け渡し
  3. データの送信者がデータを公開鍵で暗号化し、送信
  4. データの受け取り主が秘密鍵でデータを復号化



情報の隠蔽[6](p. 111)

- 公開鍵方式
  - 秘密鍵を知らなければ、データを復号化できない仕組み
  - 公開鍵と秘密鍵は対
  - 秘密鍵は、データを受け取る側しか知らない暗号鍵
    - 他の人に知られてはならない暗号鍵
  - 公開鍵は、他人に知られても良い暗号鍵
  - 利点: 秘密鍵を割り出そうとすると、膨大な時間がかかるので、事実上不可能
  - 欠点: 共通鍵暗号方式に比べて、復号化処理に時間がかかる

情報の隠蔽[7](p. 111)

- WWWでは、公開鍵暗号方式を利用
  - SSL(Secure Socket Layer)と呼ばれている
- Webの場合、URLが「https://」で始まっていれば、SSLでの通信
  - https: HTTP over SSL
  - 「http://」の場合は、普通の暗号化しない通信

Webでの個人情報の入力時には、URLがhttpsで始まっているかどうかを確認しよう!

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

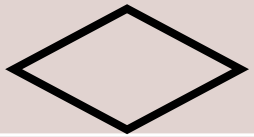
アルゴリズムの書き方(p. 119)

- 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

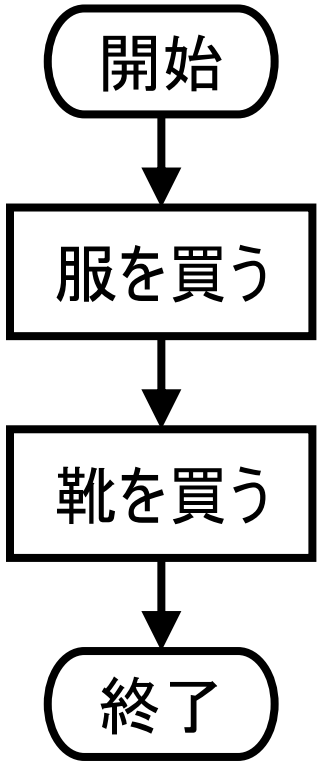
フローチャート[1](p. 119)

- 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)

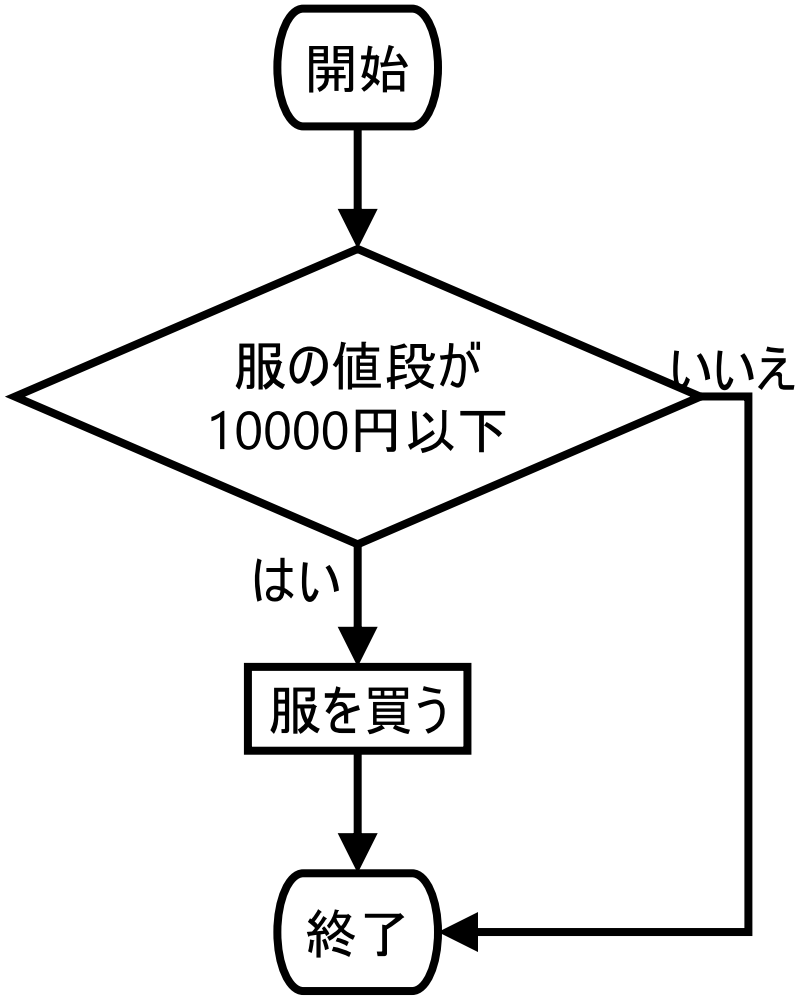
記号	意味
	開始と終了
	処理
	条件判断
	処理の流れ

順次処理の例

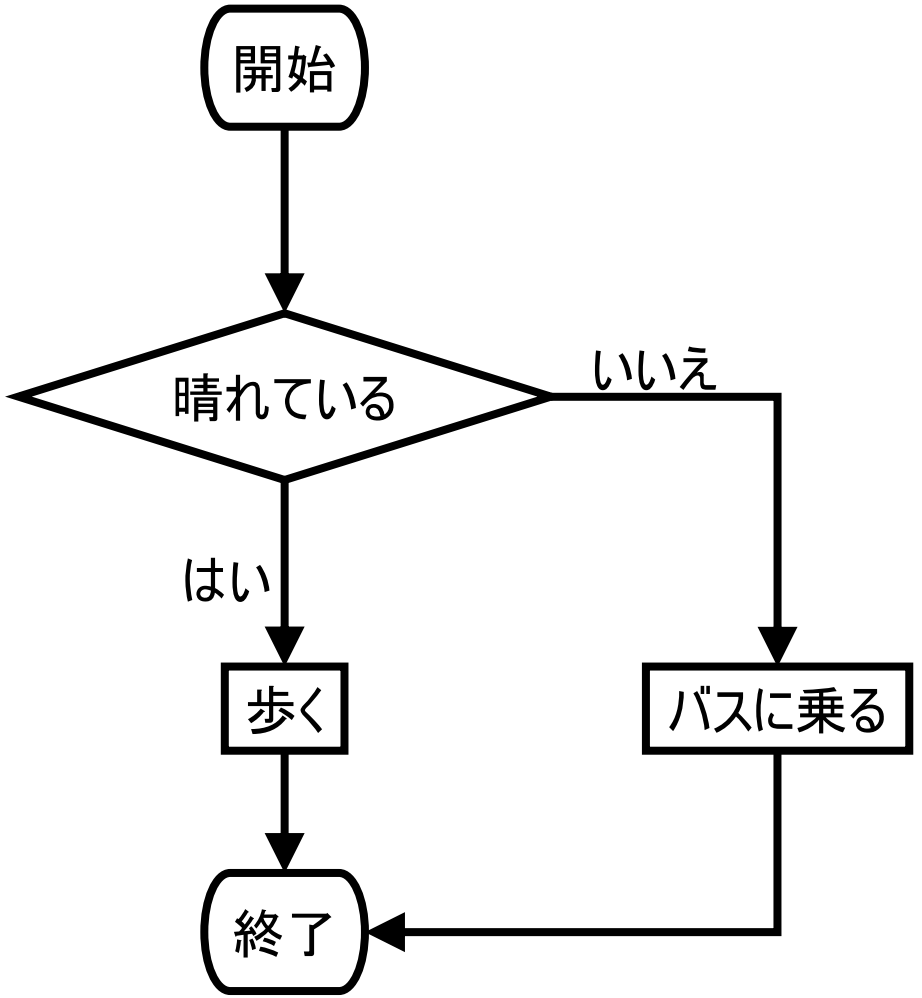


フローチャート[3](p. 119)

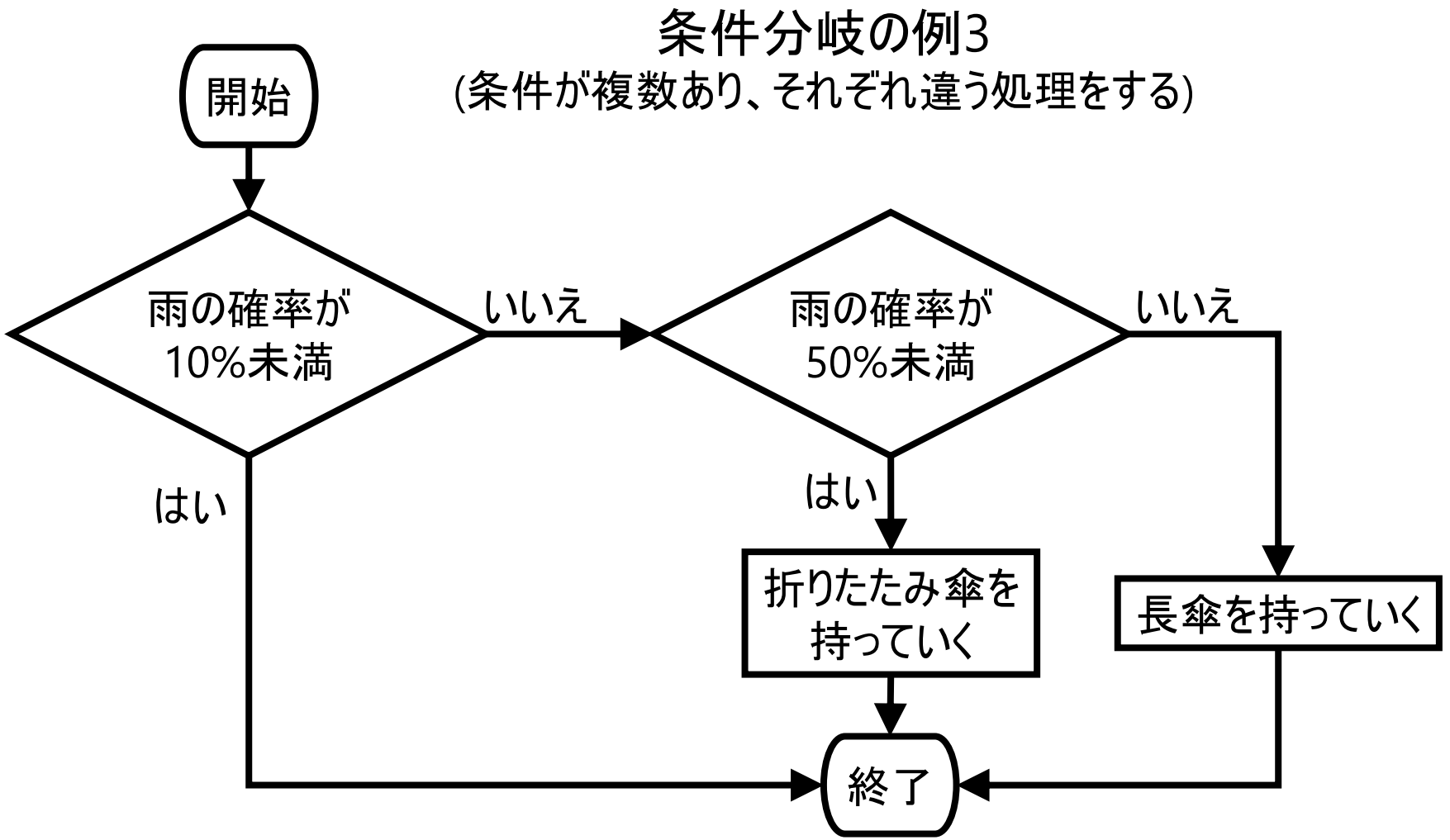
条件分岐の例1
(条件判断が「いいえ」の場合何もしない)



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)

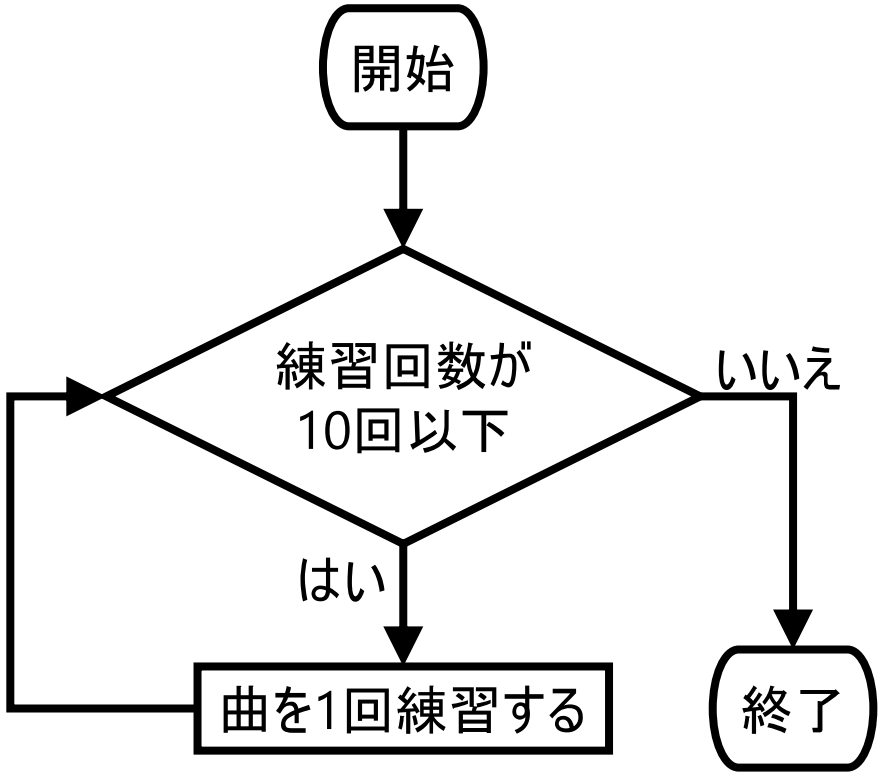


フローチャート[4](p. 119)

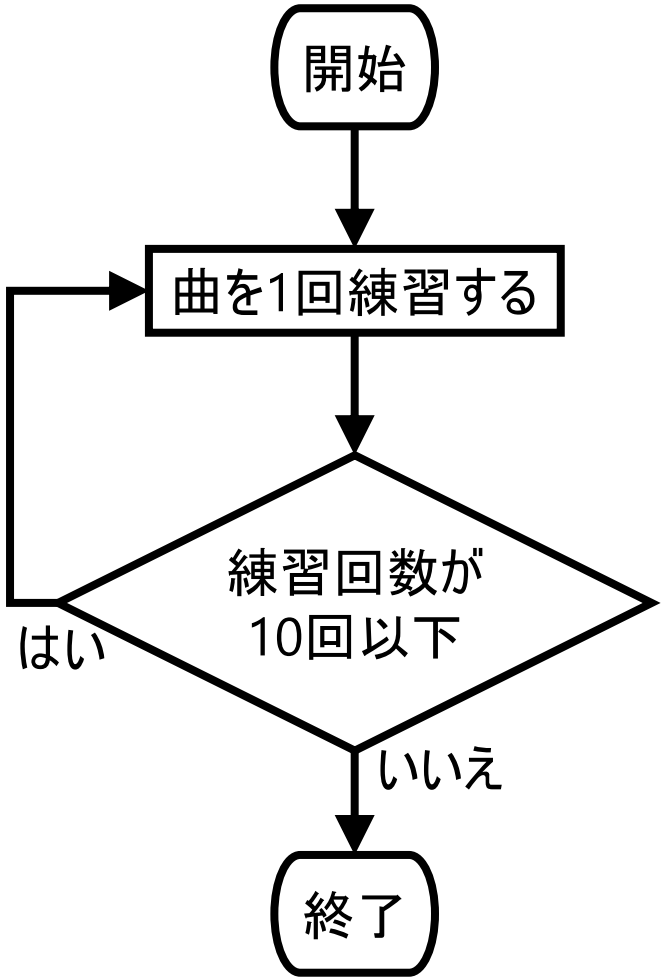


フローチャート[5](p. 119)

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)



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: 同じなので見つかった!

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、右半分は探索対象にする

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、右半分は探索対象にする

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、右半分は探索対象にする

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回するごとに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

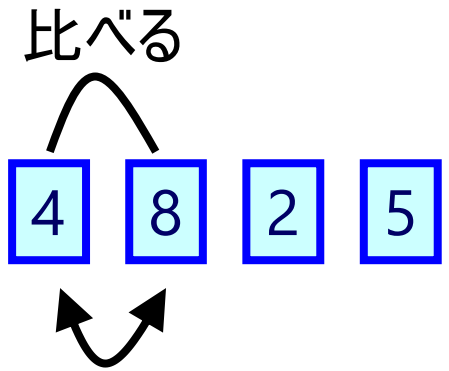
4	8	5	2

- 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(8, 4, 5)で同じようにする

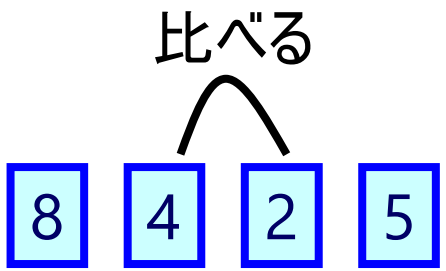
バブルソート[1](p. 124)

- 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

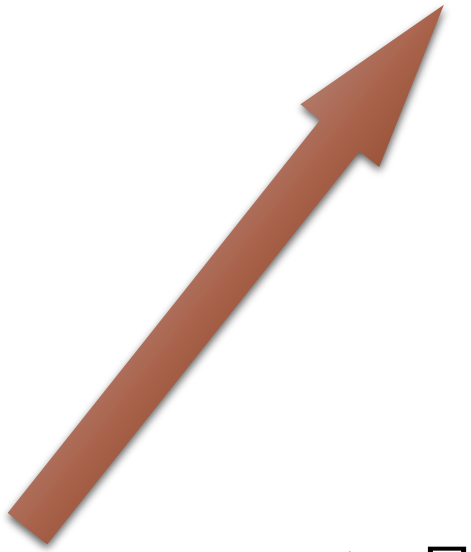
バブルソート[2](p. 124)

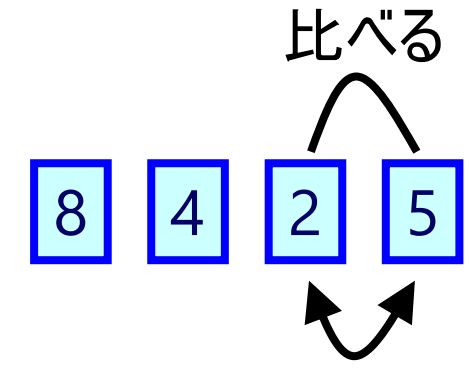


4より8の方が大きいので交換

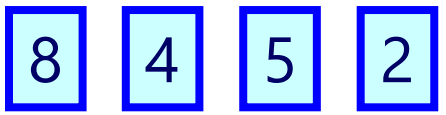


2より4の方が大きいのでそのまま





2より5の方が大きいので交換



- 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

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

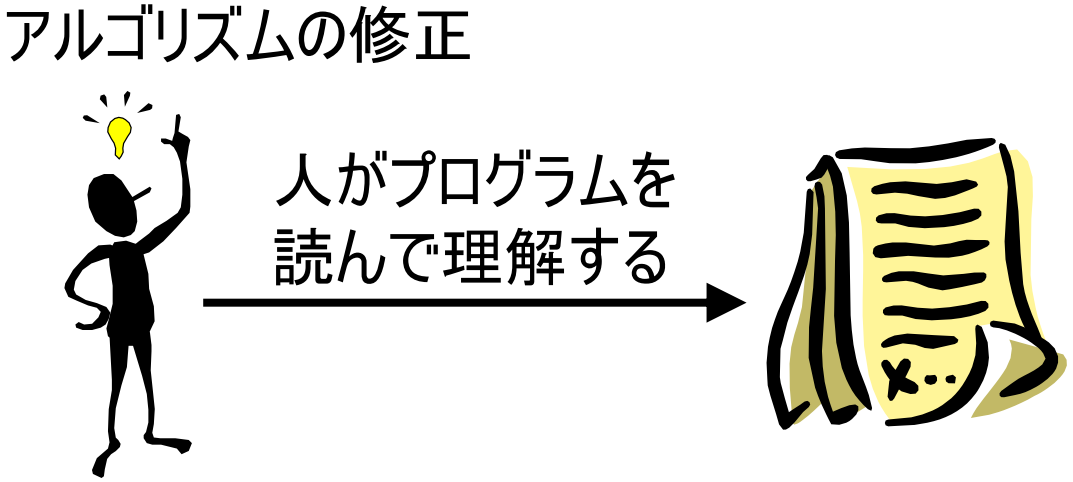
※同じコンピュータで同じアルゴリズムの処理をしても、そのときどきで処理に必要な時間が異なる

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



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: $N/2+(N-1)\log_2 N$

  } N: 並べ替える数の個数

  ※$\log_2 N$: Nを2^kとしたときの「k」

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: $N/2+(N-1)\log_2 N = N/2 + N\log_2 N - \log 2N$
      →$N\log_2 N$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(...)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logN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までをかけあわせた数), 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0.0000001秒かかるコンピュータ)

n	logn	n	nlogn	n^2	n^3	2^n	n!
10	0.0000003	0.000001	0.000003	0.00001	0.0001	0.0001024	0.36
20	0.0000004	0.000002	0.000009	0.00004	0.0008	0.1048576	7700年
30	0.0000005	0.000003	0.000015	0.00009	0.0027	107	8×10^{18}年
50	0.0000006	0.000005	0.000028	0.00025	0.012	3.4年	
100	0.0000007	0.00001	0.000066	0.001	0.1	4×10^{15}年	
10000	0.0000013	0.001	0.0133	10	1.2日		
1000000	0.0000023	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 ≒ 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こと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
$(30 - 1)! \times 1/100万 = 8.8 \times 10^{30}回 \times 1/100万 \fallingdotseq$ 3800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(1/5)

- 実習のため、24102教室に集合