アルゴリズムとデータ構造
第1週

掛下　哲郎
kake@is.saga-u.ac.jp

学習教育目標

基本的な「データ構造」と「アルゴリズム」について，概念，考え方，効率を理解

1. Big O記法で，アルゴリズムの計算・記憶量を評価できる
2. アルゴリズムとデータ構造の相互依存関係を理解できる
3. 与えられたアルゴリズムの動作を説明できる
4. 基本的なアルゴリズムを理解できる
5. 基本的なデータ構造を理解できる
6. アルゴリズムを用いた問題解決法を理解できる

講義スケジュール：計10回

週	講義計画
1－2	導入
3	探索問題
4－5	基本的なデータ構造
6	動的探索問題とデータ構造
7	アルゴリズム演習（第1回）
8－9	データの整列
10－11	グラフアルゴリズム

第1週で学ぶこと

- アルゴリズムとは何か？
- データ構造とは何か？
- アルゴリズムの評価尺度と評価方法
- Big O 記法（オーダー記法）

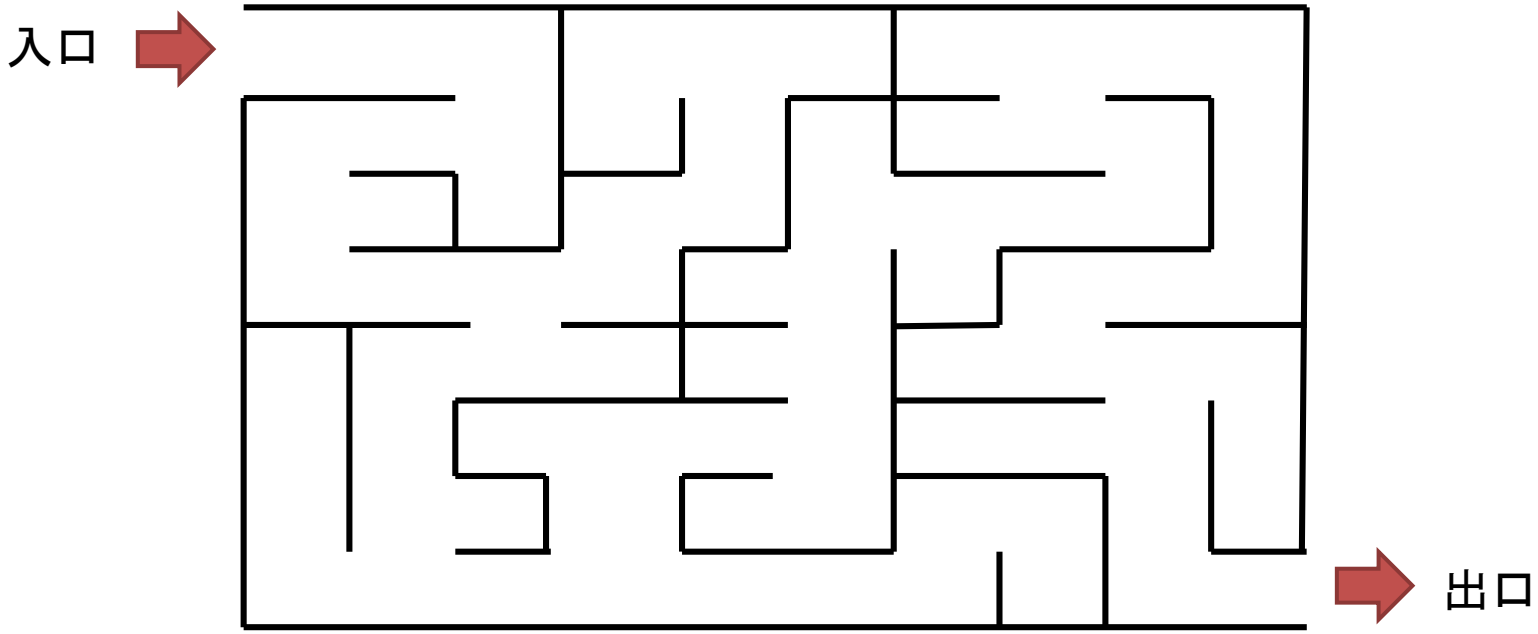
アルゴリズムとは？

Algorithm

- 与えられた問題を解くための明確に定義された操作からなる有限の手順
  - 自然言語（日本語など）で記述するのが普通.
    - プログラムよりも抽象的な記述
    - プログラムとは違い，作成者の意図を記述する.
  - 有限回の操作で終了.（有限性）
  - 各操作は明確に定義されており，誰が実行しても同じ結果が得られる.（確定性）
- 同じ問題でも何種類かのアルゴリズムがある.
  - アルゴリズムによって特徴が異なるので，状況に応じて適切なアルゴリズムを選ぶことが重要

複数のアルゴリズムがある例

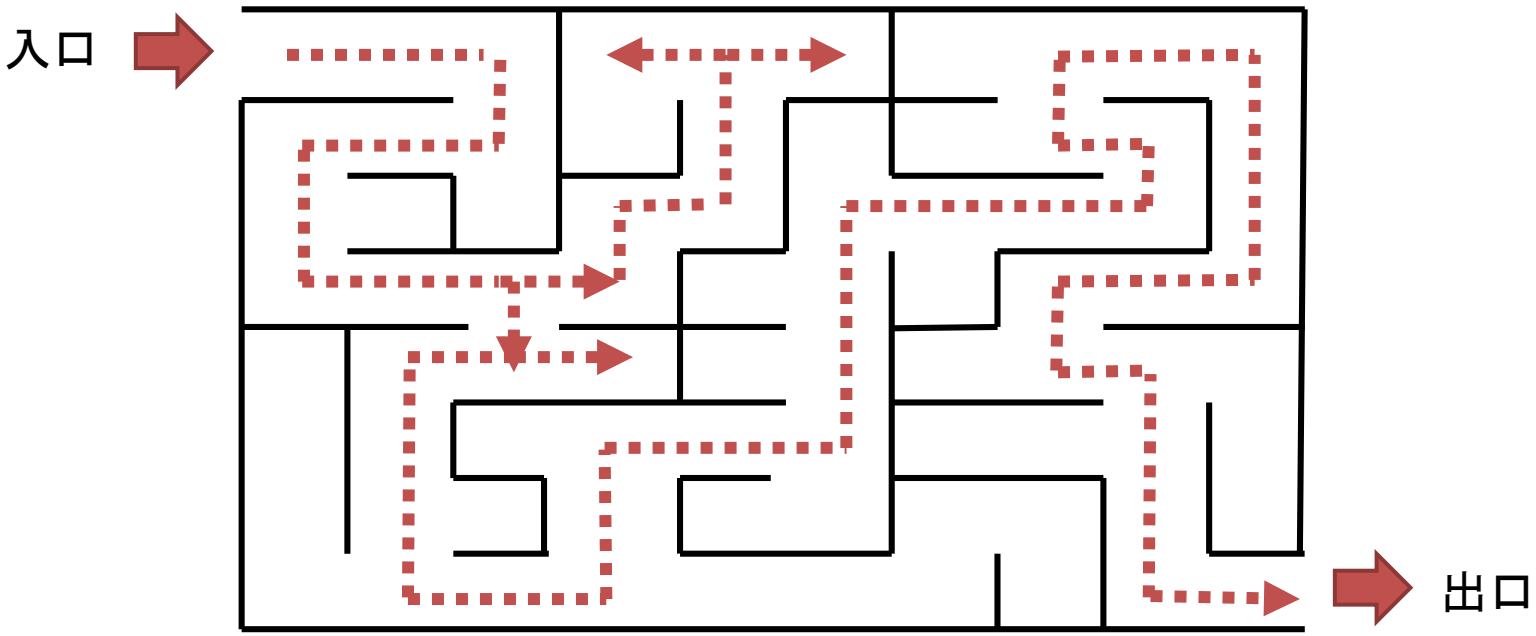
迷路探索問題



入口から入って出口に至る経路を探索する.

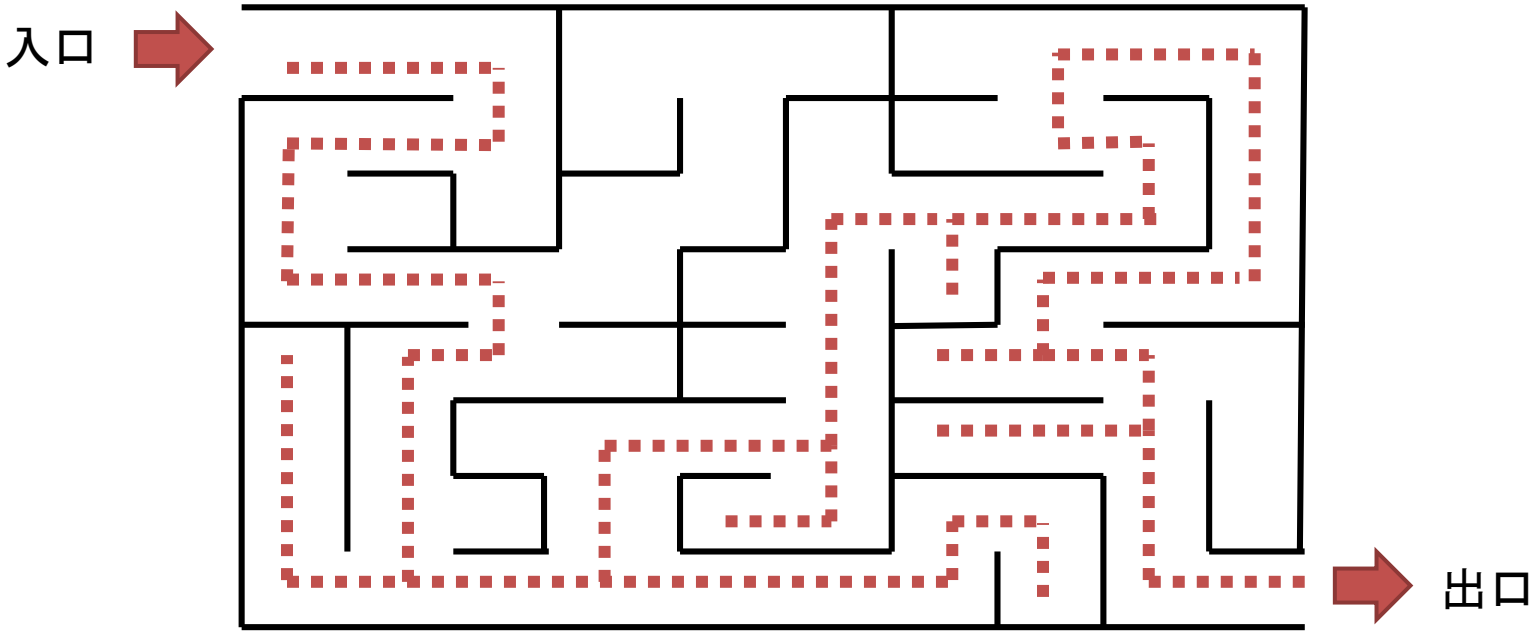
- 出口に至る経路があれば「成功」
- 出口に至る経路がなければ「失敗」

アルゴリズム1:しらみつぶしに探索



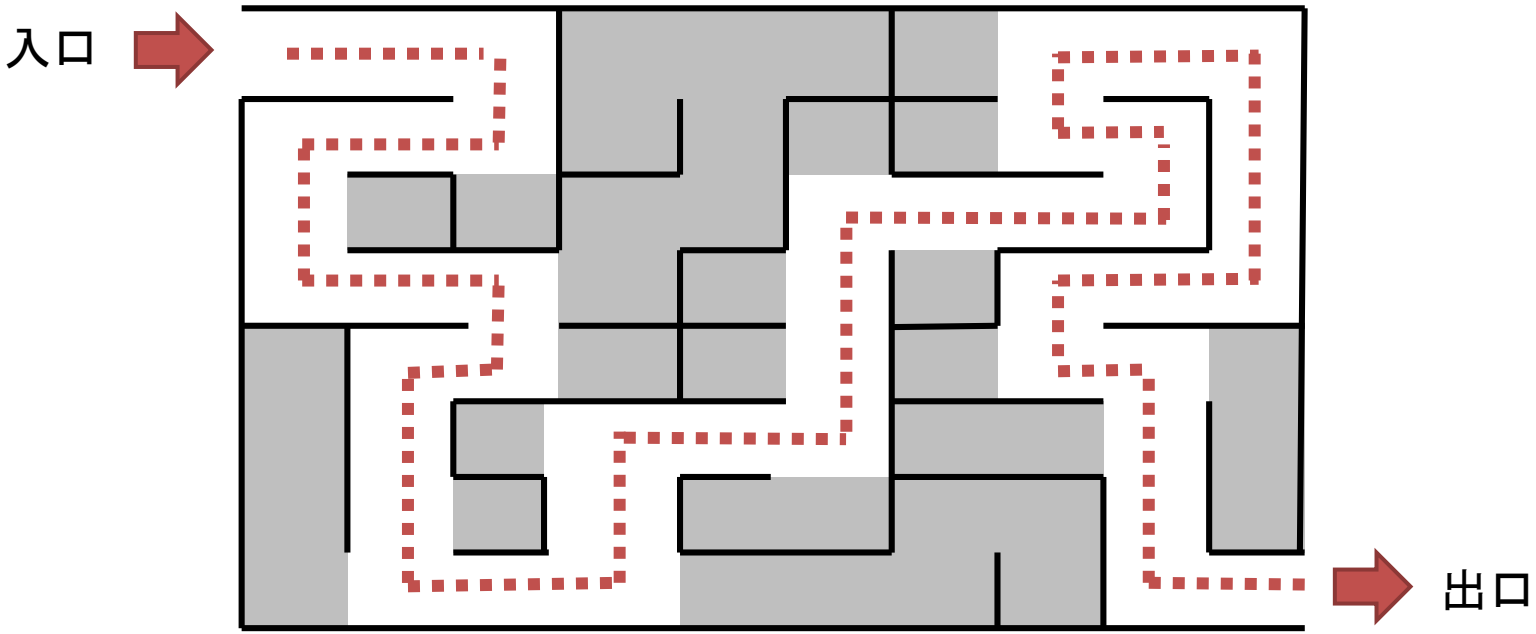
- 可能なすべての経路をしらみつぶしに探索する.
- 行き止まりになったら，別の経路を探す.
- 出口に到達したら,「成功」として終了.
- 全ての経路を探索したら,「失敗」として終了

アルゴリズム2：右手法



- 入口では右手で壁を触る．
- 右手で壁を触ったまま，その方向に進む．右手を壁から離さない．
- 出口に到達したら，「成功」として終了．
- 入口に到達したら，「失敗」として終了．

アルゴリズム3：行き止まりを埋める



- 行き止まりのマス（3方向以上が壁のマス）を埋める．
- 埋めたマスによって行き止まりになるマスがあればそれも埋める．
- 全てのマスが埋められたら，「失敗」として終了．
- 1つでもマスが残ったら，「成功」として終了．

データ構造とは?

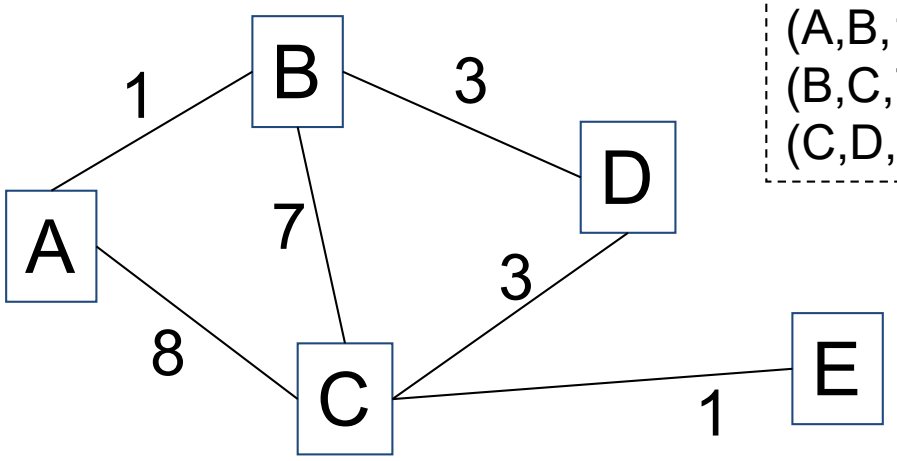
Data Structure

- 処理対象のデータを計算機で扱うために上手に組織化して記憶させるための方法
- データ構造の具体例
  - 変数，配列，構造体
  - 連結リスト，スタック，キュー，ヒープ
  - 二分探索木，平衡二分探索木，グラフ

N. Wirth
「アルゴリズム」＋「データ構造」＝「プログラム」
アルゴリズムとデータ構造は相互に依存する.

データ構造の例

- 最短経路問題：経路図が与えられたとき最短路を求める問題
  - アルゴリズム：どのように調べれば最短経路を探せるか？
  - データ構造：経路図をどのようなデータ形式で持つと合理的か.



一次元リスト形式

(A,B,1), (A,C,8),
(B,C,7), (B,D,3),
(C,D,3), (C,E,1)

二次元配列形式

	A	B	C	D	E
A	0	1	8	∞	∞
B	1	0	7	3	∞
C	8	7	0	3	1
D	∞	3	3	0	∞
E	∞	∞	1	∞	0

整理1：アルゴリズム，データ構造とは何か？

- アルゴリズム

  与えられた問題を解くための明確に定義された操作からなる有限の手順

- データ構造

  処理対象のデータを計算機で扱うために上手に組織化して保持するための方法

同じ問題でも，それを解くアルゴリズムはいくつもある

同じデータでも，それを表現するデータ構造はいくつもある

アルゴリズムとデータ構造は相互に依存している

アルゴリズムの評価

- アルゴリズムの良し悪しをはかる尺度
  1. **正当性**（正しく動作しなければならない）
  2. **計算時間**（短い時間で終了する方が良い）
  3. **記憶領域**（少ないメモリで動く方が良い）
  4. 容易な理解（分かりやすい方が良い）
  5. 容易な開発（プログラムにしやすい方が良い）
  6. 容易な保守（追加変更しやすい方が良い）

1〜3：データ構造とアルゴリズムで教育

4〜6：プログラミング概論・演習，ソフトウェア工学で教育

- *バランスが重要*

  全ての尺度で望ましいものを作れるとは限らない

アルゴリズムの効率

- 「時間計算量」と「領域計算量」が特に重要
  - 時間計算量（Time Complexity）
    - どれぐらい計算時間がかかるか
  - 領域計算量（Space Complexity）
    - どれぐらい記憶領域（メモリ，ハードディスク等）が必要か

計算量

- 計算に必要な時間は**入力サイズ n** に依存
  - 入力データが大きければ，より長い時間が必要
  - 計算時間をnの関数T(n)で表現
- 具体的な計算時間は計算機に依存
  - 仮想機械RAM (Random Access Machine) などを仮定
    - 記憶領域無限
    - メモリアクセス，四則演算，比較演算は単位時間で可能
- **漸近的計算量**に着目
  - 入力サイズ n が十分大きいときの計算時間
- nが同じでも処理時間が一定とは限らない
  - 入力データの性質にも依存する
  - 処理時間の平均値または**最悪値**を比較する

例：多項式の計算

- $P(x) = a_n x^n + a_{(n-1)} x^{(n-1)} + \ldots + a_1 x + a_0$
  - $a_n, a_{(n-1)}, \ldots, a_0, x_0$ を入力して，$P(x_0)$を計算
- 4次式の場合

  $a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0$
  - 律儀に計算

    → 掛け算10回+加算4回 = 四則演算**14**回
  - ホーナーの方法

    $(((a_4 x + a_3) x + a_2) x + a_1) x + a_0$

    → 四則演算**8**回

多項式の計算（n次式の場合）

- 律儀な方法
  - 掛け算 $n + (n-1) + (n-2) + \ldots + 2 + 1$ 回
  - 加算 n 回
  - 合計：$n(n+1)/2 + n = \mathbf{(n^2 + 3n)/2}$ 回
- ホーナーの方法
  - 掛け算 n 回
  - 加算 n 回
  - 合計：**$2n$** 回

n=10,000で，四則演算に要する時間が10^{-6}秒/回とすると
律儀な方法 → 50秒
ホーナーの方法 → **0.02秒!!**

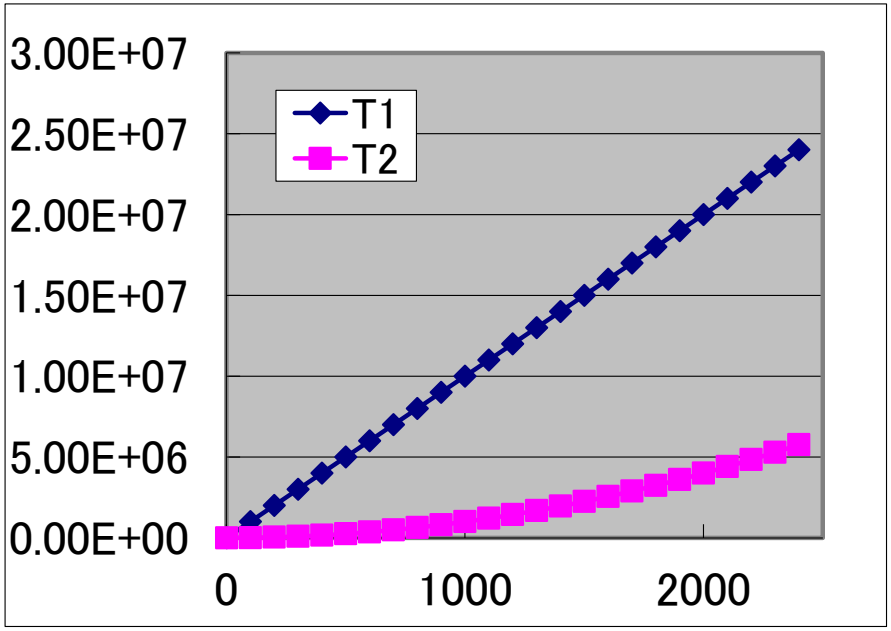
計算量の漸近的な評価（$n \to$ 大）

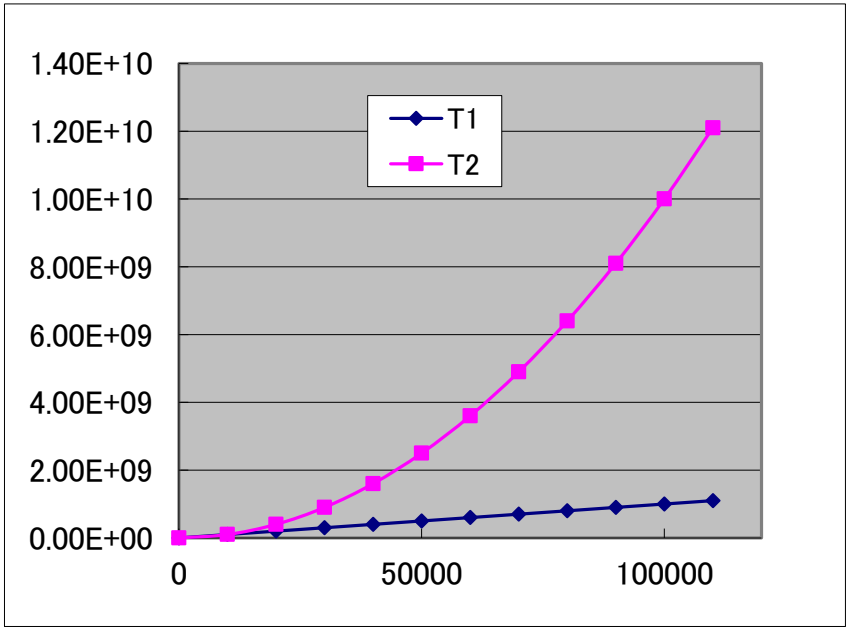
- 定数倍程度の差は気にしない
  - Mooreの法則：同じ価格のコンピュータの処理速度は5年で10倍
    - 数倍程度の処理速度の差は，長期的には無意味．
  - 大量データの処理時間の方が，少量データの処理時間より重要
    - 入力サイズnの増加に伴って計算時間がどのような増え方をするかを評価
- 例
  - $1 < \log n < n < n \log n < n^2 < n^3 < 2^n < n!$

$T_1(n) = 10000n$, $T_2(n) = n^2$ の場合でも同様か？

n=10000でT2はT1に追いつく

n≫10000ではT2はT1より
はるかに大きくなる





$T_1(n) = 10000n$, $T_2(n) = n^2$ の場合でも同様

様々な関数の相互比較

n	log n	n log n	n^2	n^3	2^n	n!
1	0	0	1	1	2	1
2	0	0	4	8	4	2
5	1	3	25	125	32	120
10	1	10	100	1,000	1,024	3,628,800
20	1	26	400	8,000	1,048,576	2.43E+18
50	2	85	2,500	125,000	1.13E+15	3.04E+64
100	2	200	10,000	1,000,000	1.27E+30	9.3E+157
200	2	460	40,000	8,000,000	1.61E+60	:
500	3	1,349	250,000	1.25E+08	3.3E+150	:
1000	3	3,000	1,000,000	1.00E+09	1.1E+301	:

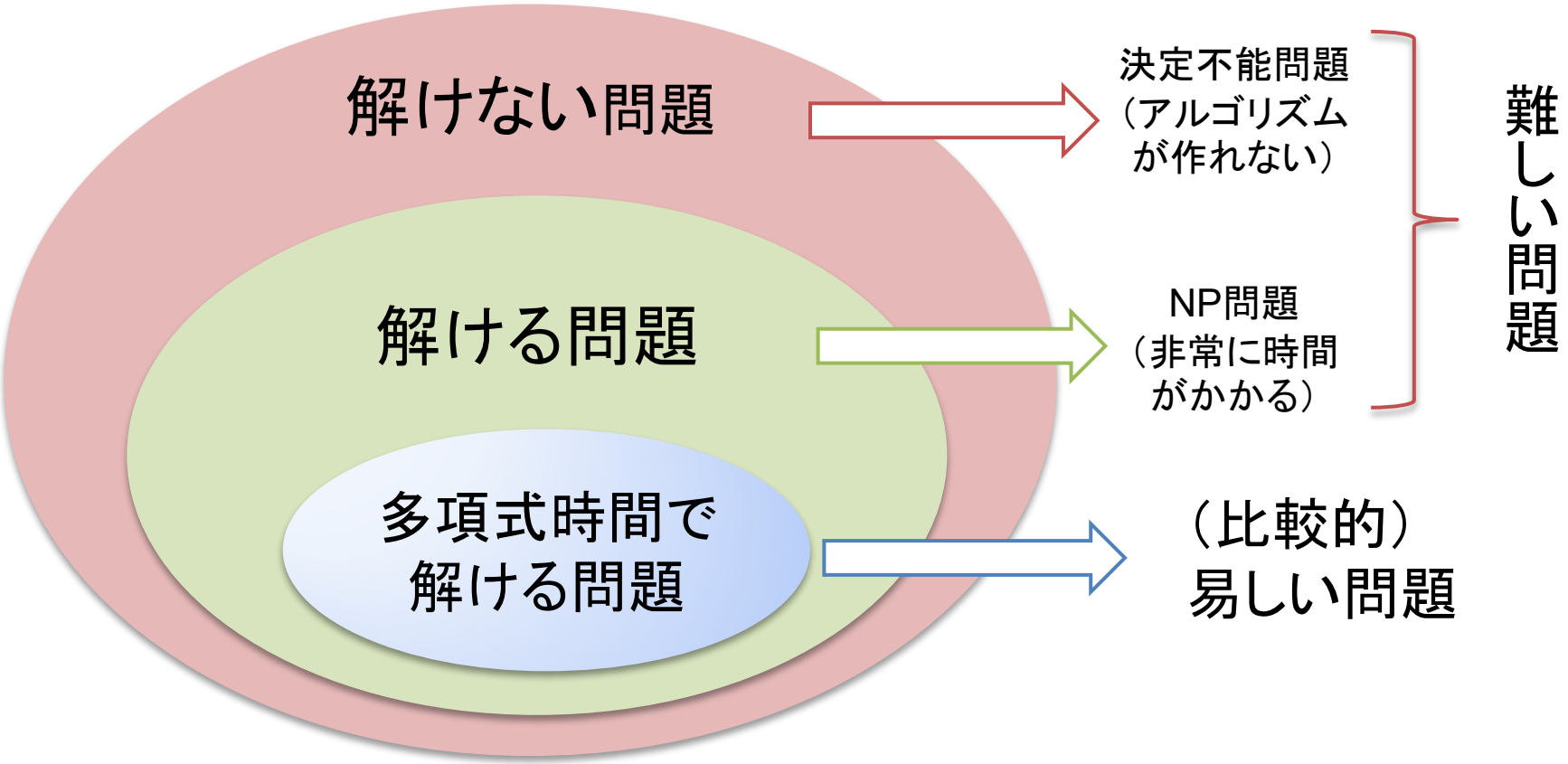
小数点以下四捨五入

1.1×10^{301}

評価の分かれ目
「多項式時間」と「指数時間」

- 多項式時間アルゴリズム
  - 計算量が多項式で表現 → 我慢できる速度（？）
  - 1, log n, n, n log n, n^2, n^3
- 指数時間アルゴリズム
  - 計算量が指数関数等で表現 → 現実的でない
  - 2^n, n!
- 例：n^2 と 2^n
  - 1命令を10^{-8} 秒で処理．n=100のとき
    - n^2 → 10^{-4}秒
    - 2^n → 4×10^{14}年

機械的に記述できる問題



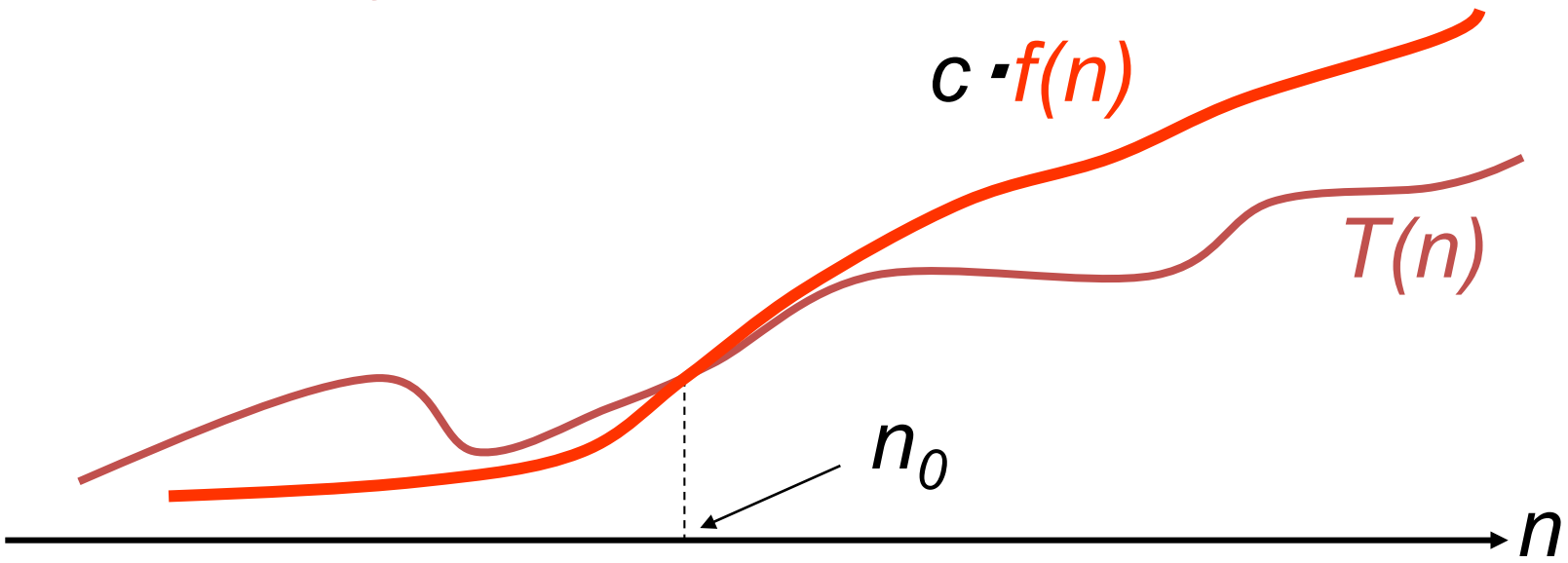
整理2

- アルゴリズムの評価尺度
  - 時間、領域
  - 容易性（理解、開発、保守）
- 評価方法
  - 仮想的な計算機上で計算
  - 入力サイズを n とし、その関数で表現
  - 漸近的な振る舞いを議論（n→大）

オーダー記法（Big O 記法）

- 計算量の上限値の評価 O(*f(n)*)
  - 定義：計算量*T(n)*がオーダー*f(n)*であるとは，
    正の定数*c*とn_0が存在して，n_0以上の任意の*n*に対して
    $T(n) \leqq c \cdot f(n)$ が成立すること．

    $T(n)$ = O($f(n)$) と書く



オーダーの例

- *10000*は O(1)
  $n_0=0 \leqq n$ のとき $10000 \leqq c1$ $(c = 10000)$
- $2n^2 + n + 5$ は $O(n^2)$
  $n_0=5 \leqq n$ のとき $2n^2+n+5 \leqq cn^2$ $(c = 4)$
- $1000n^2+10000n$ も $O(n^2)$
  $n_0=10000 \leqq n$ のとき
  $1000n^2+10000n \leqq cn^2$ $(c = 1001)$

アルゴリズムの例（最大値を探索）

1. 整数データの列を取得
2. 最初のデータを「仮最大値」とする
3. 2番目のデータから始めて，最後のデータになるまで以下を繰り返す
   3.1 当該データが「仮最大値」よりも大きいならば，「仮最大値」を更新する
4. 「仮最大値」を最大値として返す

C++で記述した例

```
//
// 最大値を探索
//
#define N 10
#include <iostream>

int search(){
   int x[N];
   int max;

   // 1. 整数データの列を取得
   for(int i=0; i<N; i++) cin >> x[i];

   // 2. 最初のデータを「仮最大値」
   // とする
   max = x[0];

   // 3. 2番目のデータから始めて，
   // 最後のデータになるまで以下を
   // 繰り返す
   for(int i=1; i<N; i++)

      // 3.1　当該データが「仮最大
      // 値」よりも大きいならば，「仮
      // 最大値」を更新する
      if(max<x[i]) max=x[i];

   // 4.「仮最大値」を最大値として返
   // す
   return max;
}
```

オーダーの計算

1. 整数データの列を取得 — O(n)時間
2. 最初のデータを「仮最大値」とする — O(1)時間
3. 2番目のデータから始めて，最後のデータになるまで以下の処理を繰り返す — 繰り返し回数=n-1

   条件分岐があるときは，すべての場合の時間の最大値を用いる．

   3.1 当該データが「仮最大値」よりも大きいならば，「仮最大値」を更新する — O(1)時間
4. 「仮最大値」を最大値として返す — O(1)時間

$$T(n) = O(n) + O(1) + (n-1) \times O(1) + O(1) = O(n)$$

今日のポイント

- アルゴリズム，データ構造の定義
- アルゴリズムを評価するための尺度と、その評価方法
- Big O 記法の定義とオーダー計算の具体例