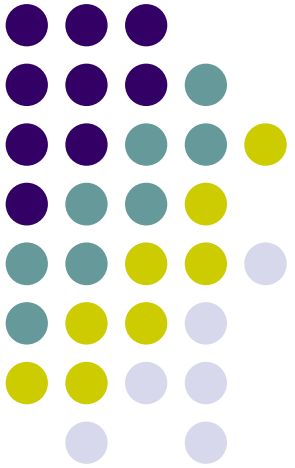
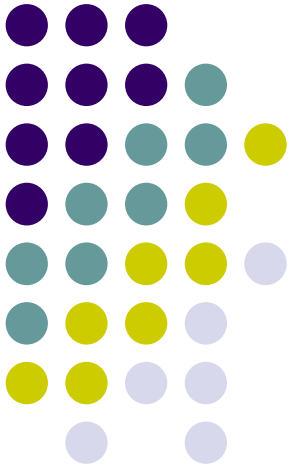
情報処理概論
後半5回目



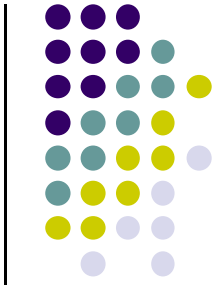
今日の内容

- 工学部で勉強したからこそ理解できるコンピュータに関する概念
  - パターンの表現(正規表現) ← 先週までで済み
  - **再帰呼出し** ←先週までで済み
  - **オブジェクト指向プログラミング**
  - **プログラムの実行時間**
- コンピュータ・ネットワーク (インターネット)
  - 来週

再帰呼出し(復習)

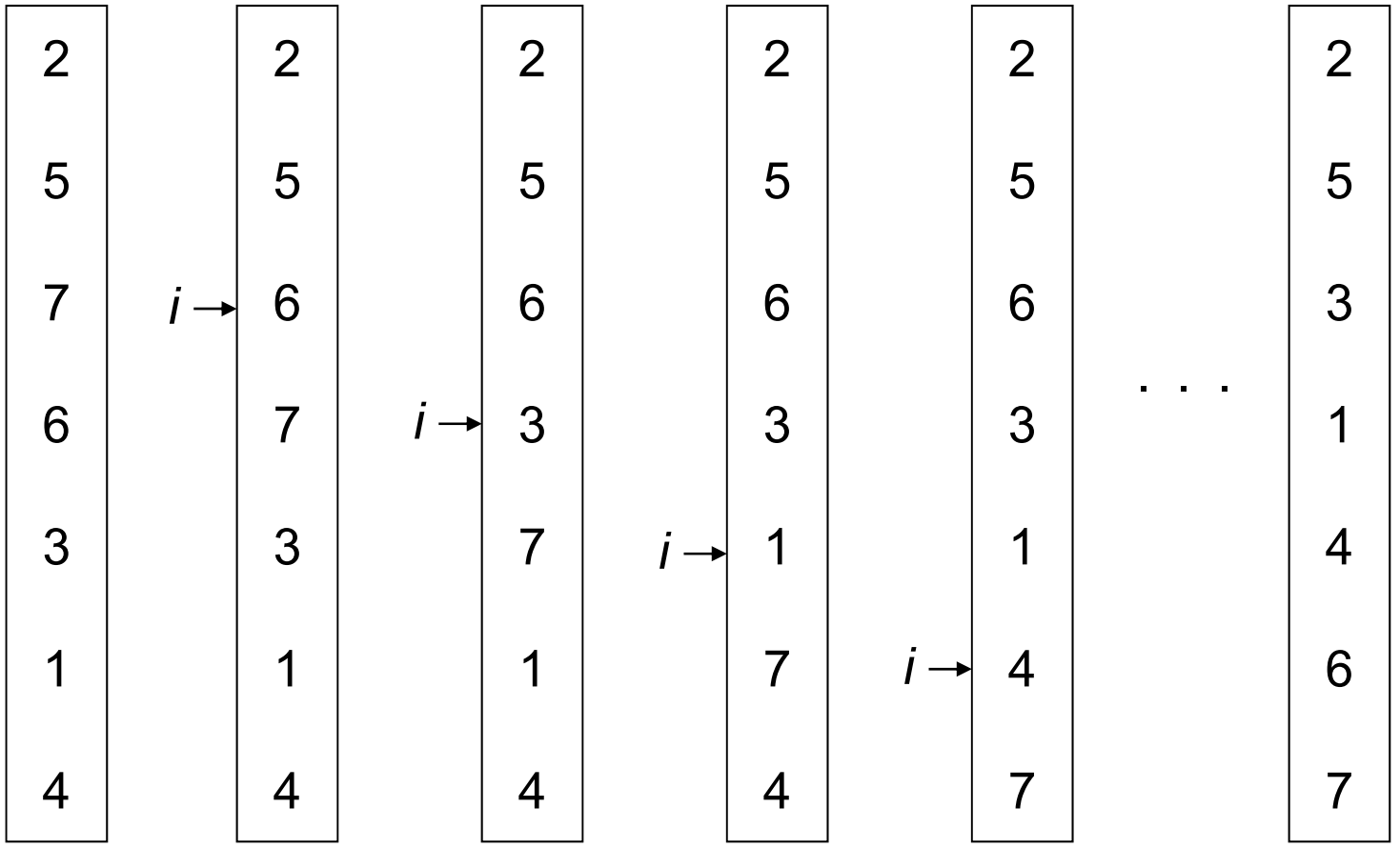


ソート

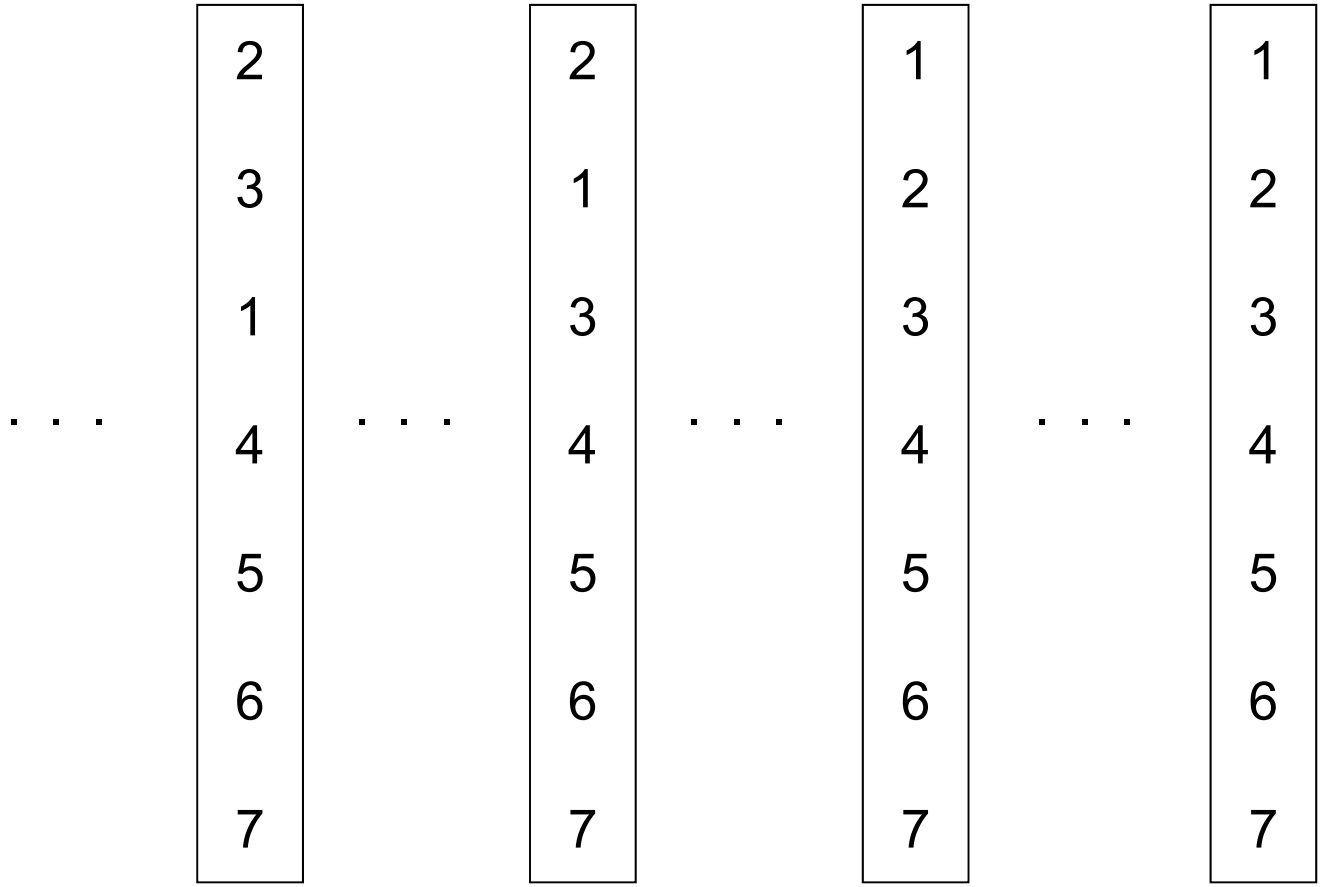


- ソートとは?
  - 与えられたリストLを値が小さい順(昇順)or大きい順(降順)に並べ替える
- 一番簡単なアルゴリズム: バブルソート
  - リストの要素数をNとする
  1. iを1からN-1まで変化させて
  2. (i+1)番目の要素がi番目の要素より小さければ入れ換える
  3. 1～2を入れ換えがなくなるまで繰り返す

バブルソートの動作例



バブルソートの動作例 (続き)



バブルソートの計算量

- 昇順になるようソートする
- 最良のケース
  - すべてが昇順に並んでいる場合: (N-1)回の比較
- 最悪のケース
  - すべてが降順に並んでいる場合:
    - 1巡目: (N-1)回の比較，(N-1)回の入換え
    - 2巡目: (N-1)回の比較，(N-2)回の入換え
    - :
    - N巡目: (N-1)回の比較，0回の入換え
  - N×(N-1)の比較，N×(N-1)/2の入換え

バブルソートの計算量 (続き)

- 問題規模(今の例ではN)に対する計算量について，Nの変化に対する計算量の変化を表わしたもの
- すなわち，
  - 加算的な定数は無視
  - 固定値の乗数も無視

した，計算量とNとの関係を

計算量のオーダ

と呼び，*O(…)*で表わす

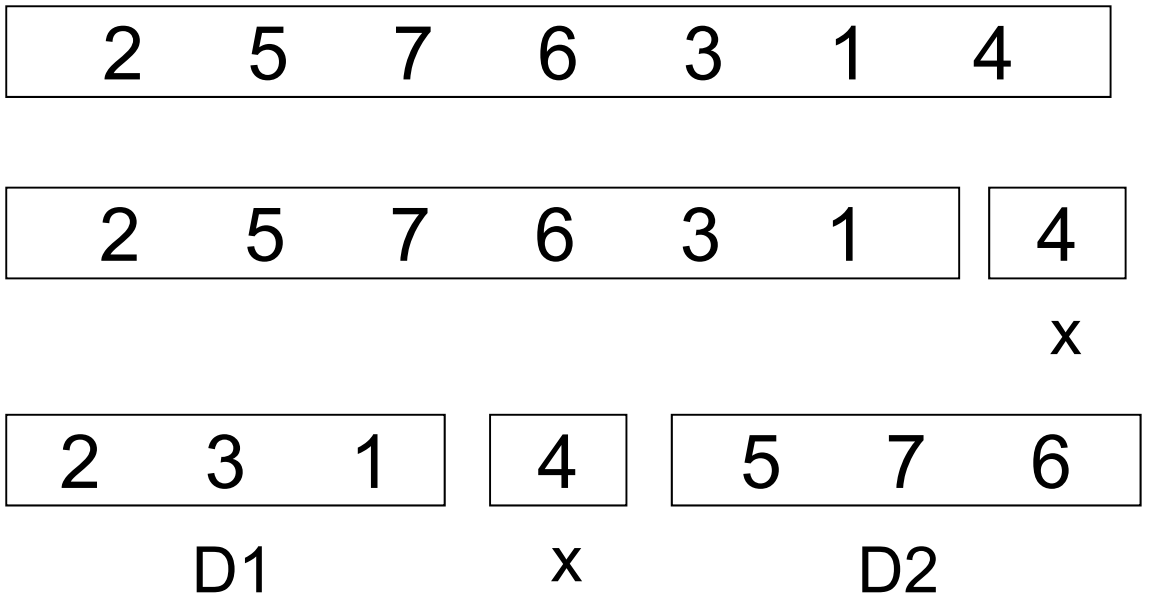
- バブルソートの計算量は $O(N^2)$

クイックソート

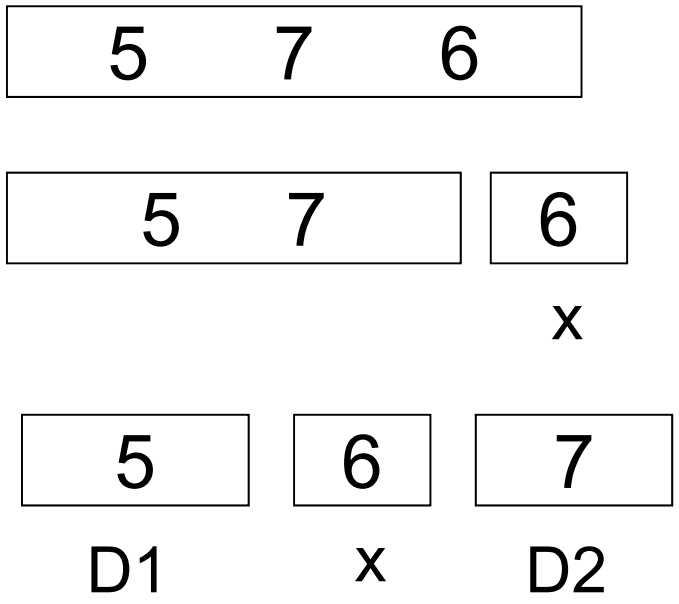
- アルゴリズム
  - Lの要素数が1ならば
    - 何もしない
  - そうでなければ
    - Lから一要素xを選択する
    - xより小さいLの要素をD1とする
    - x以上のLの要素をD2とする(ただしD2にはxは含まない)
    - D1をソートしてR1を得る
    - D2をソートしてR2を得る
    - R1, x, R2 の順にならべてソートされたリストとする

再帰呼出し

クイックソートの動作例



クイックソートの動作例 (続き)



クイックソートの計算量

- 1回の分割での比較回数: (N-1)
- 要素数Nのクイックソートの計算量

$$Q_N = N - 1 + Q_{N1} + Q_{N2}$$

  - ここでN1,N2は分割によって生じた左右の部分の長さ
  - N1+N2=N-1
- N1とN2は元のリストの並び方によって変るため計算量も異なるが，様々な場合が一様に発生するとして平均をとると

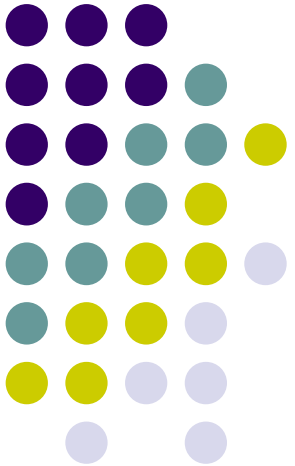
$$Q_N = N - 1 + 2(Q_0 + Q_1 + \cdots + Q_{N-1}) / N$$

$$Q_0 = 0, Q_1 = 1$$

- この漸化式を解くと

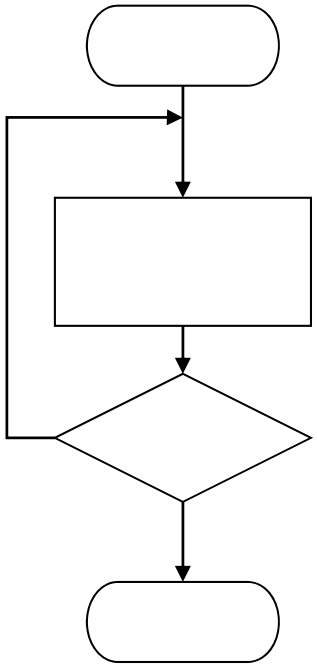
$$Q_N = 2N \log N \qquad \Rightarrow O(N \log N)$$

オブジェクト指向プログラミング



努力と根性があれば どんなプログラムも開発できるのか?

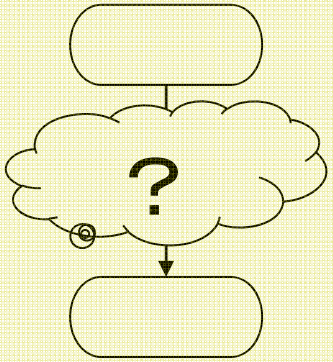
- 実験データ処理プログラム
- 事務処理プログラム



- 入力→処理→出力
- 基本的には上から下へ
- たまに分岐

- GUIプログラム
- 通信処理プログラム

- プログラムの流れと独立に外部からイベントが発生
- イベントを契機として様々な処理(流れがあらかじめ規定されない)

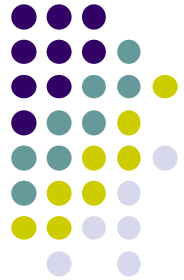


- フローチャートは書けない
- 「流れと分岐」思考の限界を越えている

オブジェクトとは?

- Object=「もの(現実世界に存在するものや概念)」
  - 目に見える物理実体
    - 本
    - 犬
    - TV
  - 目に見えない概念
    - 電波
    - 音楽
    - ローン計画
- これらの「現実世界に存在するものや概念」をソフトウェア上に表現したもの

「もの」のソフトウェア上の表現



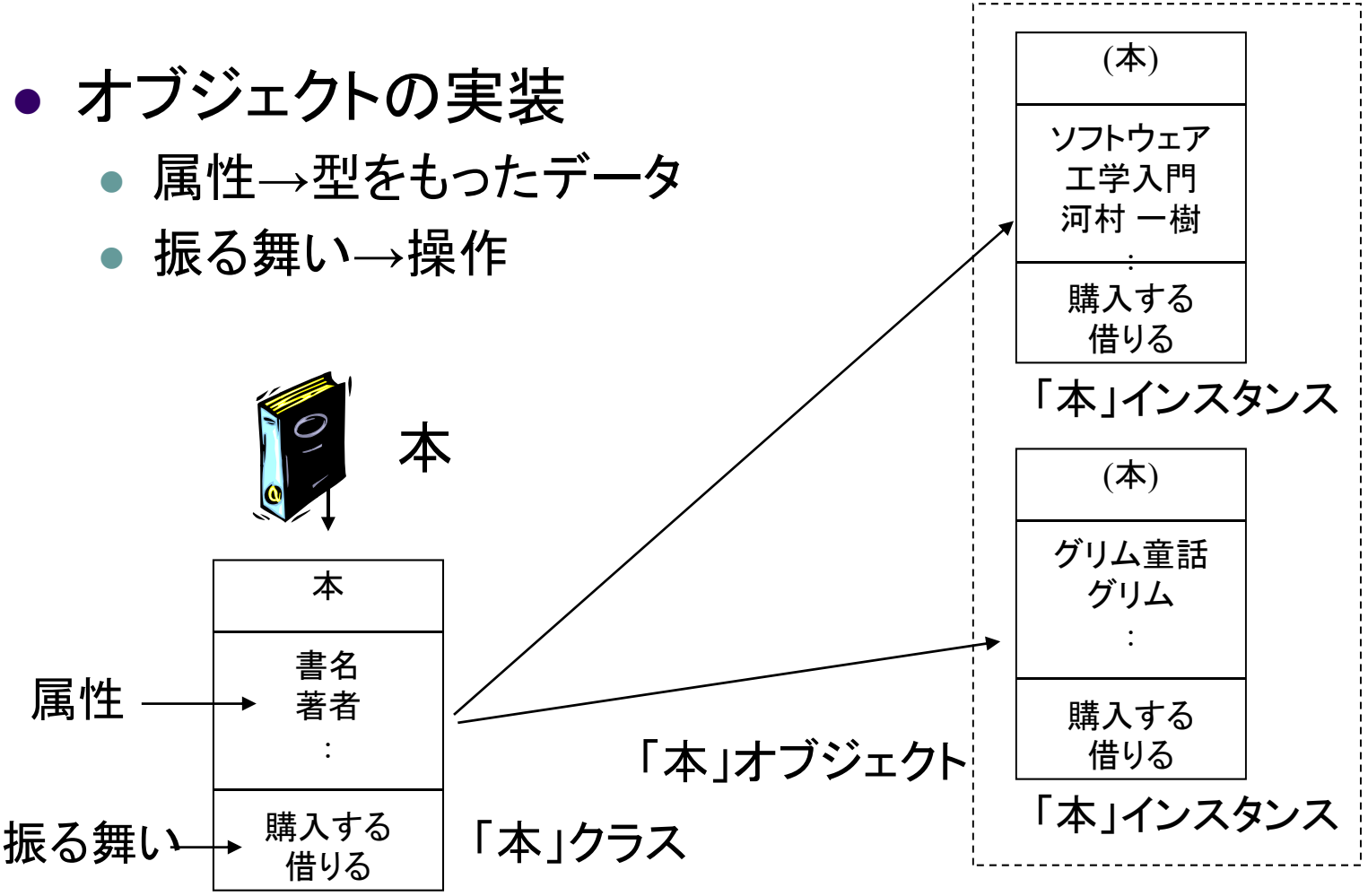
- 「もの」:=「属性」,「振る舞い」
- 例:「本」
  - 属性
    - 「書名」,「著者」,「出版社」,「価格」,「在庫」,．．．
  - 振る舞い
    - 「購入する」,「借りる」,．．．

「もの」のソフトウェア上の表現(続き)

- クラスとオブジェクト
  - 「本」という(一般の)もの=クラス
  - 「本」の具体化したもの=オブジェクト
  - ある特定の「本」の具体化したもの=インスタンス
  - クラスはオブジェクトの雛形

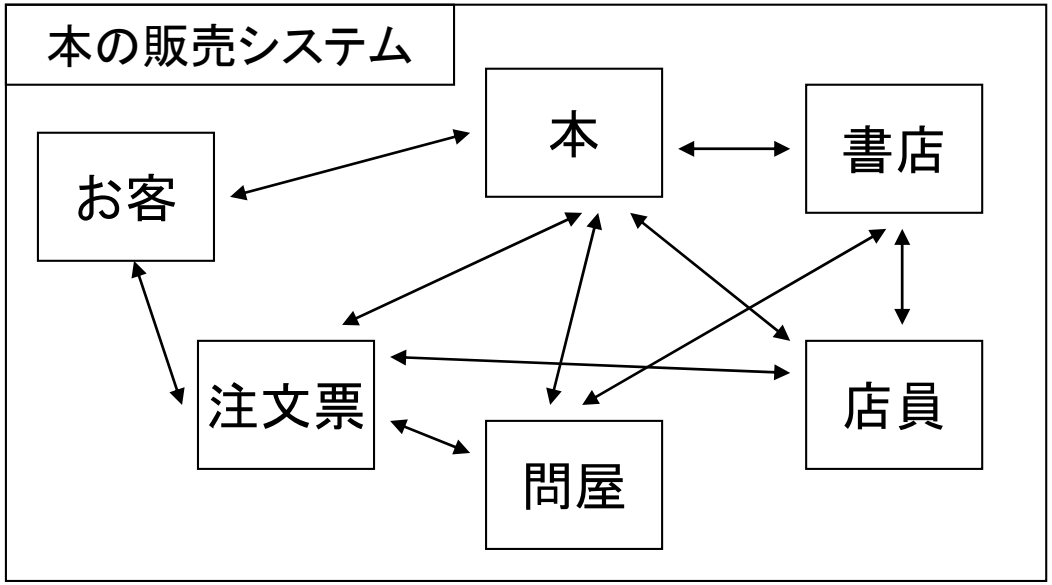
「もの」のソフトウェア上の表現 (続き)

- オブジェクトの実装
  - 属性→型をもったデータ
  - 振る舞い→操作



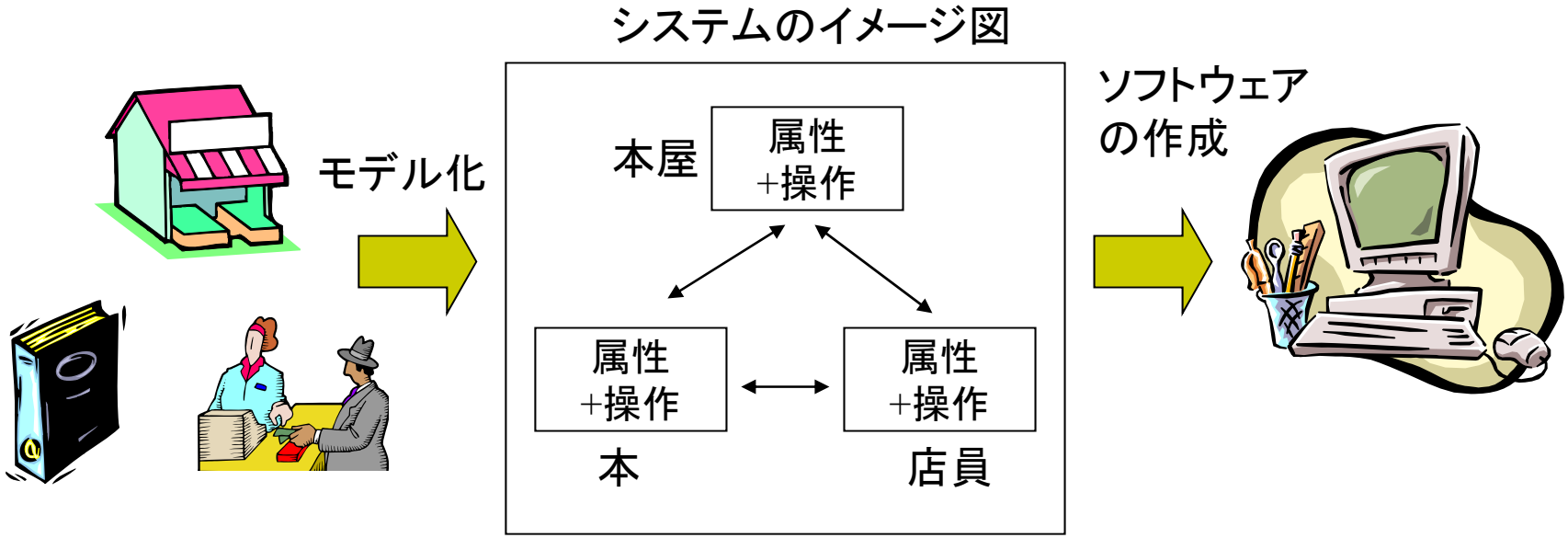
オブジェクト指向によるシステム開発

- オブジェクトという単位でシステムを分割・構成
- いろいろなオブジェクトが相互にメッセージを送りあって協調動作

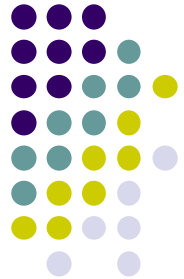


ソフトウェアの組織化，モデル化

- 構造化手法における「要求分析/設計」に相当
- 現実の「もの」(の必要な部分)をコンピュータ上に映す=モデル化



いきなり例題

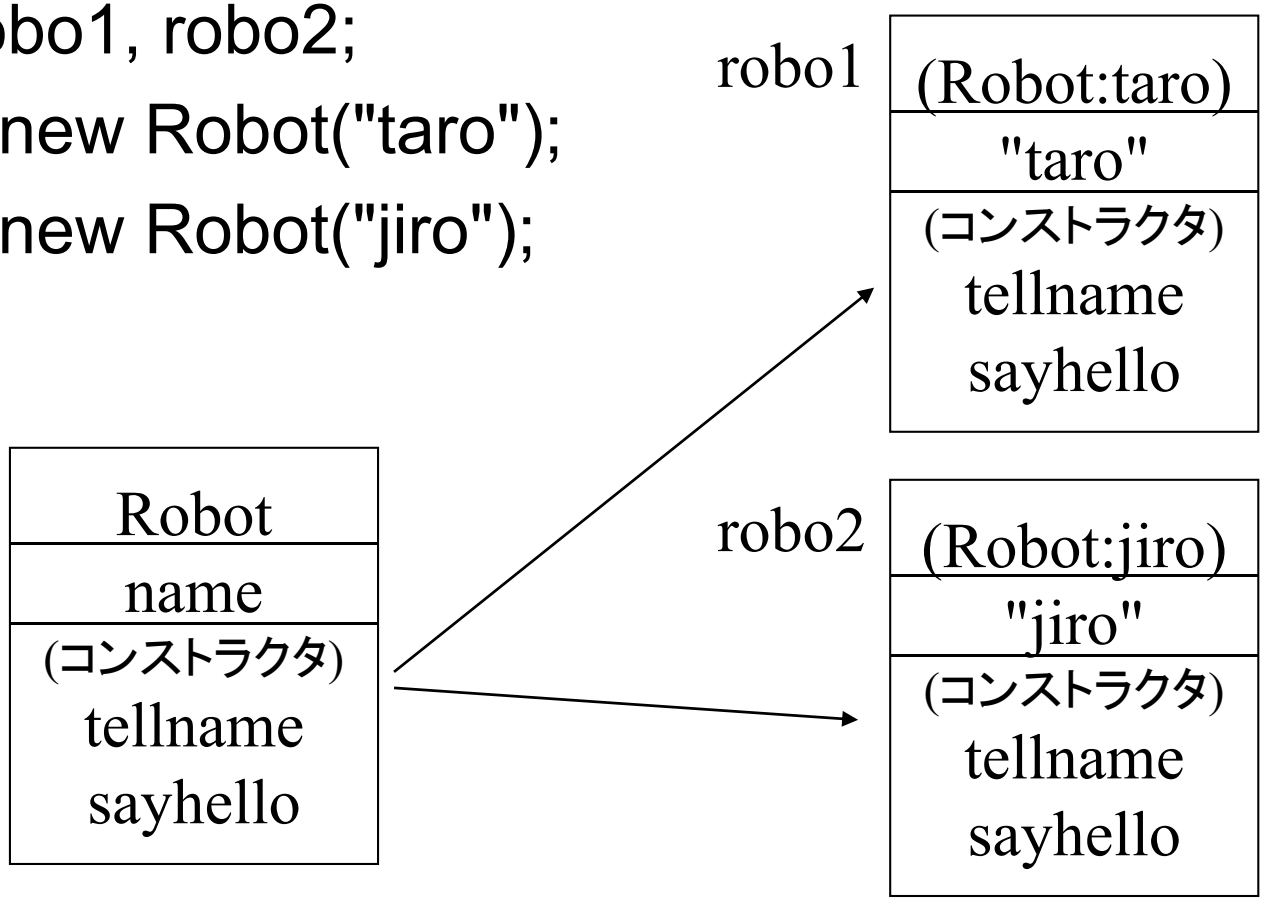


- 超単純ロボット

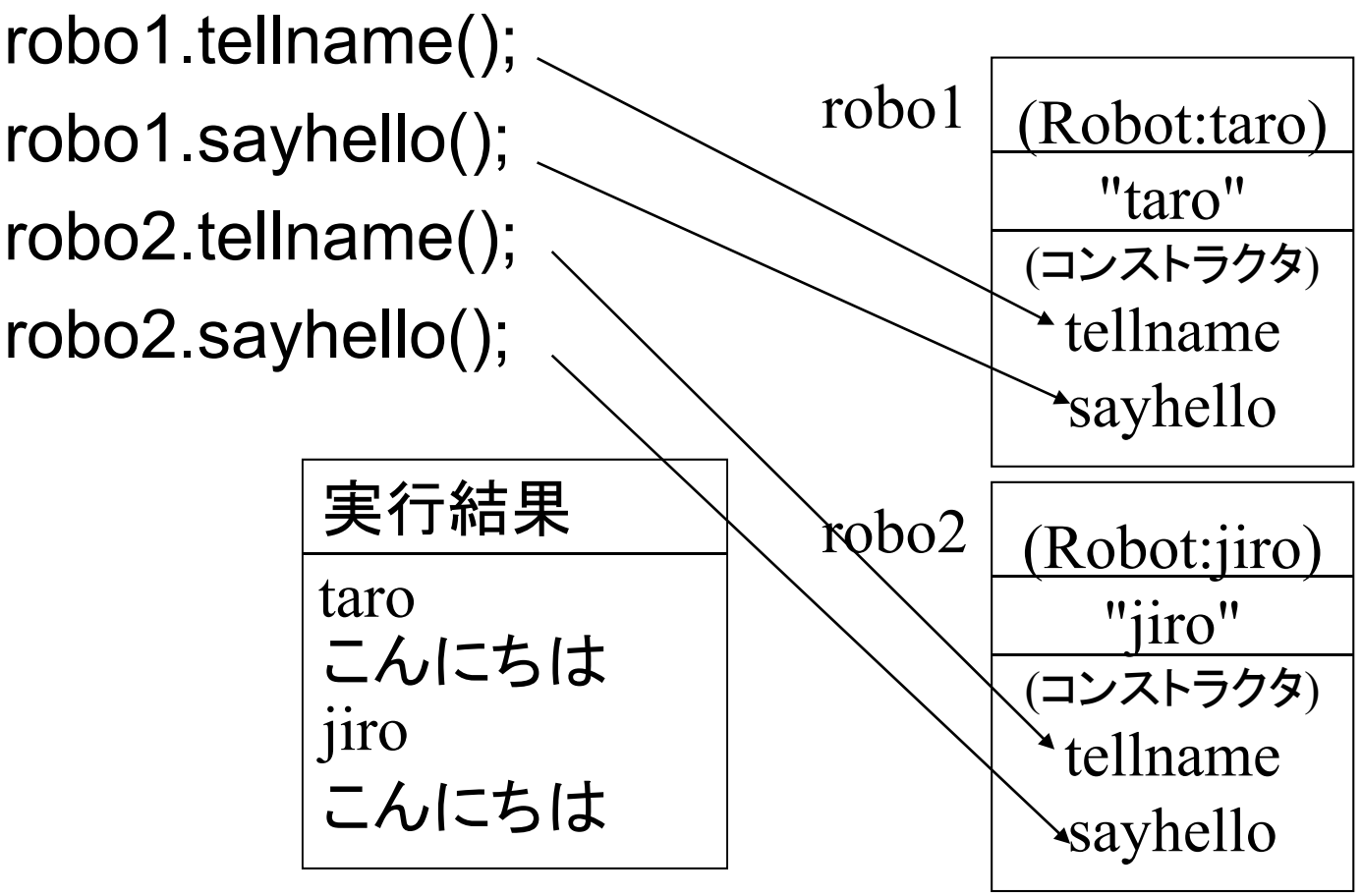
```
class Robot {
  String name;                                          属性
  Robot(string nm) { name = nm; }
  void tellname() {System.out.println(name);}
  void sayhello() {System.out.println("こんにちは");
                                                振る舞い(メソッド)
}
```

インスタンスの生成

```
Robot robo1, robo2;
robo1 = new Robot("taro");
robo2 = new Robot("jiro");
```



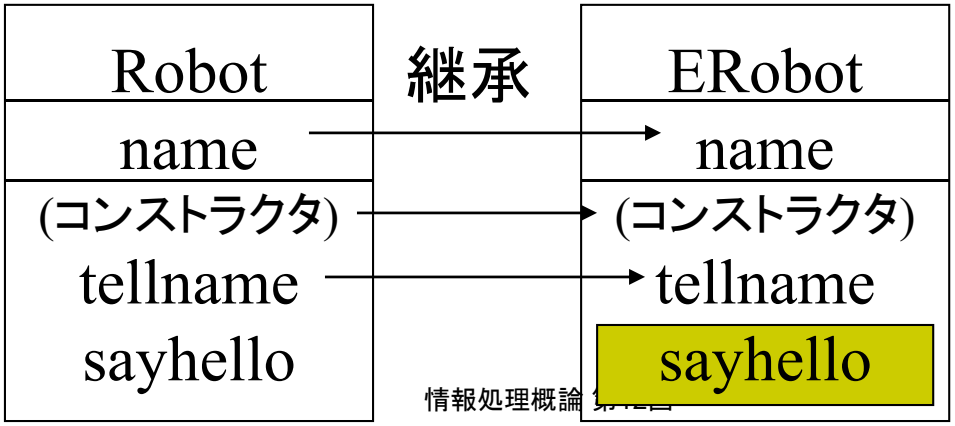
メソッドの呼出し


robo1.tellname();
robo1.sayhello();
robo2.tellname();
robo2.sayhello();
robo1
(Robot:taro)
"taro"
(コンストラクタ)
tellname
sayhello
robo2
(Robot:jiro)
"jiro"
(コンストラクタ)
tellname
sayhello
実行結果
taro
こんにちは
jiro
こんにちは

サブクラスの定義

- 英語版Robot

```
class ERobot extends Robot {
  ERobot(String nm) { super(nm); }
  void sayhello() { System.out.println("Hello"); }
}
```



サブクラスの利用

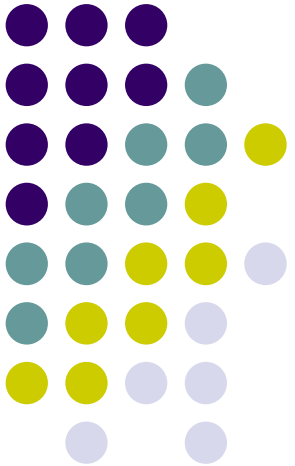
```
ERobot robo3
robo3 = new ERobot("bob");
robo3.tellname();
robo3.sayhello();
```

実行結果
bob Hello

実際のオブジェクト指向プログラミング環境

- 初期のオブジェクト指向プログラミング言語(環境)
  → Smalltalk
  - ゼロックスのパロアルト研究所でアラン・ケイが提唱・開発
- 既存言語をオブジェクト指向に
  - C + オブジェクト指向 → Objective C, C++
  - Lisp + オブジェクト指向 → CLOS
- 実用を意識したオブジェクト指向プログラミング言語
  - Java
  - Ruby
  - Python
  - 近年開発されるプログラミング言語は，ほとんどがオブジェクト指向的要素を備えている

プログラム実行時間



コンピュータは何でも計算できるか?

- チャーチ-マルコフ-チューリングの命題
  - 正確に説明できる手順なら何でも標準的なプログラミング言語でコーディングできる
  - プログラムを作ることができる

プログラムが作れれば,
何でも計算できるのか?

計算できない3つの場合

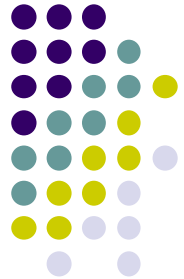
- プログラムの実行時間があまりにも長すぎる
  - 計算の手順が多すぎて，プログラムを開始してから答がでるまでに非常に長い時間(時に宇宙の寿命より長い)がかかる
- 計算不能
  - どんなコンピュータを使っても，計算できない

プログラムを作ることはできる

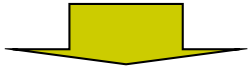
- プログラムの書き方がわからない
  - どのような計算手順によってその問題を解けるかわからない

プログラムを作ることができない

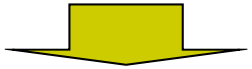
プログラムの実行時間



- コンピュータは高速



- しかし，どんな問題に対しても，一瞬にして答を出してくれるわけではない



- ある仕事をコンピュータにさせようという時に，どの位の時間でその仕事を完了できるかの見積りが重要
  - 結果を利用したい時より前に完了できそうか

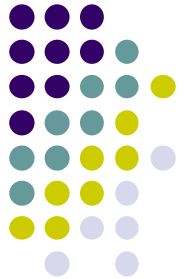
計算容易と計算困難

- 計算容易な計算
  - 大量のデータを処理しなければならない時
  - 合理的時間(多項式時間)内に完了できる

  計算
- 計算困難な計算
  - データ規模の増加に伴って
  - 多項式よりも速い速度で実行時間が増加してゆく

  計算

例(1)



- 名前，身長，体重が配列に格納されている
- 指定した身長と体重を持つ人の名前をすべて出力

	名前
1	John Jones
2	Sue Black
3	Bill Smith
4	Frank Doe
5	Jean White
6	Nancy Blike

	身長
1	67
2	67
3	73
4	68
5	67
6	71

	体重
1	120
2	131
3	166
4	140
5	131
6	162

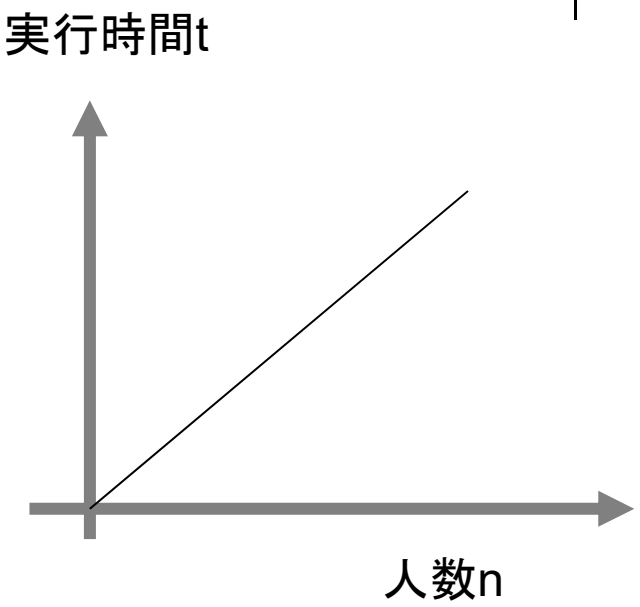
例(1)のアルゴリズム

1. データを配列に読込む(データの項目数=人数をnとする)
2. 目標とする身長と体重を入力
3. iを1からnまで変化させながら i～iiを繰り返す
   i. i番目の人の身長と体重を取出す
   ii. 身長と体重がともに目標の値と等しいならば，その人の名前を出力する

この部分の時間を考える

例(1)の実行時間

人数n	実行時間t(秒)
2500	1.275
5000	2.550
7500	3.825
10000	5.100



- 1万人の検索がわずか5秒ほどでできる
- グラフで表すと直線になる

$$t = 5.1 \times 10^{-4} \times n$$

日本国民すべてに対して検索すると

- n=1.2億

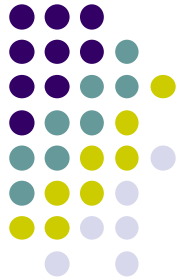
$$t = 5.1\times10^{-4}\times1.2\times10^{8} = 6.12\times10^{4}$$

- 6.12万秒=17時間

- 日本国民すべてに対し，1日もかからずに検索が済む

計算容易

例(2)



- 配列に格納された学生の点数を昇順に並べ替える=ソート
- クイックソートが平均的に速いアルゴリズム

人数n	実行時間t(秒)
2500	59.261
5000	129.021
7500	202.745
10000	279.042

$$t = 2.1 \times 10^{-3} \times n \times \log_2 n$$

日本国民すべてに対してソートすると

- n=1.2億

$$t = 2.1\times10^{-3}\times1.2\times10^{8}\times\log_2(1.2\times10^{8})$$

$$= 2.52\times10^{5}\times26.84 = 6.76\times10^{6}$$

- 676万秒=1879時間=78日=約2ヶ月

- かなり長い時間がかかるが，計算不可能というほどではない
- 国勢調査なら，大型の10倍も速いコンピュータを使うだろうから，約1週間で済む

計算容易

もう少し詳しい「計算容易」の定義

- 問題の規模をnとしたときに
- コンピュータ処理の時間を求める計算式が
- n(およびlogn)の多項式である

↓

計算容易な問題

- あるいは，多項式=polynomial なので

↓

P問題

計算容易な問題の計算時間の例

$$t = 3 \times n^2$$

$$t = 4 \times n^3 + 16 \times n^2 + 7 \times n + 8$$

$$t = 4 \times n^2 \times \log_2 n + 3 \times n$$

$$t = 17 \times (\log_2 n)^2 + n$$

例(3): ハノイの塔

- 前の例と同じように時間を計ってみる

円盤の数n	実行時間t(秒)
2500	?
5000	?
7500	?
10000	?

永遠に実行してるように見える

例(3): ハノイの塔 (続き)

- nを小さくして時間を計ってみる

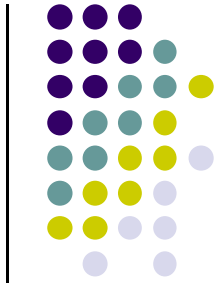
円盤の数n	実行時間t(秒)
1	1秒以下
2	1秒以下
3	1秒以下
:	
8	1
9	3
10	6

例(3): ハノイの塔 (続き)

- nをもうちょっと大きくして時間を計ってみる

円盤の数n	実行時間t(秒)
11	11
12	23
13	45
14	90
15	180
16	360

ハノイの塔の計算時間



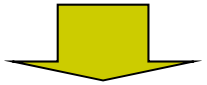
$$t = 5.49 \times 10^{-3} \times 2^{n}$$

nがここにある

nについての多項式ではない

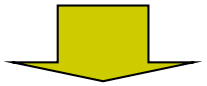
計算困難

- 問題の規模をnとしたときに
- コンピュータ処理の時間を求める計算式が
- n(およびlogn)の多項式でない
  - 一般にn乗の項を含む



計算困難な問題

- 多項式ではない=non polynomial なので



NP問題

ハノイの塔の実行時間

n	t(近似値)
5	0.17秒
10	5.62秒
15	3.00分
20	1.60時間
25	2.13日
30	68.23日
35	5.98年

n	t(近似値)
40	191.30年
45	6120.94年
50	195870年
55	6267840年
60	200571000年
65	6418270000年
70	205385000000年

超高速コンピュータなら解けるのでは?

- 1台のコンピュータを1000倍高速化する
- 問題をうまく分割して1000台のコンピュータで同時に処理する

↓

百万倍の高速化

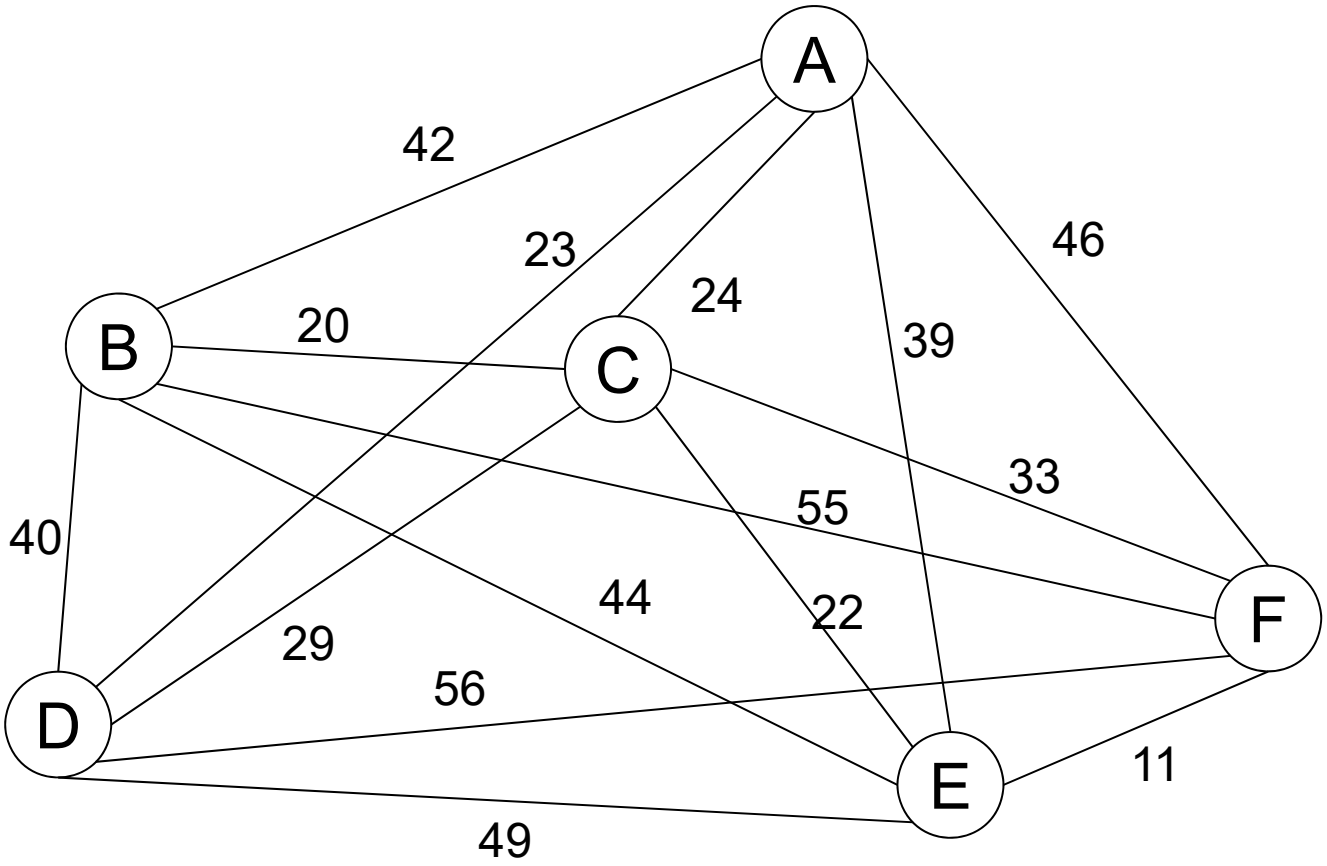
↓

いままでより円盤が20個多い場合を処理できるようになるだけ

解答がコスト高になる問題の実際例

- 最短距離の道筋の発見: 巡回セールスマン問題; Traveling Salesman Problem (TSP)
  - セールスマンが自分の住んでいる都市から出発
  - いくつかの都市をそれぞれ1回ずつ訪問
  - 最後に自分の都市に戻る
  - 最短距離となる巡回の仕方を求める

Traveling Salesman Problem



なぜTSPが重要か?

- TSPに帰着できる現実の問題が多くある
  - トラックの配送経路の制御
  - 送電システムの決定

- NP完全問題である
  - これが多項式時間で解けるなら，他のNP問題も多項式時間で解ける