コンピュータサイエンス第２

南出 靖彦

第 2 回

今日の内容

- アルゴリズムとは
- ローカル変数, グローバル変数
- 暗号：フォローアップ

準備: Windows

- コマンドプロンプトを実行，CS2 フォルダに移動.
  - `cd Documents`
  - `cd CS2`
- 講義のウェブページから `day2.zip` をダウンロードする.
  - Downloads（ダウンロード）フォルダに `day2.zip` を置かれる
  - ファイルを開く → すべてを展開
  - 展開先として，Documents¥CS2 を指定して，展開
- day2 フォルダに移動
  - `cd day2`

準備: Mac

- CS2 フォルダに移動.
  - `cd Documents`
  - `cd CS2`
- 講義のウェブページから `day2.zip` をダウンロードする.
- Terminal で day2 を CS2 に移動
  - `mv ~/Downloads/day2 ./`
  - `cd day2`

アルゴリズム (algorithm)

アルゴリズムとは

- 計算手順
- 目標を達成する計算方法
- 良いアルゴリズム $\Leftrightarrow$ 効率の良いアルゴリズム
  - 計算時間が短い

整数に関するアルゴリズム

- 指数関数
- 最小公倍数
- フィボナッチ数
- n 番目の素数
- ....

指数関数：二つのアルゴリズム

- 単純なアルゴリズム

$$n^k = n * n^{k-1}$$

$\Rightarrow$ k に比例した計算時間

- 工夫したアルゴリズム

$$\begin{aligned} n^{2k} &= (n^2)^k \\ n^{2k+1} &= n \times (n^2)^k \end{aligned}$$

$\Rightarrow$ $\log_2 k$ に比例した計算時間

注意：大きい数に対しては、本当は掛け算や割り算などの演算が定数時間ではできないので、上の計算時間は単純化しすぎ

- 単純なアルゴリズム

```
def exp1(n,k):
    r = 1
    for i in range(k):
        r = n * r
    return r
```

- 工夫したアルゴリズム

```
def exp2(n,k):
    if k == 0:
        return 1
    elif k % 2 == 0:
        return exp2(n * n, k // 2)
    else:
        return n * exp2(n * n, k // 2)
```

Python を対話的に使って試してみよう

```
$ python -i exp.py
>>> exp1(2,10)
1024
>>> exp2(2,10)
1024
>>>
```

第２引数が非常に大きいと実行時間の差がわかる

```
>>> exp1(2, 500000)
....
>>> exp2(2, 500000)
....
```

python の終わり方

```
>>> exit()
```

フィボナッチ数：計算時間

$fib(n)$: n 番目のフィボナッチ数

$$\begin{aligned} fib(0) &= 0 \\ fib(1) &= 1 \\ fib(n+2) &= fib(n+1) + fib(n) \quad (n \geq 0) \end{aligned}$$

$F(n)$: $fib(n)$ を実行した時に、関数 fib 何回呼ばれるか

$$\begin{aligned} F(0) &= 1 \\ F(1) &= 1 \\ F(n+2) &= F(n+1) + F(n) + 1 \quad (n \geq 0) \end{aligned}$$

展開して計算すると

$$\begin{aligned} F(n+2) &= F(n+1) + F(n) + 1 = (F(n) + F(n-1) + 1) + F(n) + 1 \\ &> 2 \times F(n) \end{aligned}$$

計算時間：$F(n) \geq 2^{\frac{n}{2}}$ ($n \geq 3$ の時)

- n が少し大きくなると計算時間が長すぎる

フィボナッチ数：計算時間

```
$ python -i fib.py
>>> fib(10)
55
>>> fib(20)
6765
>>> fib(30)
832040
>>> fib(50)
```

止まらないので、control + C で止めてください。

- Windows: control + break(pause) または Fn + control + break

注意：動的計画法の考え方を用いるアルゴリズムで，$fib(n)$ を n に比例する時間で計算できる

ローカル変数とグローバル変数

- ローカル変数：関数の仮引数，関数内で定義される変数
  - 関数（又はプログラム本体）の中でのみ有効
  - プログラム本体や他の関数内に同じ名前の変数があっても実体は別

- グローバル変数：関数の外で定義される変数
  - プログラム本体からも各関数内部からも触ることができる
  - プログラム実行中ずっと存在し続ける
  - 関数内でグローバル変数に代入する場合は，global 宣言を用いる

ローカル変数

- プログラム本体や他の関数内に同じ名前の変数があっても実体は別なので混同は起こらない。
- 同じ関数が何度も呼び出される場合にもその関数定義内のローカル変数は毎回名前は同じだが実体は別になる。

```
def test(a):
    a = 100
    b = 200
    print(a, b)

a = 1
b = 2
test(a)
print(a, b)
```

実行結果

```
100 200
1 2
```

- 関数 test 中の変数 a,b とプログラム本体中の変数 a,b は，別の変数として扱われる

グローバル変数

グローバル変数：関数外の変数，global 宣言された変数

- プログラム本体からも各関数内部からも触ることができる
- プログラム実行中ずっと存在し続ける

```
def test(a):
    global b
    a = 100
    b = 200
    print(a, b)
```

実行結果

```
100 200
1 200
```

```
a = 1
b = 2
test(a)
print(a, b)
```

- 関数 test の中のグローバル変数 b への代入が，プログラム本体へも影響する.

グローバル変数の応用: fib2.py

関数が何回呼び出されるか？

```
def fib(n):
    global calls
    calls = calls + 1
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2)+fib(n-1)

n = int(input())
calls = 0
print(fib(n))
print(calls)
```

```
$ python fib2.py
5
5
15
$ python fib2.py
10
55
177
$ python3 fib2.py
20
6765
21891
```