# データ構造とアルゴリズム (第3回)

http://coconut.sys.eng.shizuoka.ac.jp/algo/06/

静岡大学工学部
安藤和敏

2006.12.07

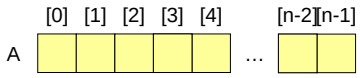
## 第2章アルゴリズムの基本データ構造

2.1 配列
2.2 連結リスト
2.3 スタック
2.4 キュー

## 第2章アルゴリズムの基本データ構造

2.1 配列
2.2 連結リスト
2.3 スタック
2.4 キュー

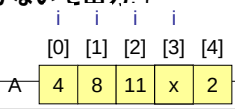
## 配列




## アルゴリズム2.1 (配列へのデータの追加)

**入力**:**サイズ** *n* **の配列** A **および追加するデータ** x
**アルゴリズム**:
i=0;
while ((A[i]**にデータが格納されている**)&&(i<n)){
  i=i+1;
}
if (i==n) { "**配列に格納場所がない**"**と出力**; }
else { A[i] =x; }

実行時間は O(n)




## 第2章アルゴリズムの基本データ構造

2.1 配列
2.2 連結リスト
2.3 スタック
2.4 キュー

# 第2章アルゴリズムの基本データ構造



## 複数の仕事やデータの処理の順番

あなたは図書館で勉強していた．そこへ，あなたの親友がやってきて恋愛相談に乗って欲しいと声をかけられた．返事をしようとしていたら，親から携帯に電話がかかってきた．

(A) 図書館で勉強する．
(B) 親友の恋愛相談に乗る．
(C) 携帯電話にでる．

(C) → (B) → (A) の順番で仕事をするのが普通．

## 複数の仕事やデータの処理の順番

あなたはファーストフード店で注文を受けるバイトをしている．レジの前には3人の客が並んでいる．あなたはこの3人の客から，
(A) チーズバーガーとコーラ
(B) てりやきバーガーとコーヒー
(C) フィッシュバーガーとコーヒー
という順番で注文を受けた．

(A) → (B) → (C) の順番で仕事をする．

## 複数の仕事やデータの処理の順番

(1) 処理要求の順番が遅いものから処理を済ませる．
(2) 処理要求の順番が早いものから処理を済ませる．

アルゴリズムの分野では，(1) の処理方法を LIFO (Last In First Out の略，ライフォ，リフォと読む)と呼び，(2) の処理をFIFO (First In First Out の略，ファイフォ，フィフォと読む)と呼ぶ．

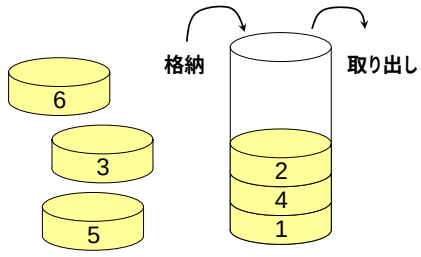
(1) のLIFO はスタック (stack) によって，(2) の FIFO はキュー (queue) によって実現される．

## スタックみたいなもの


スピンドルケース入りCD

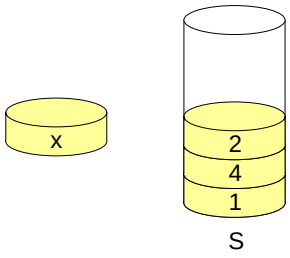
本の山

## スタックの概念図



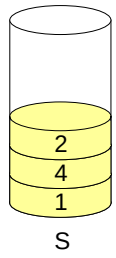
## スタックに対する操作 (push)

push(S,x): **スタック S にデータ x を格納する.**




## スタックに対する操作 (pop)

pop(S): **スタック S からデータの取り出しを行い, 取り出したデータを出力する.**




## アルゴリズム2.4 (関数 push)

```
入力:サイズ n の配列 S および追加するデータ x
push (S,x) {
  top = top + 1;
  if (top == n) {
    “オーバーフロー”と出力;
  } else {
    S[top] = x;
  }
}
```

O(1)時間で実行できる.

top=~~t~~2p==3

| | [0] | [1] | [2] | [3] | [4] |
|---|---|---|---|---|---|
| S | 1 | 4 | 2 | x | |

## アルゴリズム2.4 (関数 pop)

```
入力:サイズ n の配列 S
pop (S) {
  if (top == -1) {
    “アンダーフロー”と出力;
  } else {
    S[top] の値を出力;
    top = top -1;
  }
}
```

O(1)時間で実行できる.

S[top] ==3

top=~~t~~2p==3

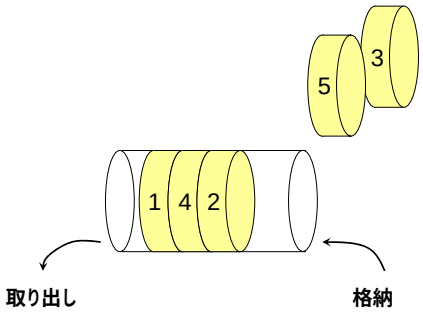
| | [0] | [1] | [2] | [3] | [4] |
|---|---|---|---|---|---|
| S | 1 | 4 | 2 | 3 | |

## 第2章アルゴリズムの基本データ構造



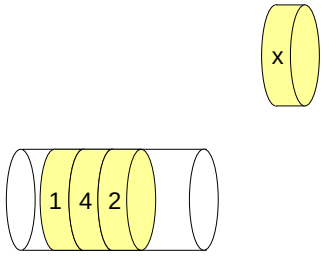
## キューみたいなもの

## キューの概念図



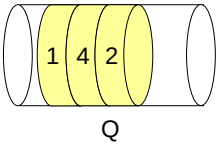

## キューに対する操作 (enqueue)

enqueue(Q,x): **キュー** Q **にデータ** x **を格納する.**




## キューに対する操作 (dequeue)

dequeue(Q): **キュー** Q **からデータの取り出しを行い, 取り出したデータを表示する.**




## アルゴリズム2.5 (関数enqueue)

```
入力:サイズ n の配列 Q および追加するデータ x
enqueue (Q,x) {
  Q[right] = x;
  right = right + 1;
  if (right == n) { right = 0; }
  if (left == right) { "オーバーフロー"と出力; }
}
```

O(1)時間で実行できる.

left==0 rightright==4

| | [0] | [1] | [2] | [3] | [4] |
|---|---|---|---|---|---|
| Q | 1 | 4 | 2 | x | |

## アルゴリズム2.5 (関数 dequeue)

```
入力:サイズ n の配列 Q
dequeue (Q) {
  if (left == right) {
    "アンダーフロー"と出力;
  } else {
    Q[left] の値を出力;
    left  = left + 1;
    if (left == n) left = 0;
  }
}
```

O(1)時間で実行できる.

Q[left] ==1

left left==1 right==4

| | [0] | [1] | [2] | [3] | [4] |
|---|---|---|---|---|---|
| Q | 1 | 4 | 2 | 3 | |

## 宿題

**第2章の演習問題の全て** (2.1**の**(2)**を除く**). (**提出しなくてもよい. 巻末の解答例を見て自己採点せよ.**)