アルゴリズムとデータ構造

第3週目

担当　情報システム部門　徳光政弘
2024年4月23日

今日の内容

- 配列
- 連結リスト
  - 単方向リスト
  - 双方向リスト（授業ではやらない、本当は大事）
- スタック
- キュー

配列へのデータの追加、削除(計算量)

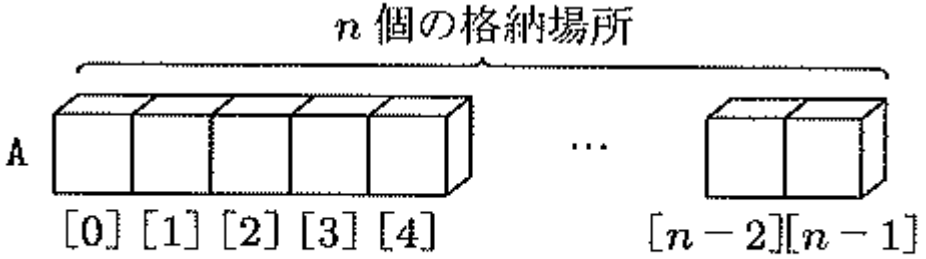

図 2.2　配列

A[2]=11の格納

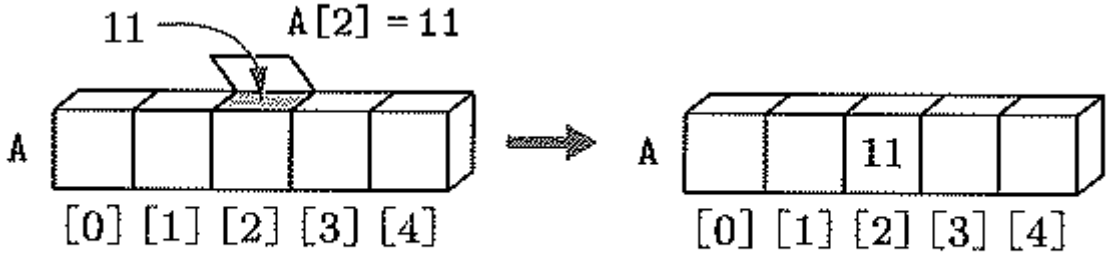

図 2.3　配列へのデータの格納

配列のデータ追加、削除の計算量

- 配列の中で空き場所を探す場合は、最悪時間計算量がO(n)となる。

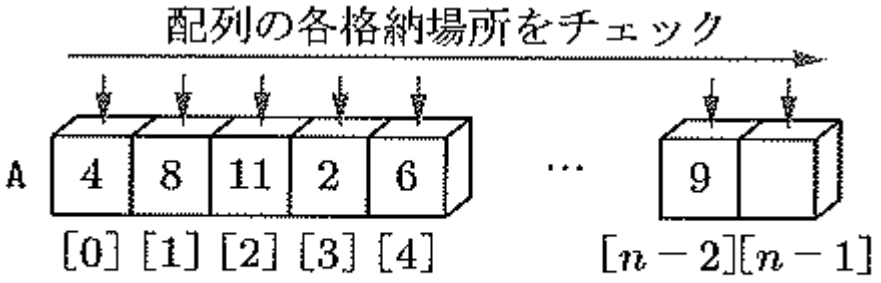

図 2.4　配列へのデータの追加

連結リスト

- リストの長さをデータの長さを大きくまたは小さくができる。リスト同士をつなぐこともできる。
- 例では数値を保持するリストになっている。

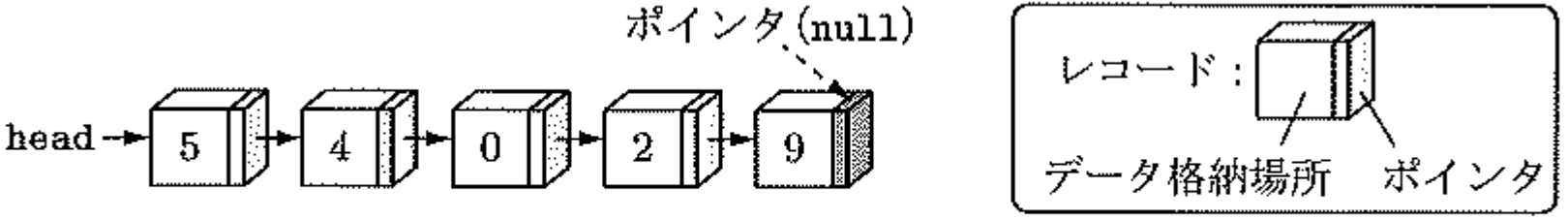

図 2.6　連結リスト

連結リストへのデータの追加

アルゴリズム 2.2　連結リストへのデータの追加

入力：連結リストの先頭を指すポインタhead，および追加するデータ x
新たなレコードR を準備する；
(レコードR のデータ格納場所)=x;
(レコードR のポインタ)=head;
head=(レコードR);

連結リストへのデータの追加

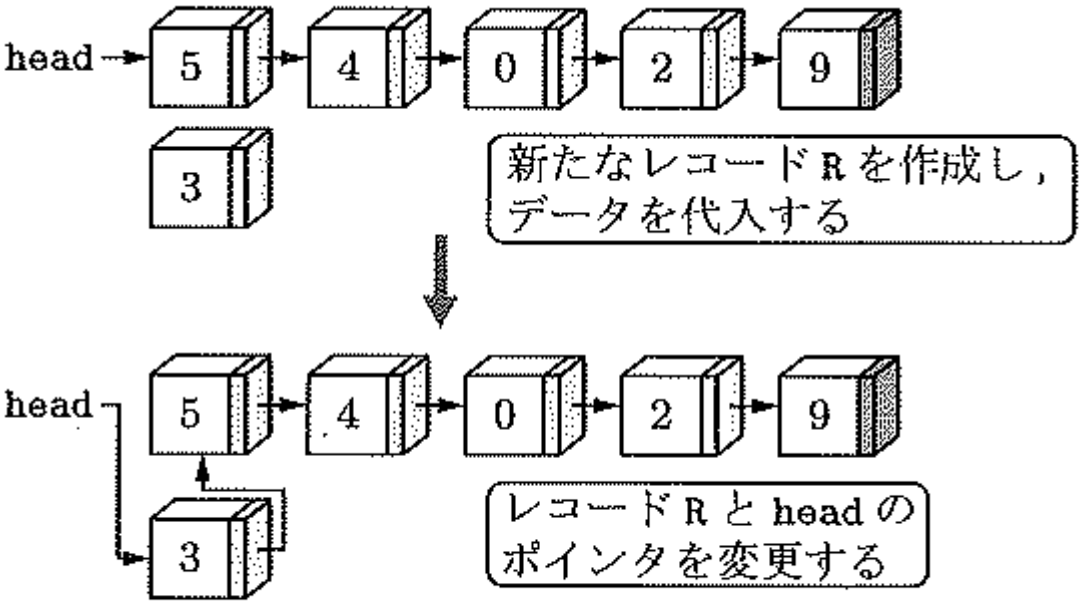

図 2.7　連結リストへの要素の追加

連結リストからデータの削除

アルゴリズム 2.3　連結リストからのデータの削除

```
入力：連結リストの先頭を指すポインタhead
if (head==null) { "削除するデータはない"と出力; }
else {
  head の指すレコードのデータを出力;
  head=(head の指すレコードのポインタが指すレコード);
  head の指していたレコードを削除;
}
```

連結リストからデータの削除

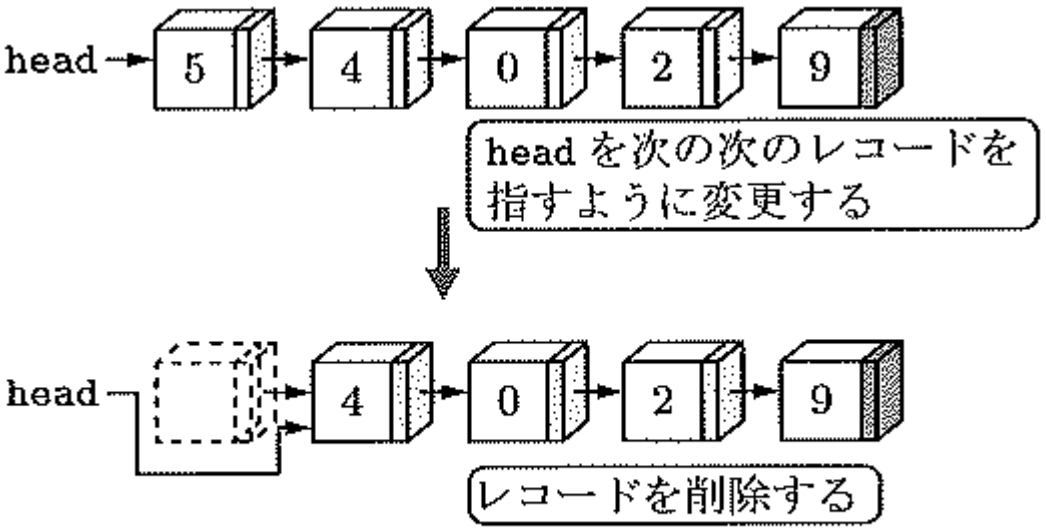

図 2.8　連結リストからのデータの削除

スタックとキュー

- 教科書の例に倣うならば
- スタック
  - 依頼すると一番最後
  - 処理要求の順番が遅いものから処理する
- キュー
  - 依頼すると一番最初
  - 処理要求の順番が早いものから処理する

スタック

push(S,x)：スタック S に対して，データ x を格納する．

pop(S)：スタック S からデータを取り出し，そのデータを出力する．

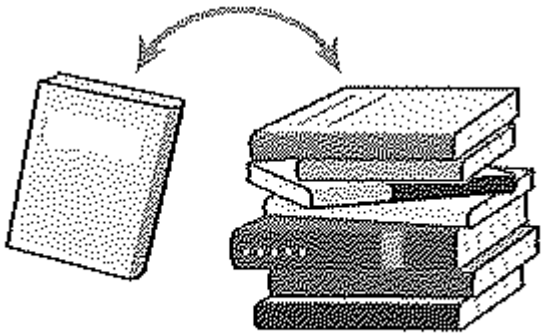

図 2.9　スタックの概念を表す例

スタックの概念図

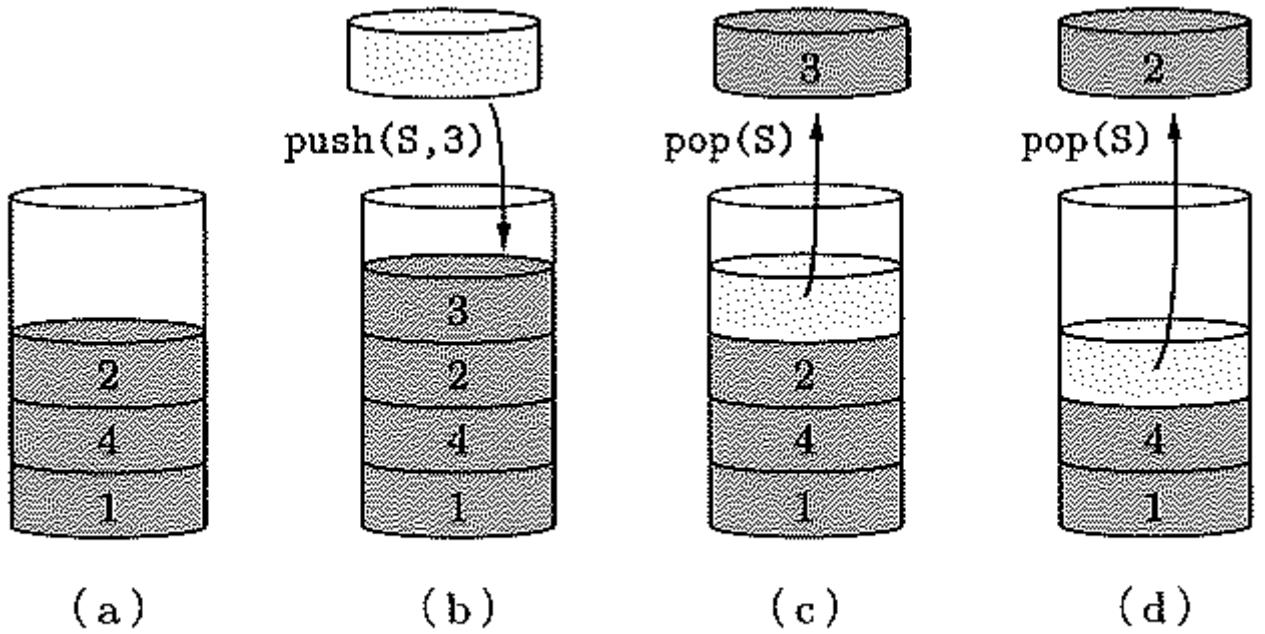

図 2.10　スタックの概念図

スタックを配列で実装

アルゴリズム 2.4　関数 push と関数 pop

関数push の入力：スタックを表すサイズ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; }
}
```

関数pop の入力：スタックを表すサイズ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; }
}
```

スタックを配列で実装

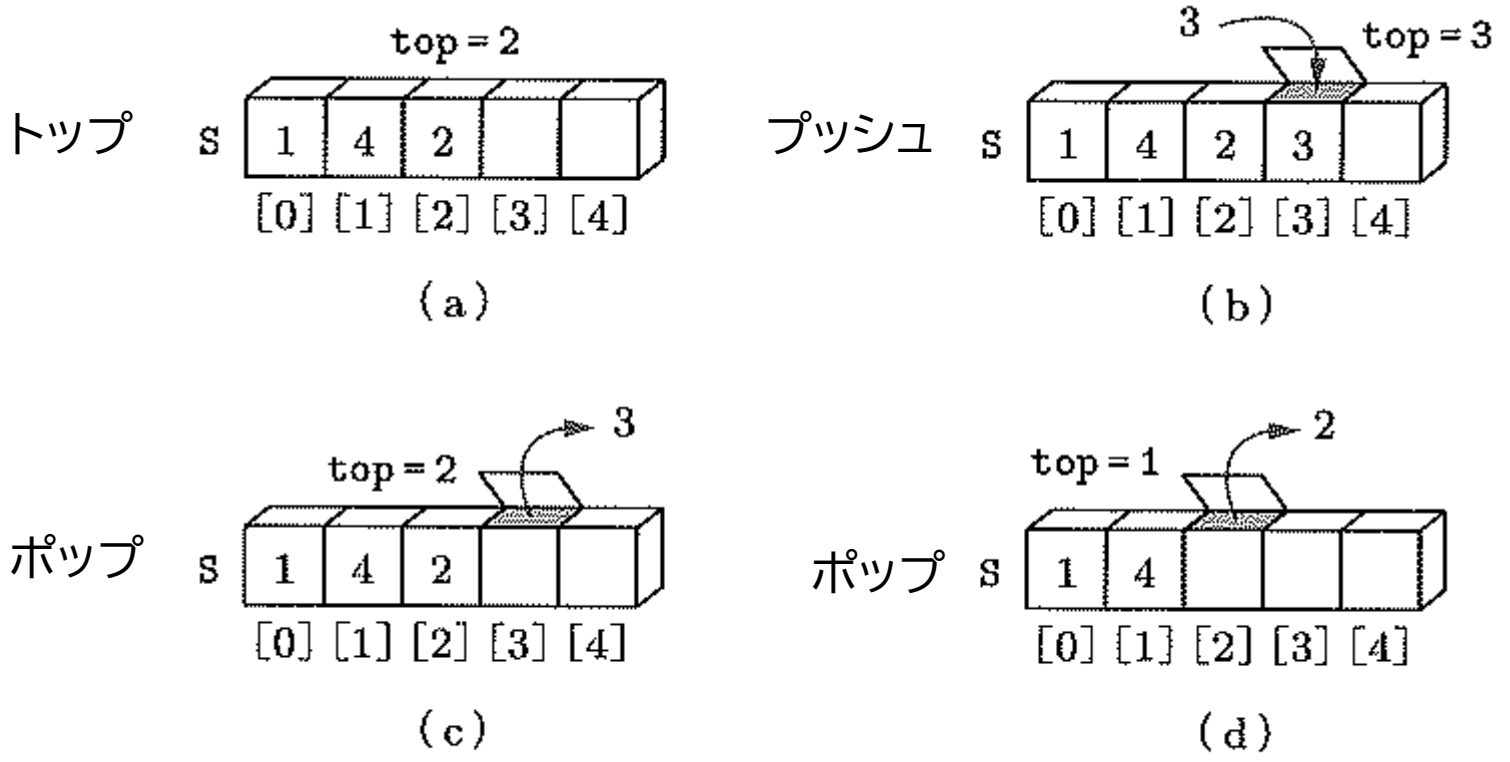

図 2.11　配列を用いたスタックの実現

キュー

- 順番待ちを再現

図 2.12　キューの概念を表す例

キューに対する操作

enqueue(Q,x)：キュー Q に対して，データ x を格納する．

dequeue(Q)：キュー Q からデータを取り出し，そのデータを出力する．

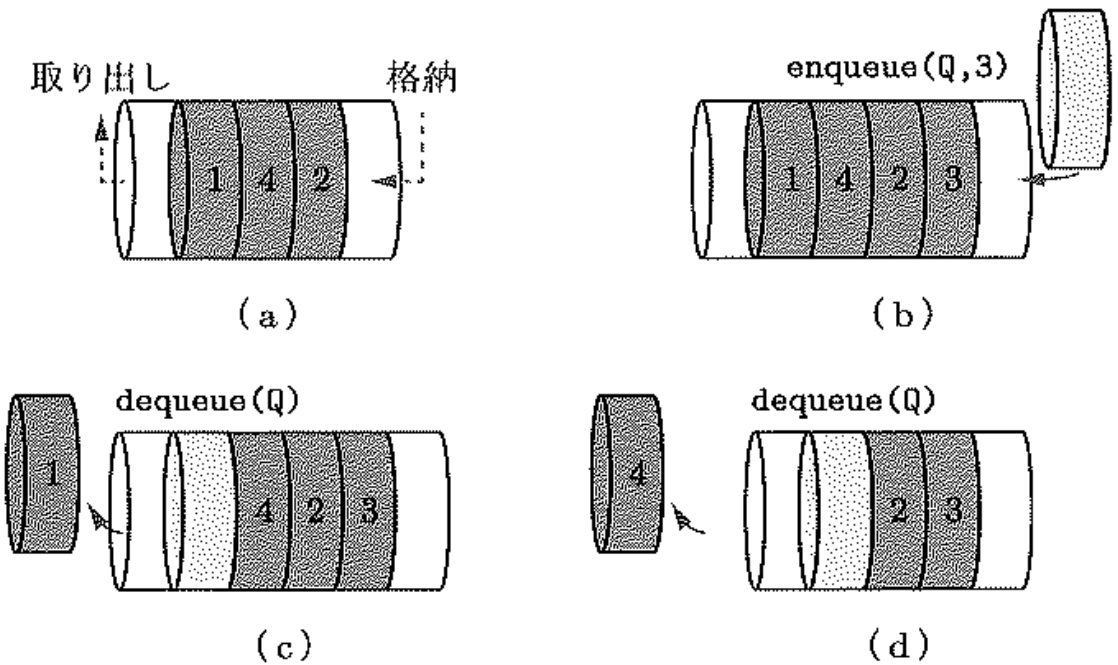

図 2.13　キューの概念図

キューに対する操作

アルゴリズム 2.5　関数 enqueue と関数 dequeue

関数enqueue の入力：キューを表すサイズ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;
```

キューに対する操作

```
    else {
      Q[left]の値を出力;
      left=left+1;
      if (left==n) left=0;
    }
  }
```

キューに対する操作

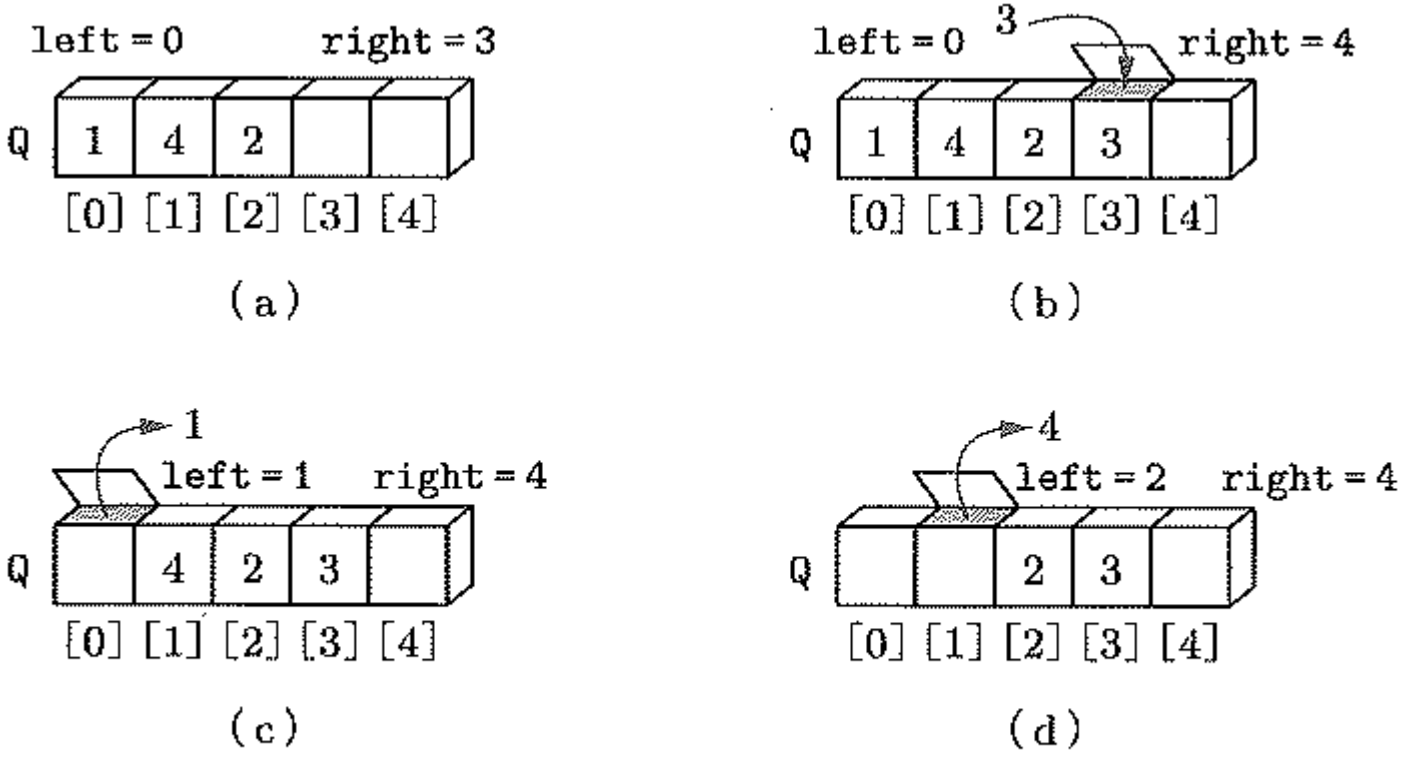

図 2.14　配列を用いたキューの実現