データ構造とアルゴリズム
（第3回）

モバイルコンピューティング研究室
柴田史久

1

1

本日の講義内容

- 基本的なデータ構造（２）
  - 待ち行列（キュー）
  - 配列による待ち行列の実現
  - 連結リスト

2

2

教科書 第4章（pp.85～86, pp.100～106）

基本的なデータ構造(2)
待ち行列

3

3

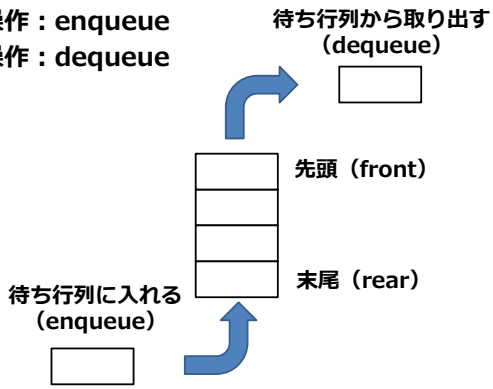
待ち行列(キュー;queue)

- 挿入が一方の端のみで行われ，
  削除が反対の端のみで行われるリスト
  - 新しい要素は最後の要素の次に挿入
  - 一番最初に挿入された要素が削除の対象
- FIFO（first-in first-out）
- 入力順に実行が必要な処理などに利用
- 先頭と末尾が同じとき待ち行列は空

4

待ち行列

- 挿入の操作：enqueue
- 削除の操作：dequeue



5

5

待ち行列の用語いろいろ

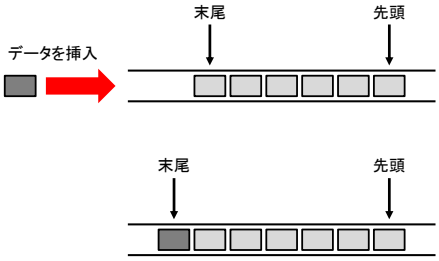
- 挿入
  - 挿入する（insert），置く（put），加える（add），
    加列する（enqueue）
- 取得
  - 取得する（get），削除する（remove），
    消す（delete），除列する（dequeue）
- 末尾
  - 末尾・後端（rear），しっぽ（tail），
    後ろ（back），終端（end）
- 先頭
  - 先頭（front），頭（head）

6

6

データの挿入

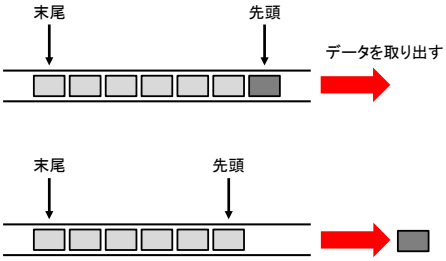
- データの挿入は必ず末尾の後ろ
  - いっぱいでないことを確認後，データを挿入し，末尾を移動



7

データの取り出し

- データの取り出しは必ず先頭から
  - 空でないことを確認後，データを取得し，先頭を移動



8

待ち行列の実現

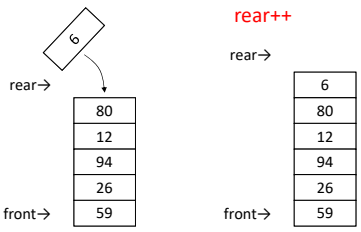
- 必要な機能
  - データを格納する領域
  - データを挿入，取り出しする機能
- どうやって実現する？
  - 配列を使ってみよう

9

9

配列による実現(挿入)

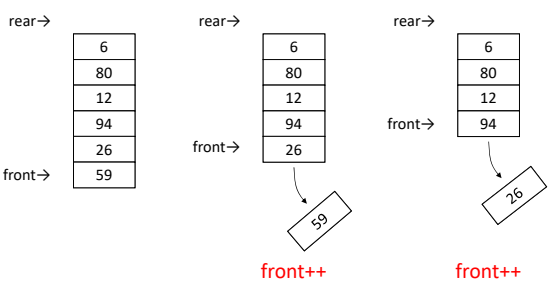
frontは先頭の要素を，rearは末尾の要素の次を指す



10

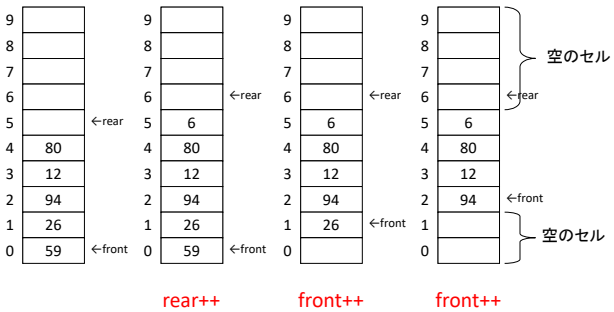
配列による実現(削除)

frontは先頭の要素を，rearは末尾の要素の次を指す



11

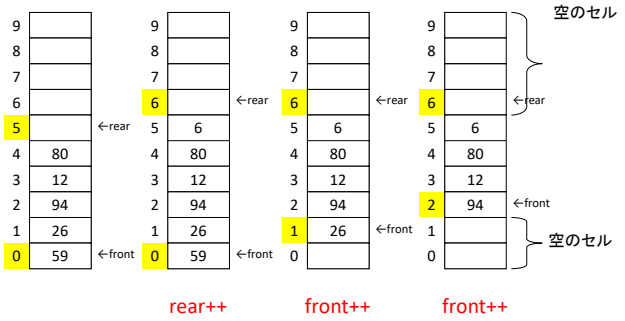
配列の添字に注目すると



12

配列を使う場合の問題点

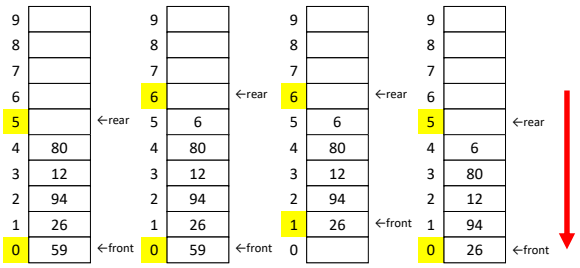
- １組の挿入と取得によりデータ列が１個分ずれる



13

データをずらしてみる

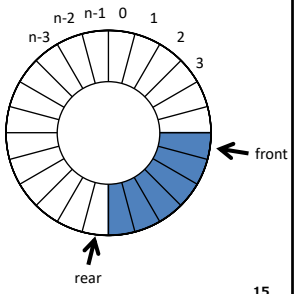
- データを１つ取得したら全体を１つずつ前にずらす
- データの移動は効率が悪い：$O(n)$



14

リングバッファ

- 配列をリング状にする
  - front=rearは空の状態
  - 最後の要素の次には最初の要素がある
  - rearは末尾の次の要素を指す
  - front，rearの増加時に配列の要素数で剰余（%）＝ラップアラウンド



15

15

空といっぱいの区別

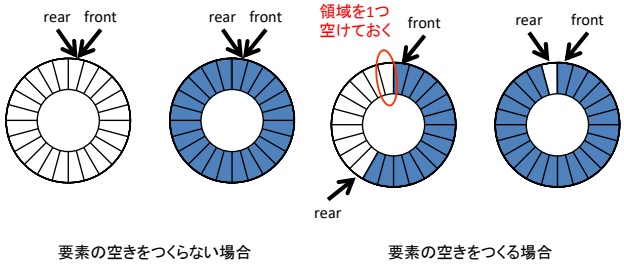
- いっぱいのときも front=rear となってしまう
- 解決策
  1. 待ち行列が空であることを示すフラグを準備
  2. 必ず1つ空の要素を残す
  3. 待ち行列に入っている要素の個数を記録
- 教科書／演習課題では 2. で解決

16

16

データの最大数の扱い

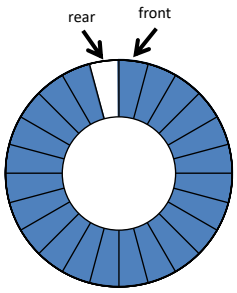
- データの最大数＋１の領域を確保する理由は？
  - データが空のときといっぱいのときを簡単に区別するため



17

いっぱいかどうかの判定（isFullメソッド）

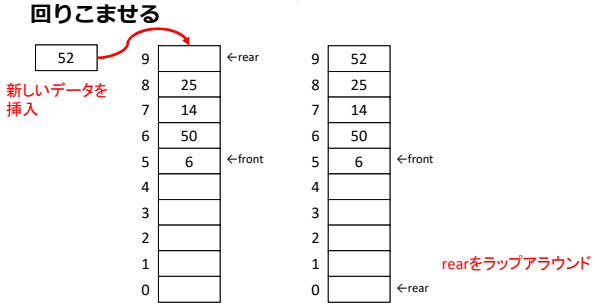
- rearの１つ次にfrontがあればいっぱい
  - ラップアラウンドの処理に注意



18

データの挿入(enqueueメソッド)

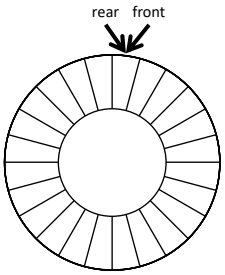
- 待ち行列のデータはリアに追加
  - 待ち行列がいっぱいならエラー（isFullメソッドを利用）
  - rearが配列の終端の場合は，データの追加後に逆端に回りこませる



19

空かどうかの判定(isEmptyメソッド)

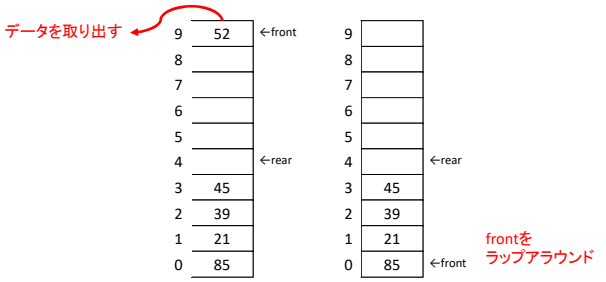
- frontとrearが同じ場所にあれば空
  - 必ず１つはあまらせるようにしているため，いっぱいの状態でfrontとrearが同じ場所になることはない



20

データの取得(dequeueメソッド)

- データの取得はフロント側から行う
  - 待ち行列が空ならばエラー（isEmptyメソッドを利用）
  - frontが配列の終端の場合は，データの取得後に逆端に回りこませる



21

教科書 第4章（pp.107～135）

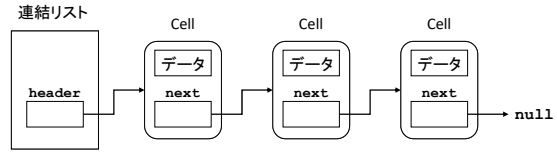
基本的なデータ構造(2)
連結リスト

22

22

連結リスト(linked list)

- データをそれぞれの要素に格納し，その要素が次の要素とつながってリストを構成するデータ構造
- 最も単純なものは下図のような単方向連結リスト




```
class Cell {
  Cell   next ;  // 次の要素へのリンク
  MyData value ; // この要素の値
}
```

自己参照型クラス定義

23

23

連結リストの基本的性質

	配列	連結リスト
任意の要素の参照	ランダムアクセス：$O(1)$	シーケンシャルアクセス：$O(n)$
特定の要素	$O(1)$	$O(1)$
リンクのメモリ	不要	要
挿入・削除	$O(n)$	$O(1)$
メモリ領域	事前に決定	自由に増減可能
連結・分割	$O(n)$	$O(1)$

24

24

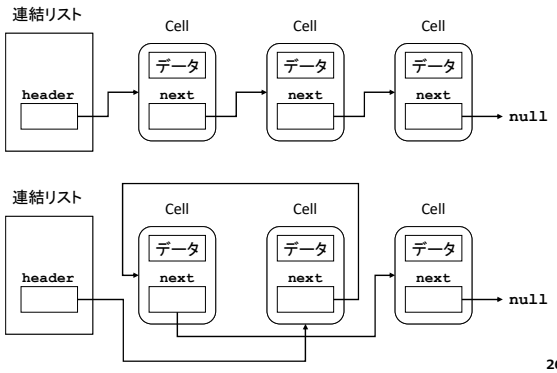
ちょっと一言

- 教科書では個々の要素を Cell というクラスで定義
- 他の本では要素（Element）やノード（Node），リンク（Link）と呼ぶことがある
- 講義は混乱しないように教科書の呼び方に統一

25

25

要素の順番変更が自由



26

26

まずは簡単な実装から

- リストの先頭に項目を挿入する
  - insertFirst
- リストの先頭の項目を削除する
  - deleteFirst
- リストを走査してその内容を表示する
  - displayList
- ダミーセルを使わない方法で説明

27

27

Cellクラス

- Cellクラスのオブジェクトは個々のデータを格納
- データは複数個がまとまっていてもよい
  - 例：携帯電話のアドレス帳
    - 氏名
    - 読み仮名
    - 電話番号
    - メールアドレス
    - etc

28

28

Cellクラスの雛形

```
public class Cell {

  Cell    next ;
  MyData  data ;

  public Cell(MyData d) {
    this.data = d ;
    this.next = null ;
  }

}
```

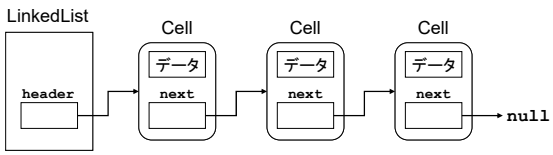
29

29

LinkedListクラス

- LinkedListクラスのオブジェクトはリスト自体
- データが増えれば伸びていく

LinkedListクラスは第5週の演習ではAddressクラスに相当



30

30

LinkedListクラスの雛形

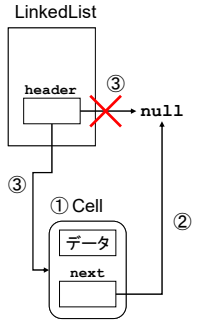
```
public class LinkedList {
  private Cell header ;    // リストの最初のリンクを指す

  public LinkedList() {    // コンストラクタ
    header = null ;        // 初期状態ではリストにCellはない
  }
  public boolean isEmpty() {  // リストが空かどうかを判定
    return (header == null) ;
  }

  …                          // その他のメソッド

}
```

31

31

リスト先頭にデータを挿入

- insertFirstメソッド



① 新しいCellオブジェクトを作成
② 新しいCellオブジェクトのnextを
元のheaderが指していたものに
③ headerが指す先を
新しいCellオブジェクトに

32

32

insertFristメソッドの実装例

```
public class LinkedList {
  …                              // フィールド，その他のメソッド

  public void insertFirst(Mydata d) {

    Cell p = new Cell(d) ;   // 新たなCellを作る
    p.next = header ;        // p の次は元の header
    header = p ;             // header は p

  }

  …                          // その他のメソッド

}
```

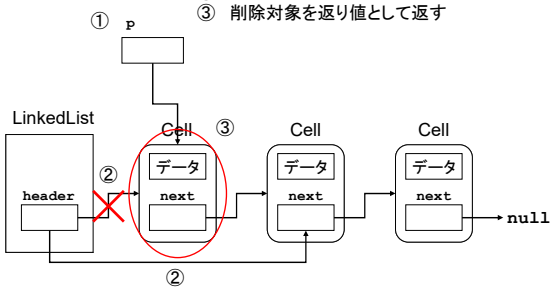
33

33

リスト先頭からデータを削除

- deleteFirstメソッド

① 削除対象をpが指すようにする
② headerが指す先をheaderのnextが指す先にする
③ 削除対象を返り値として返す



34

34

deleteFirstメソッドの実装例

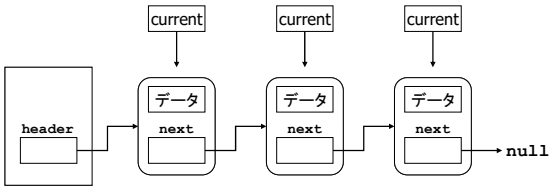
```
public class LinkedList {

  …                              // フィールド，その他のメソッド

  public MyData deleteFirst() {

    // 要素が存在しない場合(=headerがnull)の処理

    Cell p = header;           // 削除対象を p に保存
    header = header.next ;     // 削除:headerはheaderの次
    return (p.data) ;          // 削除対象のデータを返す

  }

}
```

35

35

データの表示 —displayListメソッド—

- 先頭の要素から順番にデータを表示
- 要素数を n とすると，計算量は $O(n)$



36

36

displayListメソッドの実装例

```
public class LinkedList {
  …                                  // フィールド，その他のメソッド

  public void displayList() {

    Cell current = header;       // リストの先頭からスタート
    while (current != null) {    // リストの末尾まで

      // データを表示

      current = current.next ; // 次のリンクへ進む
    }

}

  …                                  // その他のメソッド
}
```

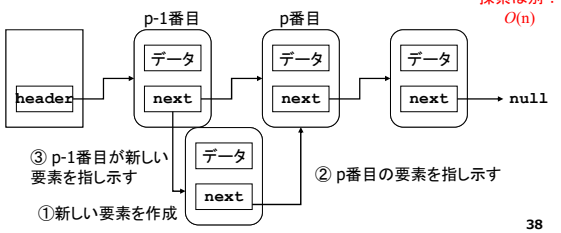
37

37

任意の位置へのデータの挿入

- p番目にデータを挿入したい
  - ① データを格納する新しい要素を作成
  - ② p番目の要素を指し示すように設定
  - ③ p-1番目の要素が新しい要素を指し示すように設定
- 変更は全体に影響しないので計算量は $O(1)$

探索は別！ $O(n)$



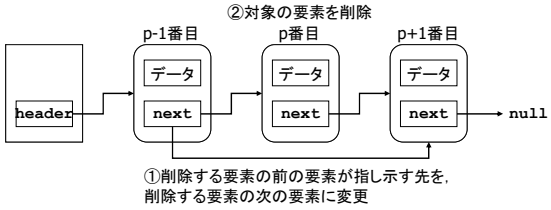
38

38

任意の位置からのデータの削除

- 削除対象を検索→p番目のデータが削除対象
  - ① p-1番目の要素が指し示す先を，p+1番目に変更
  - ② 要素ごと削除
  - ③ 消した要素を返り値として返す
- 変更は全体に影響しないので計算量は $O(1)$

探索は別！ $O(n)$



39

39

まずは簡単な実装から

- リストの先頭に項目を挿入する
  - insertFirst
- リストの先頭の項目を削除する
  - deleteFirst
- リストを走査してその内容を表示する
  - displayList
- ダミーセルを使わない方法で説明

40

40

本格的な連結リストへ拡張

- リストの先頭に項目を挿入する
- リストの先頭の項目を削除する
- リストを走査してその内容を表示する
- 任意の位置の要素を取得する
  - getData
- 任意の位置に要素を挿入する
  - insert
- 任意の位置の要素を削除する
  - delete

41

41

サイズへの対応,他(1)

```
public class LinkedList {

  private Cell header ;    // リストの最初のリンクを指す
  private int size ;       // リストのサイズを覚えておく

  public LinkedList() {    // コンストラクタ
    header = null ;        // リストに要素(Cell)はない
    size = 0 ;             // サイズを初期化
  }
  public boolean isEmpty() {  // リストが空かどうかを判定
    return (header == null) ;
  }

  …                          // その他のメソッド

}
```

リストのサイズを覚えておくと後の処理で便利

42

42

サイズへの対応,他(2)

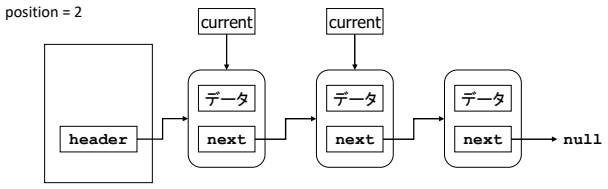
```
public class LinkedList {
  public void insertFirst(Mydata d) {
    Cell p = new Cell(d) ;  // 新たなCellを作る
    p.next = header ;       // p の次は元の header
    header = p ;            // header は p
    size++ ;                // サイズを増やす
  }

  public MyData deleteFirst() {
    if (header != null) {     // 削除対象の有無を確認
      Cell p = header;        // 削除対象を p に保存
      header = header.next ; // 削除:headerはheaderの次
      size-- ;                // サイズを減らす
      return (p.data) ;       // 削除対象のデータを返す
    }
    return (null) ;           // 削除対象がない場合
}
```

43

43

任意の位置の要素の取得

- position番目を探したい
  - ① 指定された位置がリストの範囲内かを確認
  - ② 先頭から順に指定された位置までリストをたどる



注意:positionは先頭を1番目として数える. 配列と異なるので注意

44

44

任意の位置の要素の取得

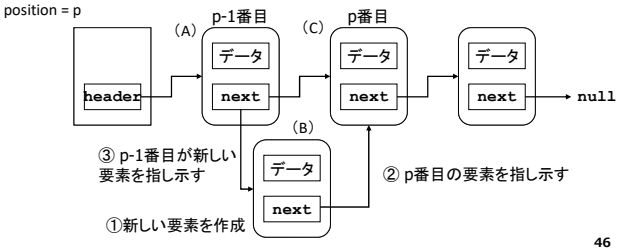
```
public MyData getMyData(int position) {

  // position がリストの範囲内かどうかを判定(sizeを利用)
  if ((position <= 0) || (position > size)) {
    return (null) ;
  }

  Cell current = header ;  // current が先頭を指すように

  // position まで要素を順番に移動する
  for (int count = 1 ; count < position ; count++) {
    current = current.next ;
  }

  return (current.data) ;
}
```

45

45

任意の位置への要素の挿入

- p番目にデータを挿入したい
  - ① データを格納する新しい要素（Cell）を作成
  - ② p番目の要素を指し示すように設定
  - ③ p-1番目の要素が新しい要素を指し示すように設定



46

46

任意の位置への要素の挿入

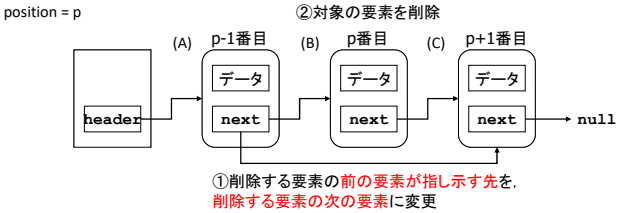
- 先頭（position == 1）にデータを挿入
  - insertFirstメソッドを利用すればよい
- 先頭以外にデータを挿入するには
  1. 挿入する場所の１つ前の要素（A）を見つける
     - １つ前の要素が存在しないならデータ挿入はできない
  2. 引数に与えられたデータを格納した新たな要素（B）を作成
  3. 新たな要素（B）の次の要素を１つ前の要素（A）の次の要素（C）に変更
  4. １つ前の要素（A）の次の要素を新たな要素（B）に変更
  5. サイズを１増やす

47

47

任意の位置からの要素の削除

- 削除対象を検索→p番目のデータが削除対象
  - ① p-1番目の要素が指し示す先を，p+1番目に変更
  - ② 要素ごと削除



48

48

任意の位置からの要素の削除

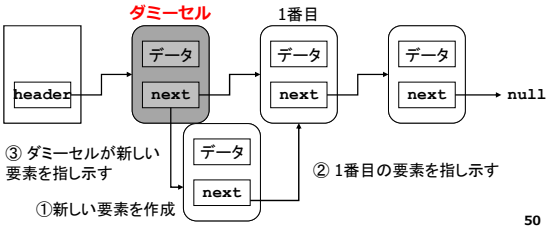
- 先頭（position == 1）からデータを削除
  - deleteFirstメソッドを利用すればよい
- 先頭以外からデータを削除するには
  1. 削除対象の１つ前の要素（A）を取得
     - １つ前の要素が存在しないならデータは削除できない
  2. 削除対象の要素（B）を取得
     - 削除対象の要素が存在しないならデータは削除できない
  3. １つ前の要素（A）の次の要素を削除対象の要素（B）の次の要素（C）に変更
  4. 削除対象の要素（B）の次の要素をnullに変更
  5. サイズを１減らす

49

49

ダミーセルの利用（挿入）

- 先頭にダミーセルを設けると場合分けが不要に
- 1番目にデータを挿入したい
  - ① データを格納する新しい要素を作成
  - ② 1番目の要素を指し示すように設定
  - ③ ダミーセルの要素が新しい要素を指し示すように設定

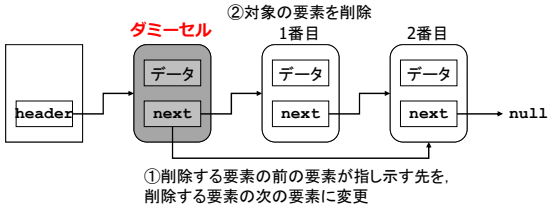


50

50

ダミーセルの利用（削除）

- 先頭にダミーセルを設けると場合分けが不要に
- 1番目のデータを削除したい
  - ① ダミーセルの要素が指し示す先を，2番目に変更
  - ② 1番目の要素を削除
  - ③ 消した要素を返り値として返す



51

51

連結リストとスタック

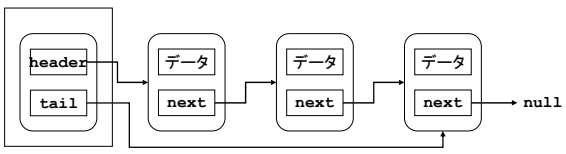
- スタックは連結リストで実現可能
  - プッシュ：リストの先頭にデータを挿入
  - ポップ：リストの先頭からデータを取得・削除
  - insertFirst，deleteFirstで実現可能！

52

52

連結リストと待ち行列

- 待ち行列は連結リストで実現可能
  - enqueue：リストの先頭にデータを挿入
  - dequeue：リストの末尾からデータを取得
    - リストの末尾を指す tailを準備しておくと便利



53

53

イテレータ

- クラスの実装に関する知識を使わずに，各要素に対する繰り返し処理を抽象化したもの
  - hasNextメソッドで次の要素の有無を確認
  - Nextメソッドで次の要素を取得
- 詳細は教科書 5.1.4 節を確認

```
MyLinkedList list = new MyLinkedList() ;

// ここで連結リスト list にデータをいれる

Iterator iter = list.iterator() ;  // list のイテレータを取得
while (iter.hasNext()) {
  System.out.println(iter.next()) ;
}
```

54

54

まとめ

- 待ち行列（キュー）
  - 配列による待ち行列の実現
- 連結リスト

55

55

参考文献

- 定本 Javaプログラマのための
  アルゴリズムとデータ構造（近藤嘉雪）
- 新・明解 Javaで学ぶ
  アルゴリズムとデータ構造（柴田望洋）
- 岩波講座ソフトウェア科学 3
  アルゴリズムとデータ構造（石畑清）
- Javaで学ぶアルゴリズムとデータ構造
  Robert Lafore（著）・岩谷 宏（翻訳）
- Java アルゴリズム+データ構造完全制覇
  オングス (著)・杉山 貴章・後藤 大地 (監修)

56

56