アルゴリズムとデータ構造

第10回　連結リストの応用

第10回のキーワード

アルゴリズム関係

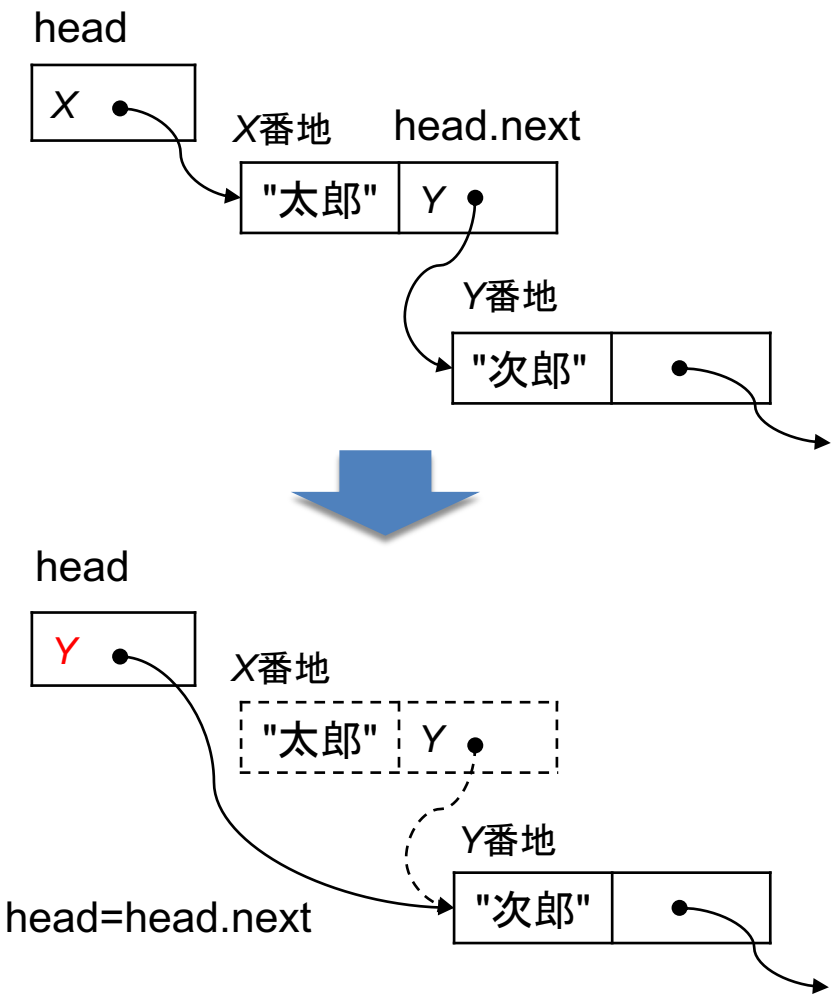
- 連結リストへの挿入
- 連結リストからの削除
- 双方向連結リスト
  （doubly linked list）
- 循環リスト
  （circular linked list）
- $O(1)$
- 連結リストの探索と整列

Java関係

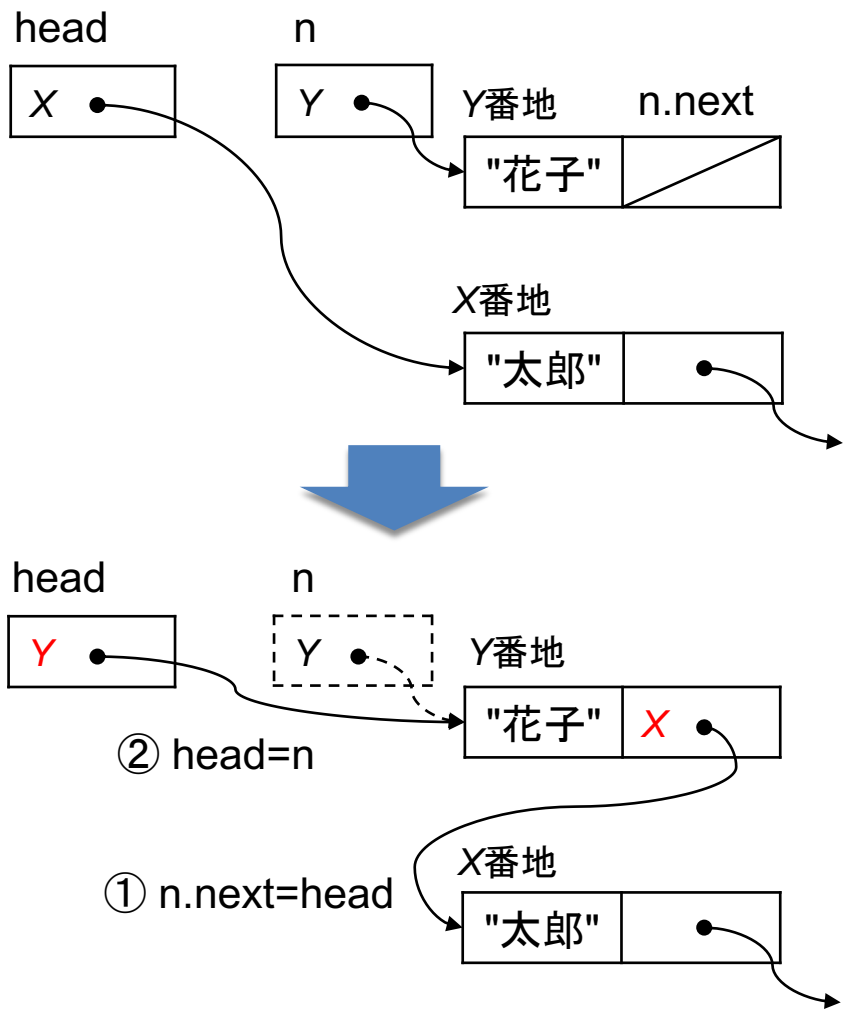
- ジェネリクス / 総称型
  （generics）
- ジェネリッククラスの定義
  class Node<E> { … }
  class List<E> { … }
- java.util.LinkedList<E>

リスト先頭での削除と挿入

- 先頭ノードの削除（pop）

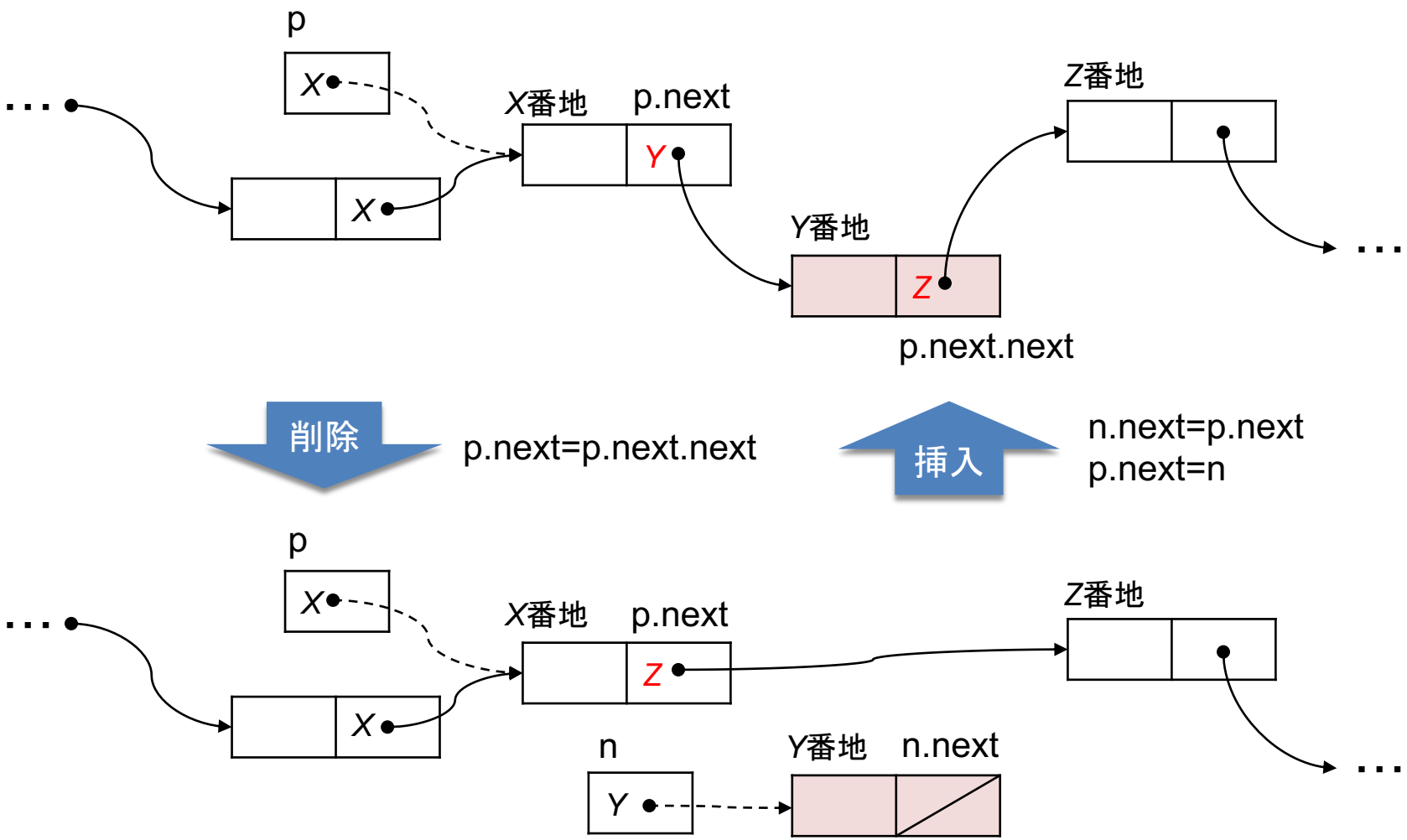


- 先頭ノードの挿入（push）



リスト途中での削除と挿入

- リストの途中（や末尾）でノードを削除・挿入する



確認問題

- 挿入と削除の実装
  - 下記は，連結リストにおけるノードの削除と挿入の実装例である。空欄を埋めてメソッドを完成させよ。

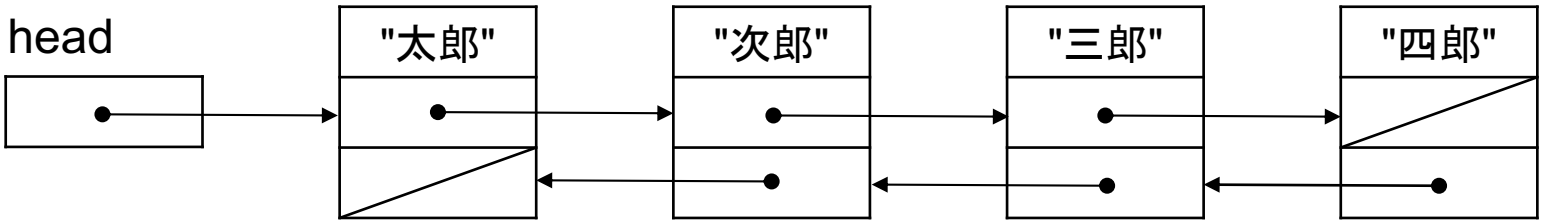
```
// ノードpの次のノードを削除
public void
removeNext(Node p) {
  if (p == null) {
    // pがnullのときは先頭を削除
    if (head != null) {
      head =
    }
  } else if (p.next != null) {
    // そうでないときは通常の削除
    p.next =
  }
}
```

```
// ノードpの次に新しいノードnを挿入
public void
insertNext(Node p, Node n) {
  if (p == null) {
    // pがnullのときは先頭に挿入
    n.next =
    head =

  } else {
    // そうでないときは通常の挿入
    n.next =
    p.next =
  }
}
```

双方向連結リスト

- 前後のノードへのリンクを保持
  - 前から後ろだけでなく，後ろから前にもたどれる
  - 現在指しているノードを削除することができる



- ノードの構造


```
class Node {
    String data;
    Node next;   // 次へのリンク
    Node prev;   // 前へのリンク
}
```

確認問題

- 双方向連結リスト
  - 双方向連結リストにおける削除と挿入の手順を図示せよ。
  - 下記はそれら実装例である。空欄入る処理を考えよ。

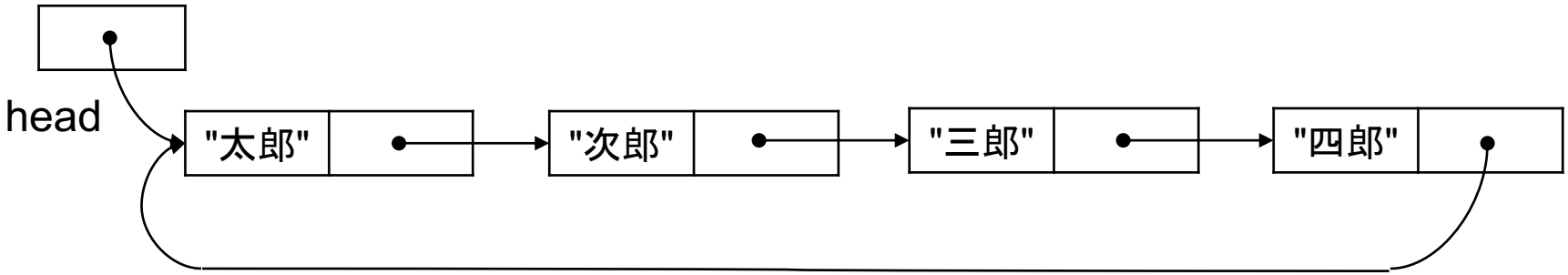
```
// リストの中にあるノードnを削除
public void remove(Node n) {
  if (n == head) {
    // 先頭を削除する場合はheadを調整
    head = n.next;
  }
  if (n.prev != null) {
                    = n.next;
  }
  if (n.next != null) {
                    = n.prev;
  }
}
```

双方向連結リストの細かい実装は
発展課題だが，要点は理解すること

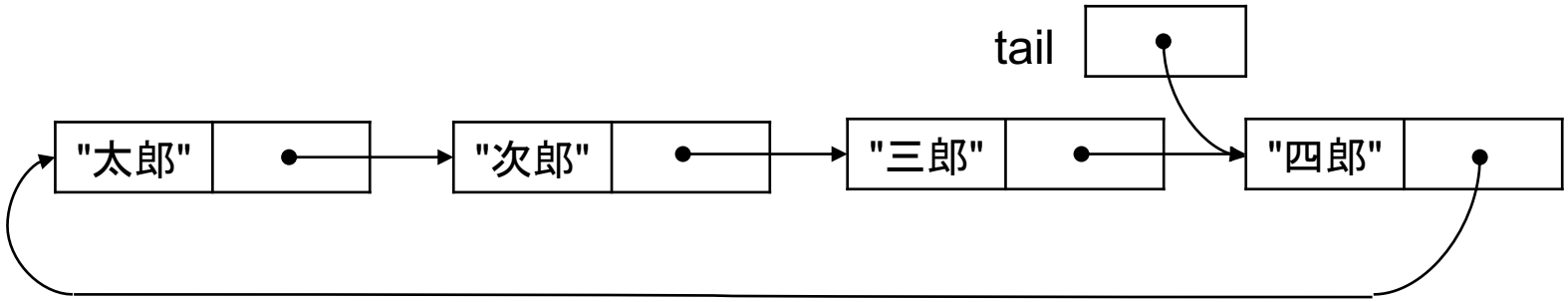
```
// ノードpの次に新しいノードnを挿入
public void insert(Node p, Node n) {
  if (p != null) {
    n.prev =
    n.next =
    p.next =
  } else {
    // pがnullなら先頭に挿入
    n.prev = null;
    n.next = head;
    head = n;
  }
  if (n.next != null) {
    n.next.prev =
  }
}
```

循環リスト

- 末尾から先頭につなげる
  - リストを「回転」し，順に繰り返して処理することが容易



- 効率的なキューの実装に利用できる
  - 末尾（tail）だけ分かれば，先頭はその次で求められる



連結リストの探索と整列

- 配列 vs 連結リスト
  - 配列：ランダムアクセス（途中の要素を直接参照可能）
  - 連結リスト：シーケンシャルアクセス（順にたどるしかない）
- 連結リストの探索
  - 基本的には，先頭から順にたどって線形探索する
  - 高速化したい場合は，リストではなく木構造などを用いる
- 連結リストの整列
  - 要素の交換ではなく，ノードの移動を使うように工夫する
  - 挿入ソートは，要素をずらす処理をせずに実現できる
  - マージソートは，一時的な領域を確保せずに実現できる

Javaジェネリクス

- ジェネリック（総称的）プログラミング
  - アルゴリズムを特定のデータ型（クラス）に依存しないようにプログラムすることで，再利用可能にする
  - 再利用時には，パラメータとしてデータ型を当てはめると，指定されたデータ型に対応した処理が実現される

- Javaジェネリクス
  - クラスまたはメソッドの定義で，「<仮クラス名>」という書式によって利用するクラスをパラメータ化できる
  - 定義の中では，仮クラス名（総称型という）を使っておく
  - それを使用する処理では，実際のクラス名を当てはめる
  - 例：class Node<E> { ... } → new Node<String>();