アルゴリズムとデータ構造

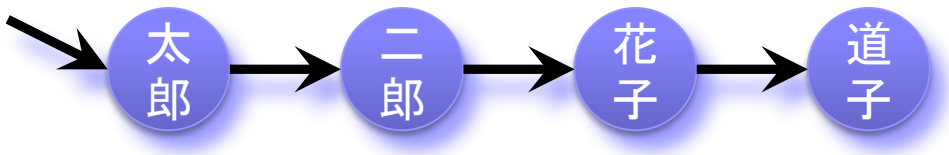
第３回補足　連結リスト

（基本的なデータ構造（リスト、スタック、キュー））

今日の内容：連結リスト

- 連結リスト（一方向リスト）の実装の詳細を学ぶ
  - 通常のheadが要素セルの列を指すやりかただと，先頭セルかその他の場合かで，挿入と削除に注意が必要．
  - 実際に使われている実装法
- 工夫：先頭にダミーセルを入れる
  - 各種の挿入・削除演算が統一的にかける
  - セルの位置＝そのセルの一つ前のセルのポインタをもつ
  - 先頭への挿入(addFirst)＝ダミーセルの次に新セルを挿入する
  - 挿入(insert) ＝セルポインタの次に新セルを挿入する
  - 削除(delete) ＝セルポインタの次のセルを削除する
- 注意：双方向リストの場合は，ダミーセルは不要

リスト



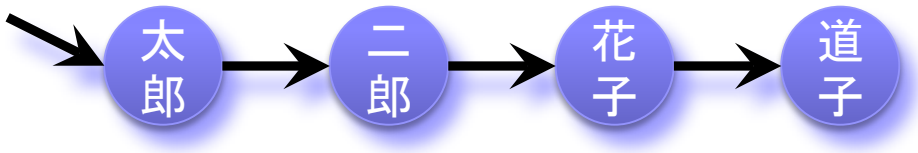
リストとは

- 要素を0個以上1列に並べたもの
- 要素の（途中への）追加，削除，検索等の演算をもつ
- いろいろな実装法がある

リストに対する操作

- INSERT(x,p,L)：リストL（要素数n）の位置pの次の位置に要素xを**挿入**
- DELETE(p,L)：リストLの位置pの次の位置の次の要素を**削除**
- FIND(i,L)：リストLのi番目のセルの内容を返す
- LAST(L) ：リストLの最後のセルの位置を返す
- PREVIOUS(p,L)：リストLにおいて、位置pの1つ前のセルの位置を返す

リスト



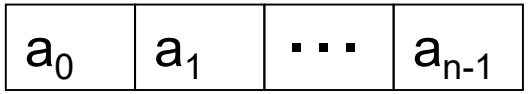
リストとは

要素を0個以上1列に並べたもの

（注意）リストは連結リストを指すことが多い

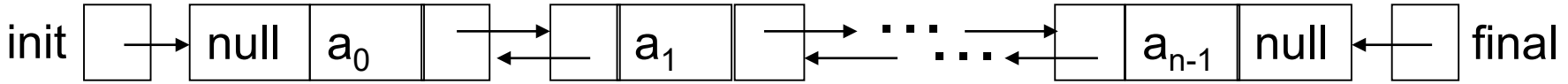
[リスト $a_0,a_1,\ldots,a_{n-1}$の実現法]

1. 配列(array)　　n個の連続領域に格納



2. 連結リスト(linked list)　　ポインタで次の要素の格納領域を指す

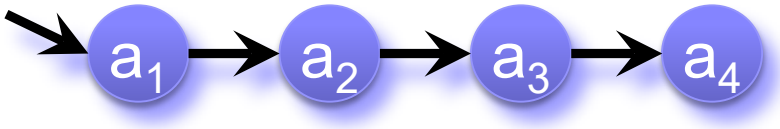
3. 双方向連結リスト(doubly linked list)　　ポインタで前後の要素の格納領域を指す



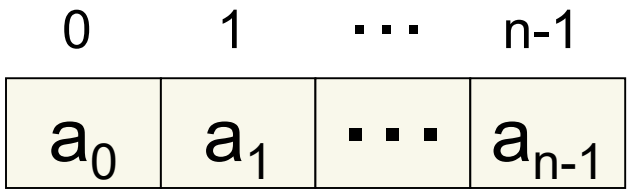
リスト:

- 配列(array)による実装

n=4



1. 配列(array)



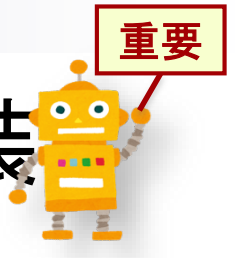
n個の連続領域に格納

コード例:% 100個の整数a[0],a[1],…,a[99]からなる配列

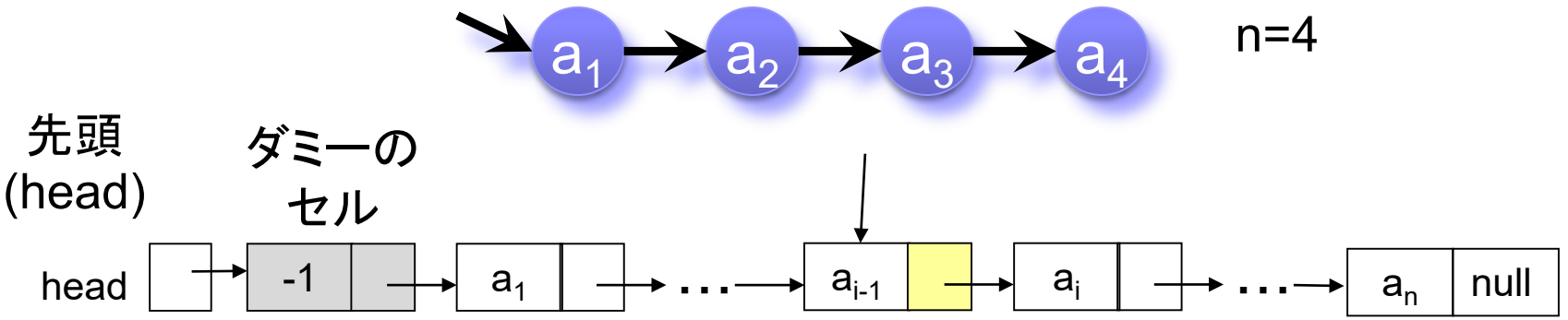
```
int A[100];                                  静的割付

int *A = (int *)malloc(sizeof(int)*100);     動的割付
```

リスト：連結リスト(linked list)による実装



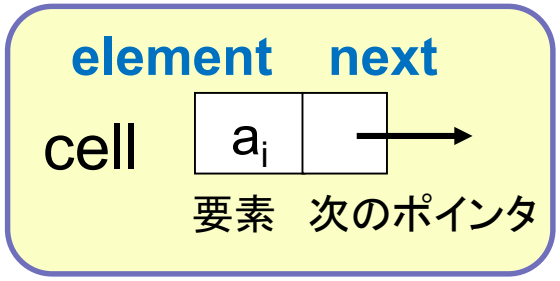
- 要素を保持する「セル」をポインタでつないで，リストを表す．
- 途中への挿入削除を効率良く実行できる



セル*の構造体（C言語のコード）

```
typedef struct _cell {
    int element;
    struct cell *next;
} cell;
```

セル*（箱図）

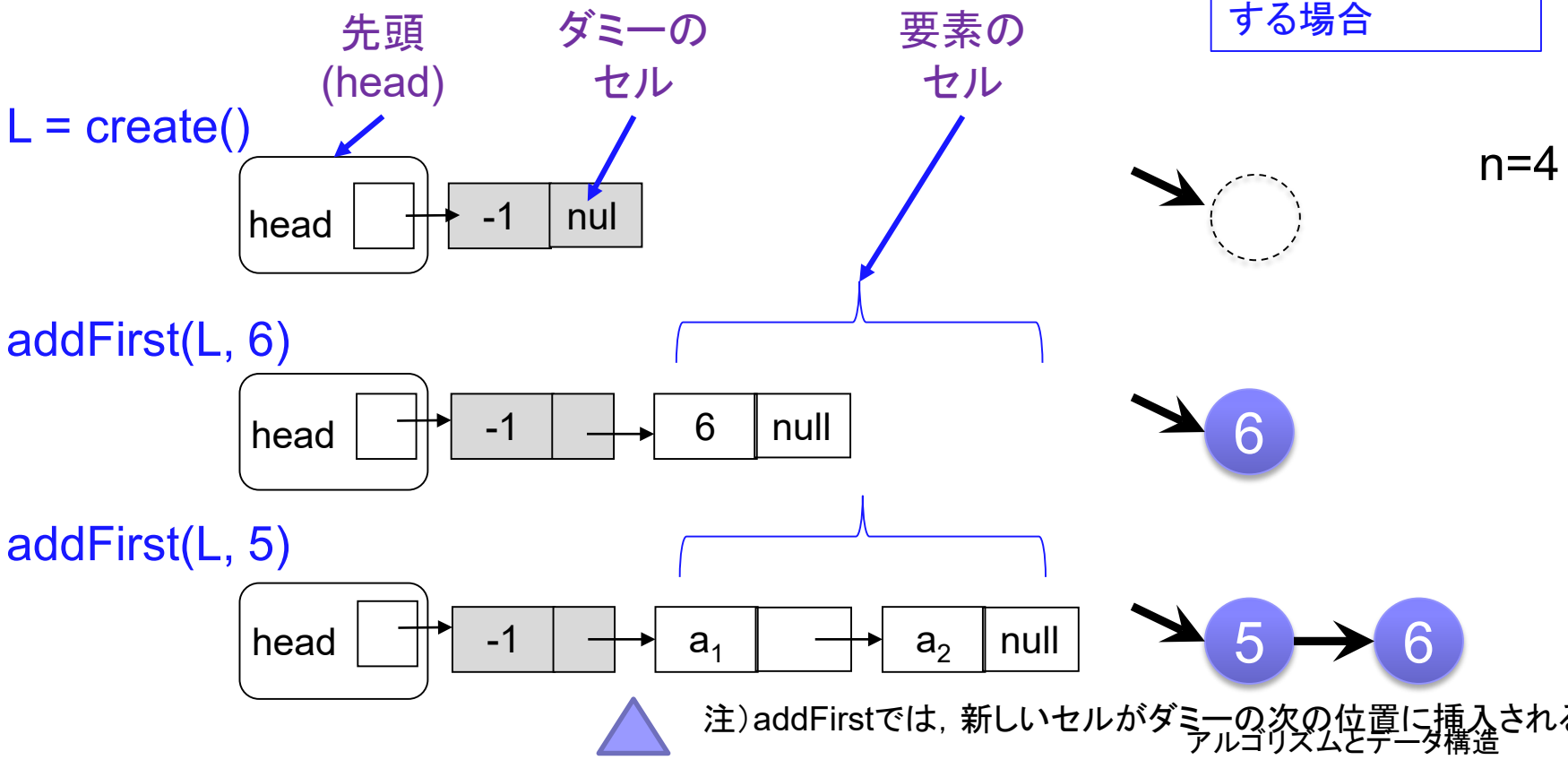


*セル = 区切られた箱や部屋のこと

リスト：連結リスト(linked list)による実装

- 要素を保持する「セル」をポインタでつないで，リストを表す．
- 途中への挿入削除を効率よくで行える
- 工夫として，先頭にダミーのセルを入れる

例：addFirst(L, a) がリストLの先頭に要素を追加する場合



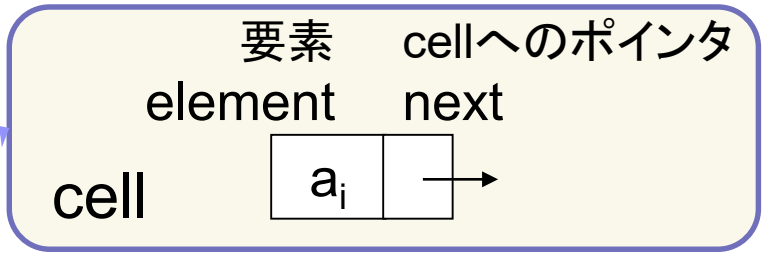
注）addFirstでは，新しいセルがダミーの次の位置に挿入される

C言語によるリストの定義(1/3)

連結リスト

```
typedef struct _cell {
    int element;
    struct cell *next;
} cell;


typedef struct _list {
   cell *head;
} list;
```



struct cell {･･･}: 構造体cellを･･･と定義

typedef ･･･ cell: ･･･をデータタイプcell型として定義

initはcell型データを指すポインタ型

```
list *create() {
   list *L = (list *)malloc(sizeof(list));
   L->head = (cell *)malloc(sizeof(cell));
   L->head->next = NULL;
   L->head->element = -1;
   return L;
}
```

sizeof(cell): cell型のデータサイズ（バイト）

malloc(n):nバイトのメモリーを確保

(cell *)malloc(n): 確保したnバイトの領域をcell型データ格納領域とみなす。

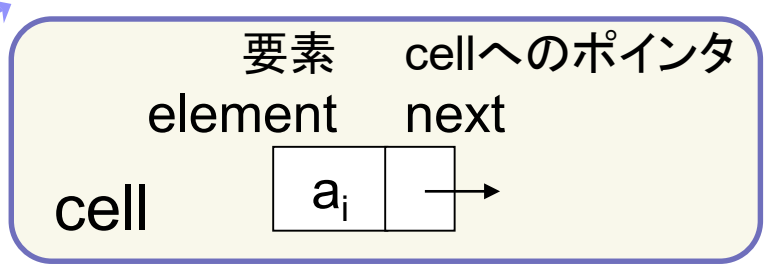
C言語によるリストの定義(2/3)

連結リスト

```
typedef struct _cell {
    int element;
    struct cell *next;
} cell;


typedef struct _list {
   cell *head;
}
```

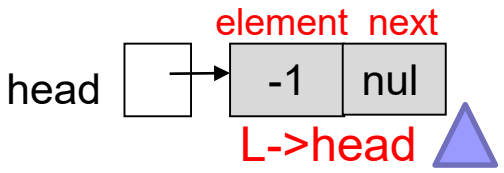
```
list *create() { //空リストを生成して返す
   list *L = (list *)malloc(sizeof(list));
   L->head = (cell *)malloc(sizeof(cell));
   L->head->next = NULL;
   L->head->element = -1;
   return L;
}
```



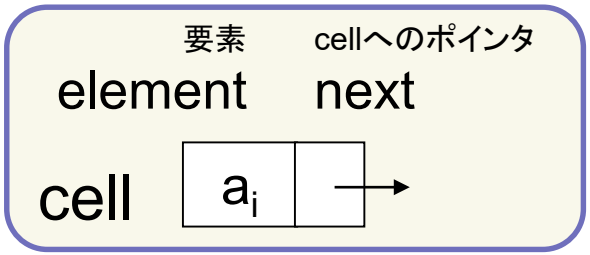
先頭
(head)

ダミーの
セル

L = create()の直後



C言語によるリストの定義(3/3)



連結リスト

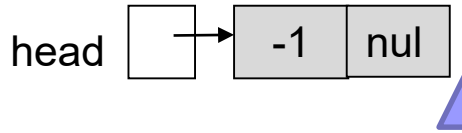
```
list *create() {
   list *L = (list *)malloc(sizeof(int));
   L->head = (cell *)malloc(sizeof(int));
   L->head->next = NULL;
   L->head->element = -1;
   return L;
}
```

```
//リストの先頭に要素を挿入する
void addFirst(list *L, int element) {
  cell *add = (cell*)malloc(sizeof(cell));
  add->element = element;
  add->next = L->head->next;
  L->head->next = add;
}
```

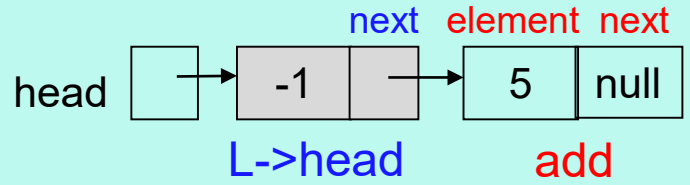
先頭
(head)

ダミーの
セル

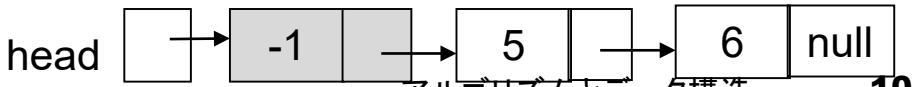
L = create()の直後



addFirst(L, 5)の直後

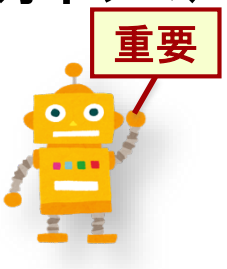


addFirst(L, 6)の直後



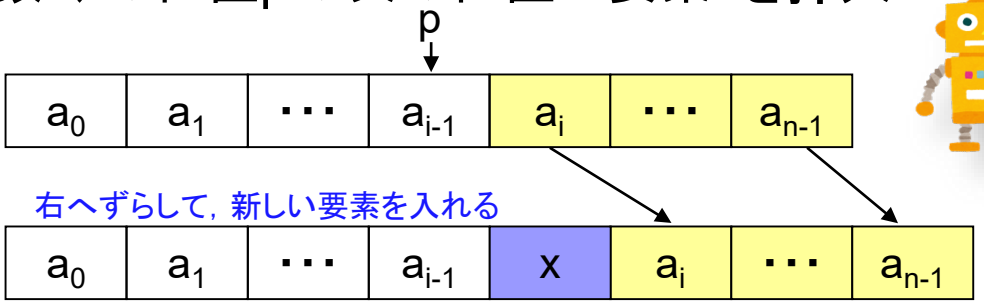
連結リストが得意な処理（挿入）

INSERT(x,p,L) : リストL（要素数n）の位置pの次の位置に要素xを挿入

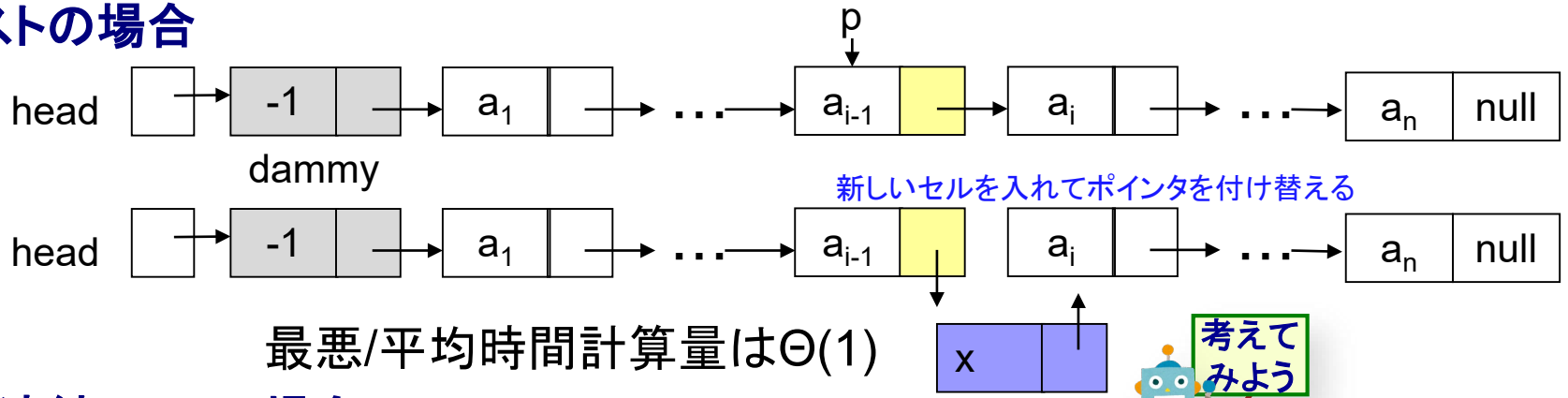


配列の場合

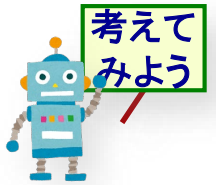
最悪/平均
時間計算量はΘ(n)



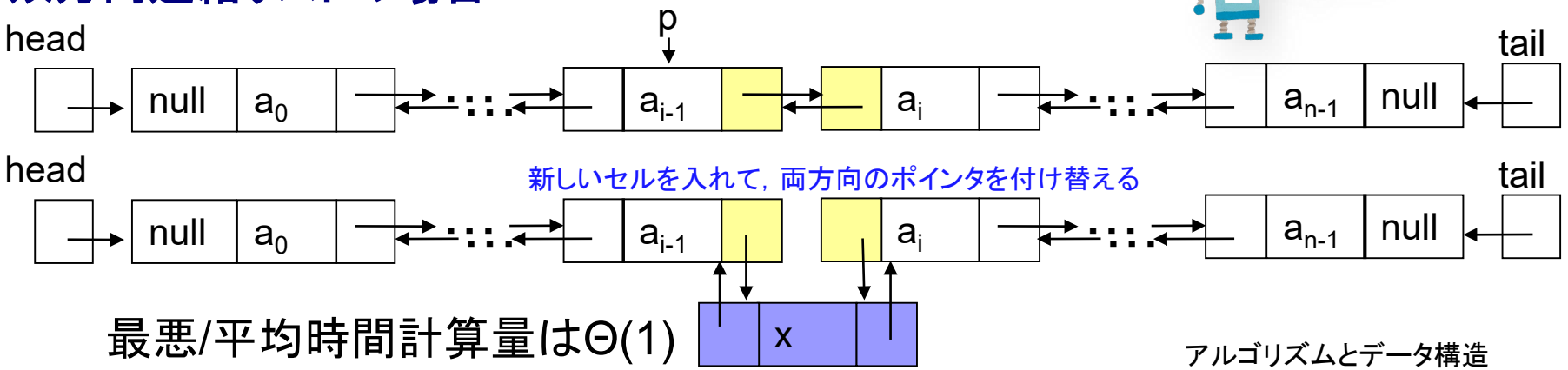
連結リストの場合



最悪/平均時間計算量はΘ(1)



双方向連結リストの場合



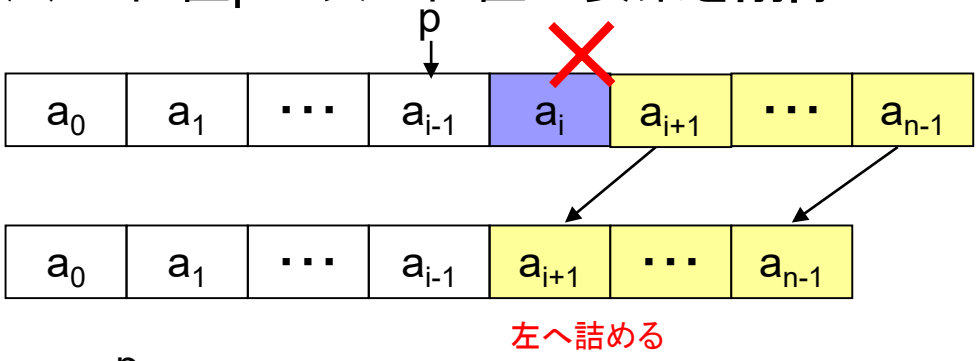
最悪/平均時間計算量はΘ(1)

連結リストが得意な処理(削除)

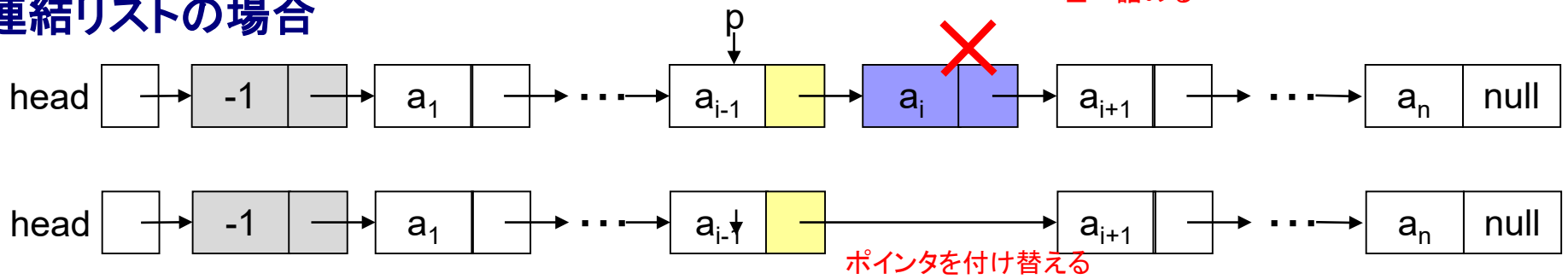
DELETE(p,L) : リストL(要素数n)の位置pの次の位置の要素を**削除**

配列の場合

最悪/平均
時間計算量はΘ(n)

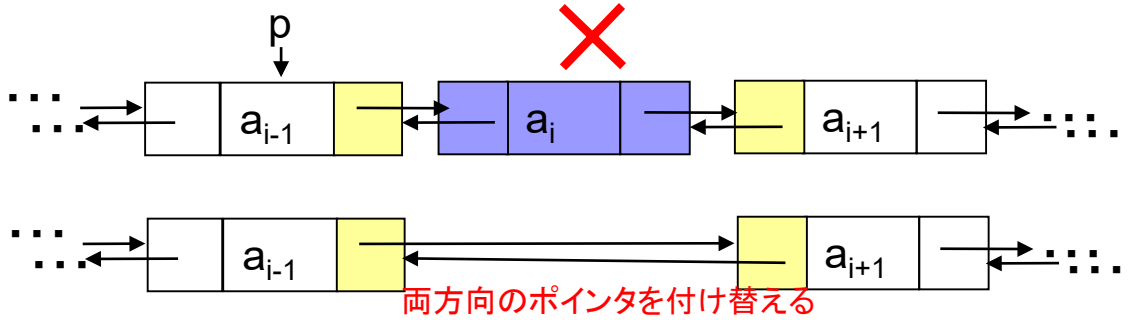


連結リストの場合



最悪/平均時間計算量はΘ(1)

双方向連結リストの場合



最悪/平均時間計算量はΘ(1)

まとめ

- 連結リストの実装の詳細を学んだ
  - 通常のheadが要素セルの列を指すやりかただと，先頭セルかその他の場合かで，挿入と削除に注意が必要．
- 工夫：先頭にダミーセルを入れる
  - 各種の挿入・削除演算が統一的にかける
  - セルの位置＝そのセルの一つ前のセルのポインタをもつ
  - 先頭への挿入(addFirst)＝ダミーセルの次に新セルを挿入する
  - 挿入(insert) ＝セルポインタの次に新セルを挿入する
  - 削除(delete) ＝セルポインタの次のセルを削除する
- 双方向リストの場合は，ダミーセルは不要