アルゴリズムとデータ構造

第4回基本的なデータ構造　その2
（リストの連結リストによる実現）

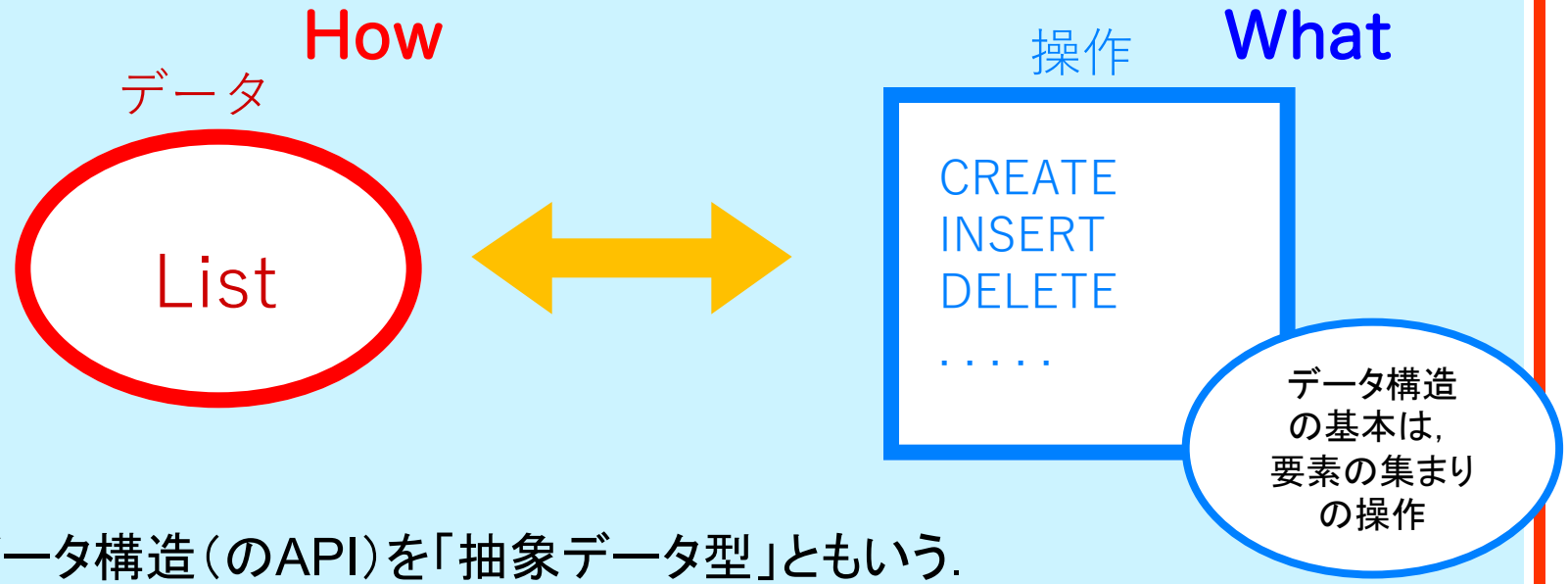
第4回基本データ構造(続)

- 前回の内容:
  - スタック:配列(と構造体)による実装
  - キュー :配列(と構造体)による実装
- 今日の内容:
  - 抽象データ型としてのリスト
  - リストの実装:双方向連結リストによる実装(ポインタ)
    - スタックとキューのリストによる別の実装
  - 付録:スタックの応用例2:逆ポーランド記法電卓(演習課題2の問4*)
- ポイント
  - 抽象データ型とその実装方法(プログラム)
  - ポインタを用いたデータ構造

第5回「二分探索木」でポインタ構造を再び学ぶ予定

抽象データ型 (Abstract Data Type)とは？

データ型を，それに適用される一組の操作で抽象的に定めたもの.



- データ構造（のAPI）を「抽象データ型」ともいう.
- データ構造には，「それは何か（What）」と「それをどのように実現するか？（How）」の二つの面がある.
- 現代的なプログラム言語やライブラリーはこの考え方に基づく.（例：C++，Java，Ruby, Python などなど）

授業で学ぶデータ構造の範囲

「プログラミング」で学ぶところ

機械語のデータ型

- レジスタ値とその番地

機械語（型がない）

基本データ型

- char, int, large int, double,

昔の言語も持っている型（C，Pascal）

構造データ型

- 配列（array），構造体（struct）

最近の言語（C++，Java, etc.）

基本的データ構造

- スタック（stack），待ち行列（queue），リスト（list）

今日のトピック

先進的データ構造

- 二分探索木（binary search tree），平衡探索木（balanced search tree），ハッシュ表（hash table）

「アルゴリズムとデータ構造」の範囲

今日のあらすじ

授業のはじめに，抽象データ型としてのリストを紹介する

それから特殊なリストであるスタックとキューを紹介する

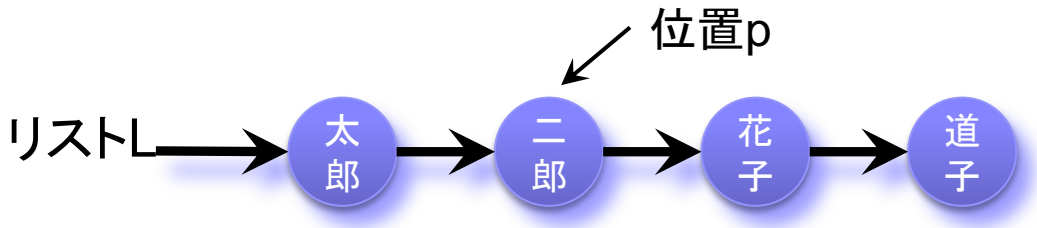
最後に，ふたたび，リストの実装に戻る

抽象データ型としてのリスト

基本

リストとは？（抽象データ型として）

0個以上の要素を一列にならべたもの



要素に順序があるところが，集合との違い

用語

- 空リスト：要素を含まないリストのこと
- リストの長さ：要素数 n
- A_i: 最初から i 番目の要素（$0 \leqq i \leqq n-1$）
- リスト中の場所を指示するための位置 p をもつ（要素へのポインタ）

基本

抽象データ型としての「リスト」に対する操作

- List L = create () : 空のリストを返す.

変更操作

- insert(L, p, x) : リストLの位置pの次に要素xを**挿入する**
- delete(L, p) : リストLの位置pの要素を**削除する**
- insert(L, x) : リストLの先頭位置に要素xを**挿入する**

変更操作がある
データ構造を
「動的データ
構造」という

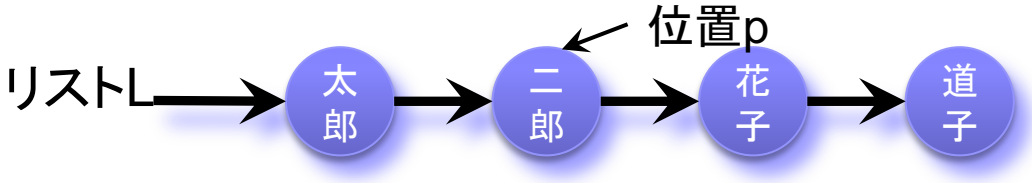
↔「静的
データ構造」

探索操作

- search (L, x) : リストLに要素xが含まれてるかを1と0で返す

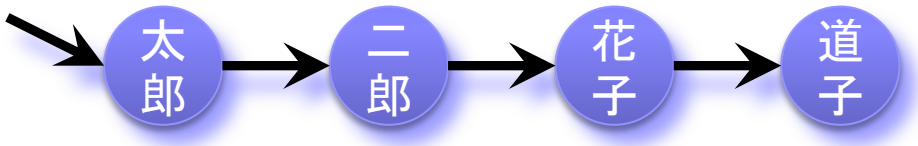
アクセス操作

- find(L, i) : リストLのi番目のセルの内容を返す（ランダムアクセス）
- last(L) : リストLの最後のセルの位置を返す
- next(L, p) : 位置pの1つ次のセルの位置を返す
- previous(L, p) : リストLにおいて、位置pの1つ前のセルの位置を返す



重要

リストの実装方法



リストとは

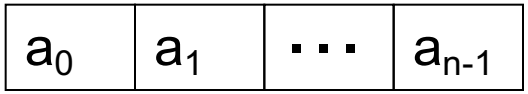
要素を0個以上1列に並べたもの

（注意）リストは連結リストを指すことが多い

[用語]「実装」とは、アルゴリズムや抽象データ型を、プログラムとして実際に作成すること、または、そのくわしい方法。

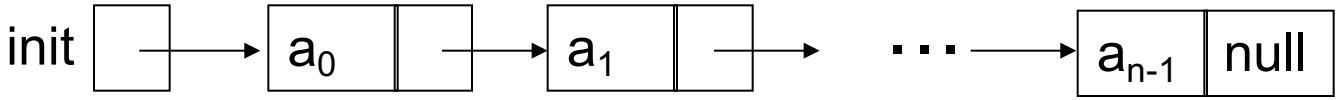
[リスト $a_0,a_1,\ldots,a_{n-1}$の実現法]

1. 配列(array)　n個の連続領域に格納



単純だが，
挿入・削除が
O(n)時間

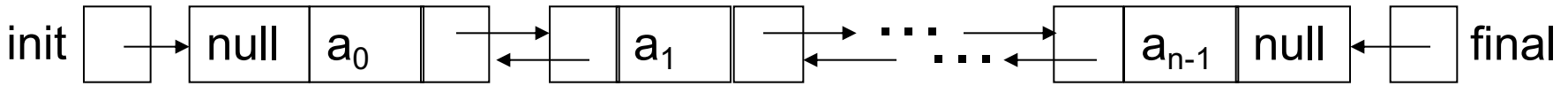
2. 連結リスト(linked list)　ポインタで次の要素の格納領域を指す（参考）



今日学ぶもの

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

挿入・削除がO(1)時間



注）initとfinalポインタを，head（先頭）とtail（末尾）と呼ぶことも多い．

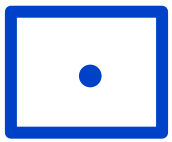
復習：ポインタとは？

ポインタ（pointer）

- セルの位置を示すデータ
- 機械語レベルでは、セルの番地そのもの
- プログラミングにおいては、その値を具体的に知る必要はない。

C言語の場合

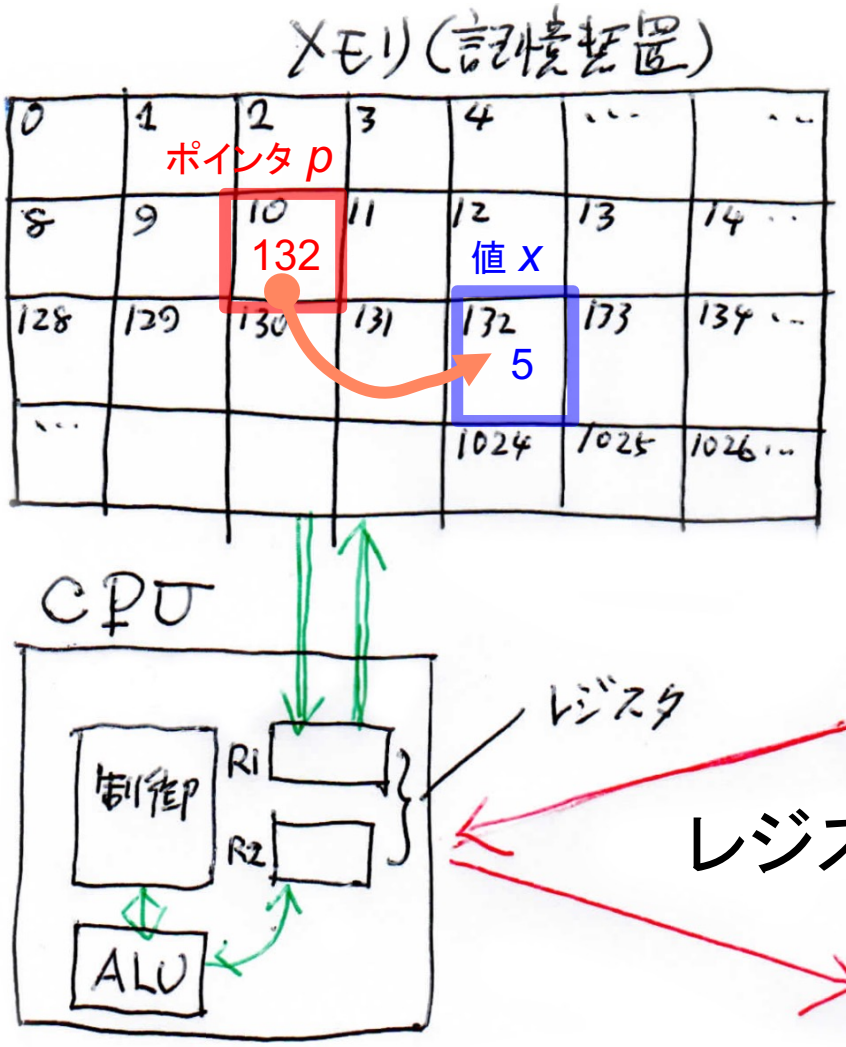
- ポインタpが指す変数の値xを、x = *pで表す。
- 変数xを指すポインタpを p = &x で取り出せる。



空ポインタ null

(C言語の場合)

復習：ポインタは番地



C言語の場合

- ポインタpが指す変数の値xを、x = *pで表す。
- 変数xを指すポインタpを p = &x で取り出せる。

レジスタレベルでみた計算機

今日のあらすじ

授業のはじめに，抽象データ型としてのリストを紹介する

それから特殊なリストであるスタックとキューを紹介する

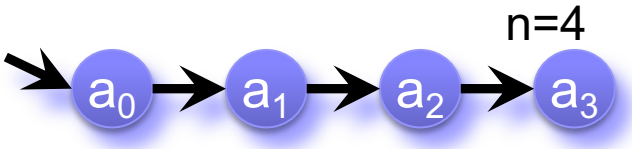
最後に，ふたたび，リストの実装に戻る

（単方向）連結リストを用いたリストの実現

基本

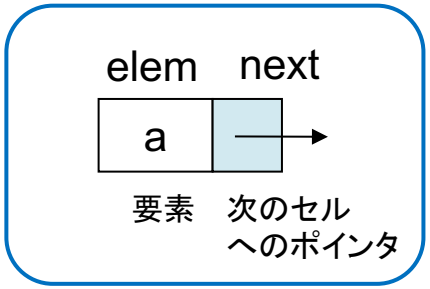
リストの（単方向）連結リストによる実装

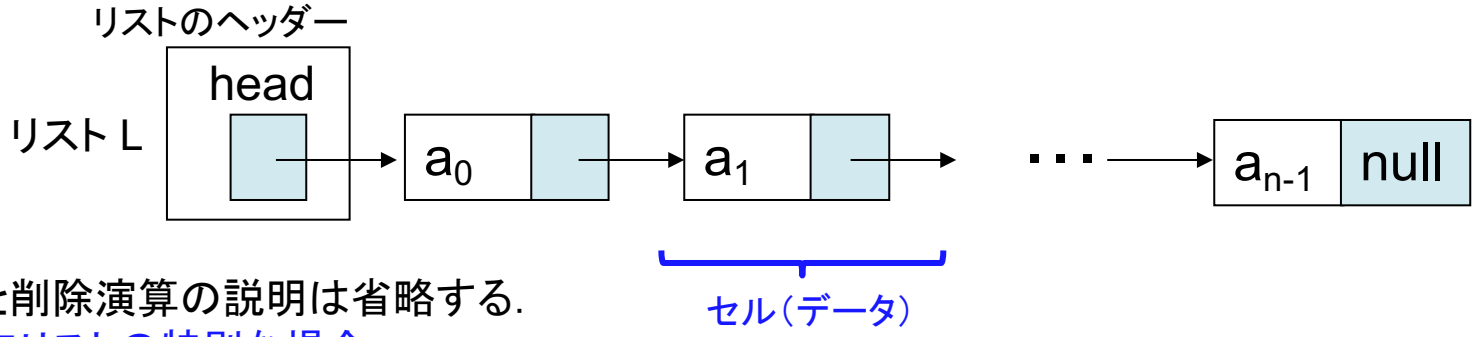
（単方向）連結リスト（singly linked list）



- 要素（elem）を保持するデータ（セル）をポインタでつないで，リストを表す．
  - 各**セル**は，要素（elem）と，直後のセル（next）を指すポインタをもつ
  - リストは，セルの列と，その先頭を指すポインタ(head)からなる．
- 位置 p（挿入場所のセルへのポインタ）が与えられたとき，途中への挿入削除を効率良く行える

セル（データ）





挿入演算と削除演算の説明は省略する．
次の双方向リストの特別な場合．
（末尾のスライド「計算量のまとめ」を参照のこと）

今日のあらすじ

授業のはじめに，抽象データ型としてのリストを紹介する

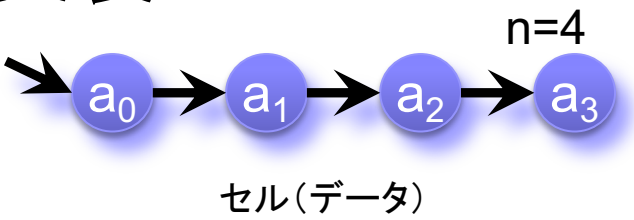
それから特殊なリストであるスタックとキューを紹介する

最後に，ふたたび，リストの実装に戻る

双方連結リストを用いたリストの実現

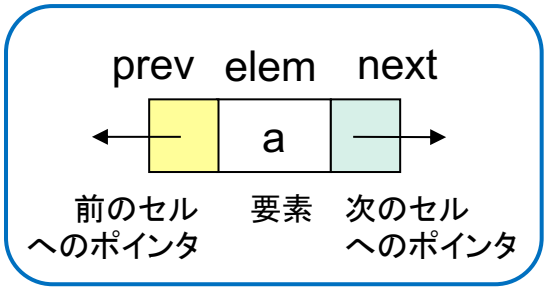
基本

リスト：双方向連結リストによる実装



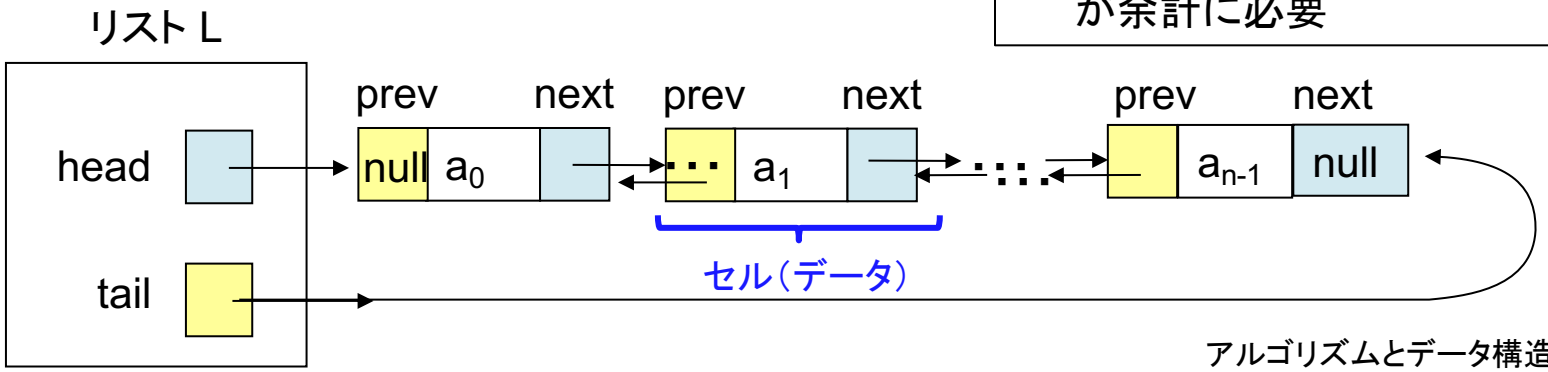
双方向連結リスト（doubly linked list）

- 要素（elem）を保持するデータ（セル）をポインタでつないで，リストを表す．
  - 各セルは，要素（elem）と，直前のセル(prev)と直後のセル（next）を指すポインタをもつ
  - リストは，セルの列と，その先頭(head)と末尾(tail)を指すポインタからなる．
  - 以下では，説明の簡略化のため，末尾ポインタ（tail）の操作の説明は略する（考えてみよう!）．
- 途中への挿入削除を効率良く行える



単方向リストとの比較：

- 長所：単方向リストよりも挿入・削除演算の実現が簡単．
- 短所：prevポインタ分のメモリが余計に必要



基本

実装:「リスト」に対する操作

○ の演算の実装方法を学ぶ

- List L = create () : 空のリストを返す.

変更操作

- ○ insert(L, p, x) : リストLの位置pの次に要素xを**挿入**
- ○ delete(L, p) : リストLの位置pの要素を**削除する**
- insert(L, x) : リストLの先頭位置に要素xを**挿入する**

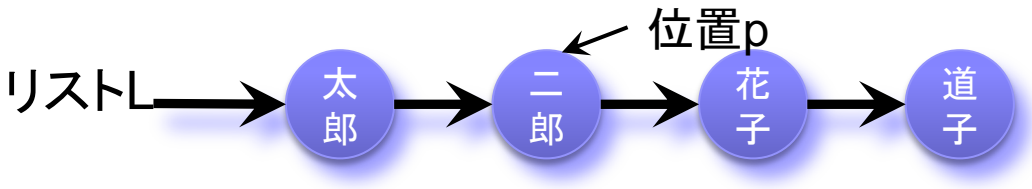
変更操作がある
データ構造を
「動的データ
構造」という
↔「静的データ構造」

探索操作

- search (L, x) : リストLに要素xが含まれてるかを1と0で返す

アクセス操作

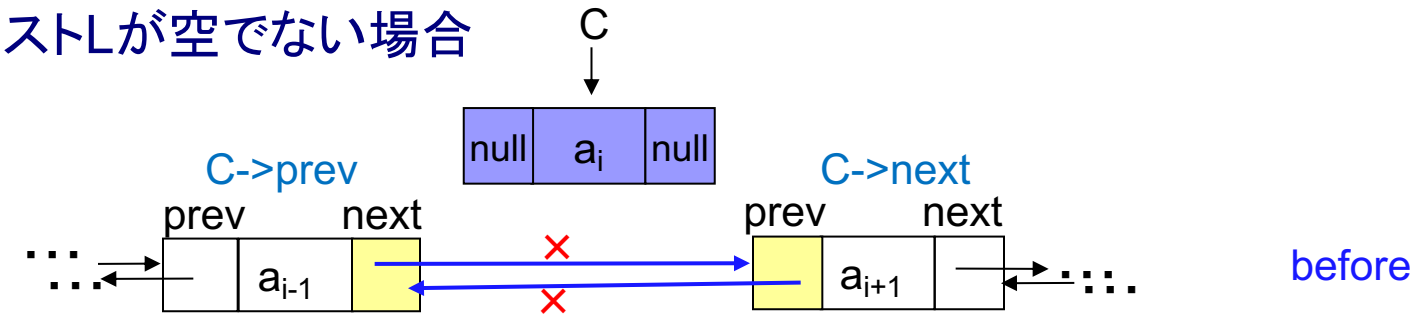
- ○ find(L, i) : リストLのi番目のセルの内容を返す(ランダムアクセス)
- last(L) : リストLの最後のセルの位置を返す
- next(L, p) : 位置pの1つ次のセルの位置を返す
- previous(L, p) : リストLにおいて、位置pの1つ前のセルの位置を返す



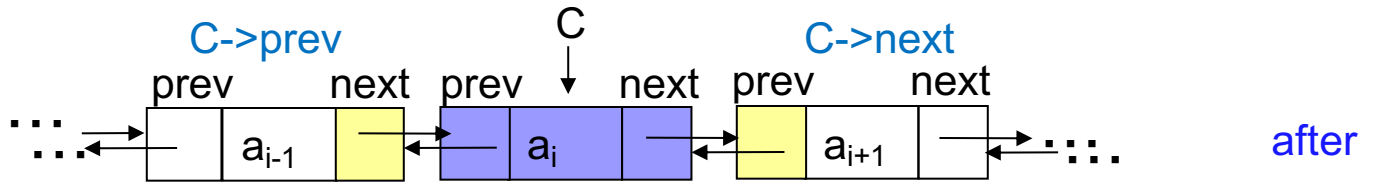
挿入演算: 一般の位置に挿入する場合

insert(L, p, C): リストLのポインタpの次に，データのセルCを挿入する

ケース1: リストLが空でない場合



insert(L, p, C):



ケース2: リストLが空の場合
ケース3: 先頭に挿入する場合
は前のページと同じ

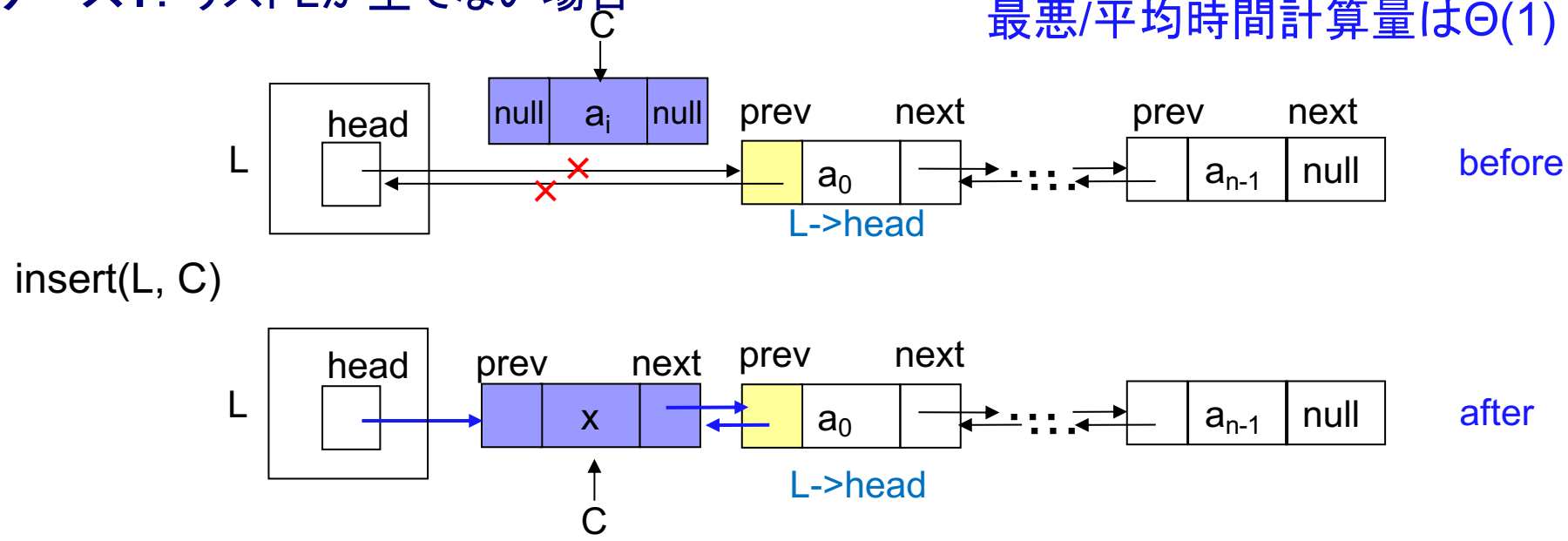
最悪/平均時間計算量は$\Theta(1)$

挿入演算：先頭に挿入する場合（特別な場合）

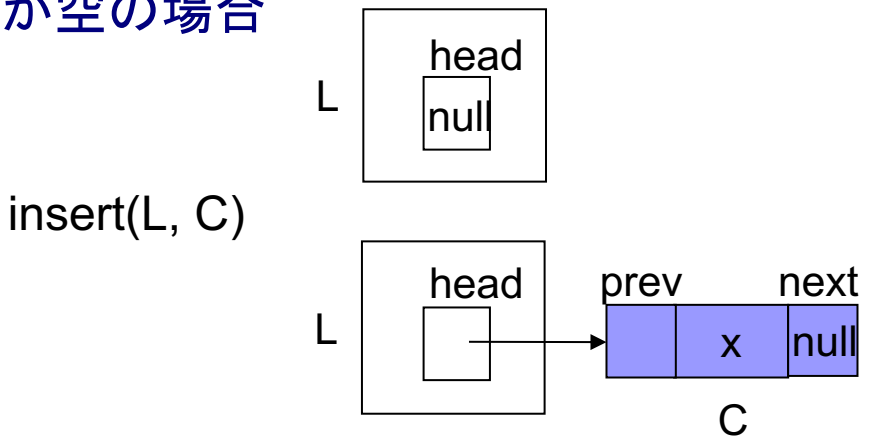
insert(L, C): リストLの先頭に，データのセルCを挿入する

ケース1：リストLが空でない場合

最悪/平均時間計算量はΘ(1)



ケース2: リストLが空の場合

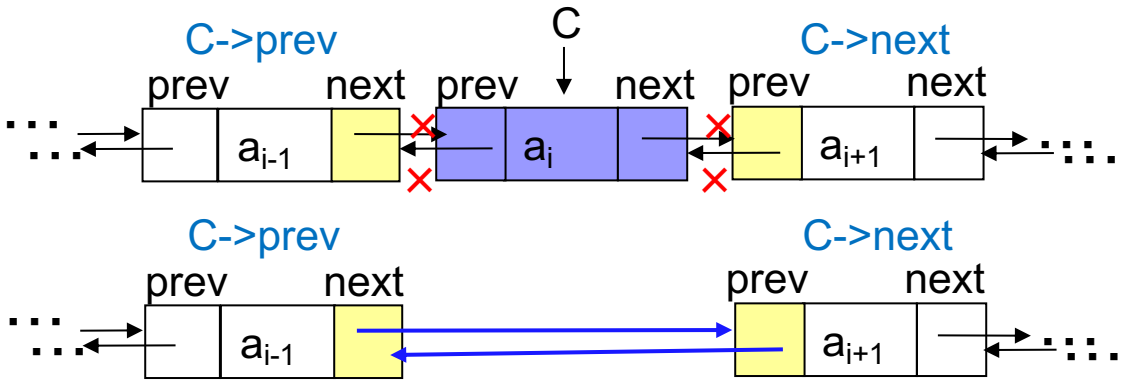


削除演算

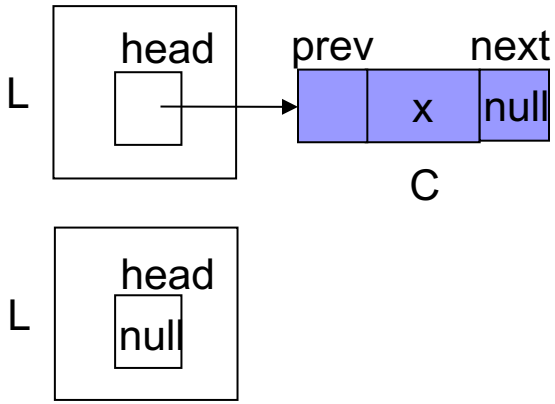
delete(L, C): リストLからデータのセルCを削除する

ケース1: データCが先頭でない場合

最悪/平均時間計算量はΘ(1)



ケース2: データCが先頭の場合



今日のあらすじ

授業のはじめに，抽象データ型
としてのリストを紹介する

それから特殊なリストである
スタックとキューを紹介する

最後に，ふたたび，リストの実装に戻る

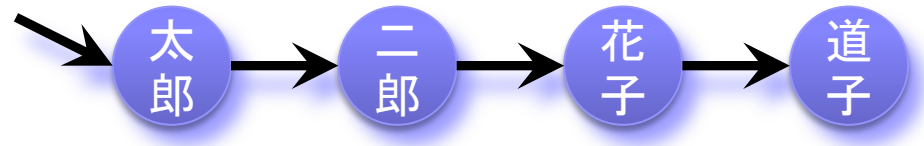
計算量の解析（まとめ）

3つのリストの実装方法

- 配列による実装，
- 連結リスト，
- 双方向連結リスト

について，各種の演算の計算量を解析する

復習：リストの実装方法



リストとは

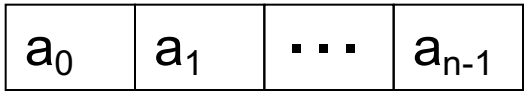
要素を0個以上1列に並べたもの

[用語]「実装」とは、アルゴリズムや抽象データ型を、プログラムとして実際に作成すること、または、そのくわしい方法。

（注意）リストは連結リストを指すことが多い

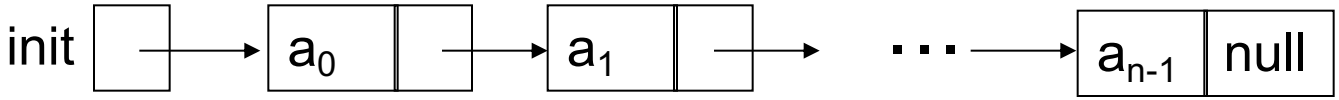
[リスト $a_0,a_1,\ldots,a_{n-1}$の実現法]

1. 配列(array)　n個の連続領域に格納



単純だが，
挿入・削除が
O(n)時間

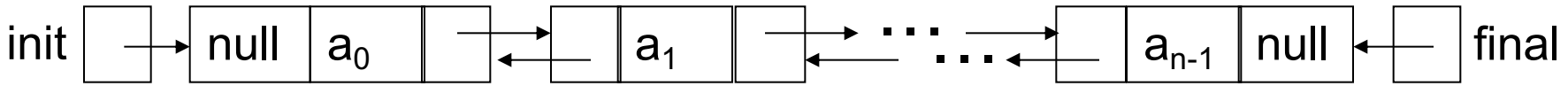
2. 連結リスト(linked list)　ポインタで次の要素の格納領域を指す（参考）



今日学ぶもの

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

挿入・削除がO(1)時間



注）initとfinalポインタを，head（先頭）とtail（末尾）と呼ぶことも多い．

基本

時間計算量：「リスト」に対する操作

○ の演算の時間計算量を見積もる

- List L = create () : 空のリストを返す.

変更操作

- ○ insert(L, p, x) : リストLの位置pの次に要素xを**挿入**
- ○ delete(L, p) : リストLの位置pの要素を**削除する**
- insert(L, x) : リストLの先頭位置に要素xを**挿入する**

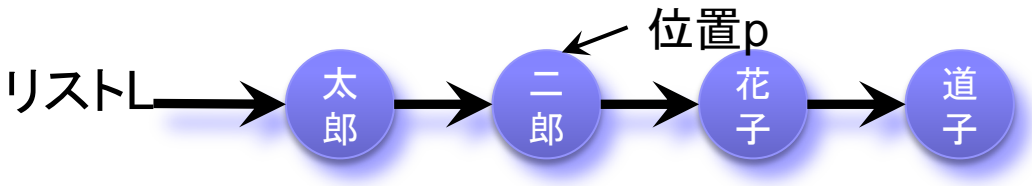
変更操作がある
データ構造を
「動的データ
構造」という
↔「静的データ構造」

探索操作

- search (L, x) : リストLに要素xが含まれてるかを1と0で返す

アクセス操作

- ○ find(L, i) : リストLのi番目のセルの内容を返す（ランダムアクセス）
- last(L) ：リストLの最後のセルの位置を返す
- next(L, p) : 位置pの1つ次のセルの位置を返す
- previous(L, p) : リストLにおいて、位置pの1つ前のセルの位置を返す

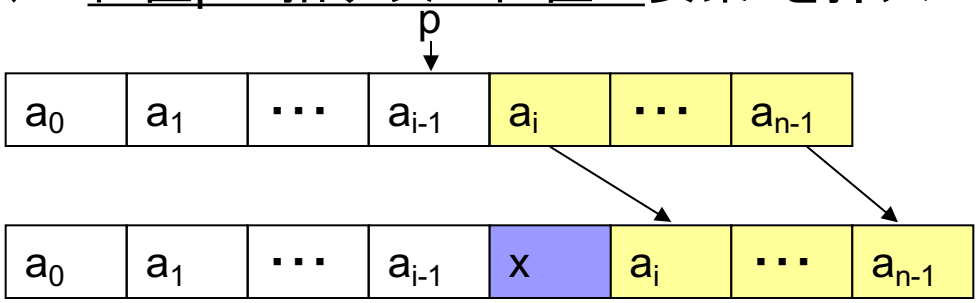


まとめ：挿入演算の計算量

重要

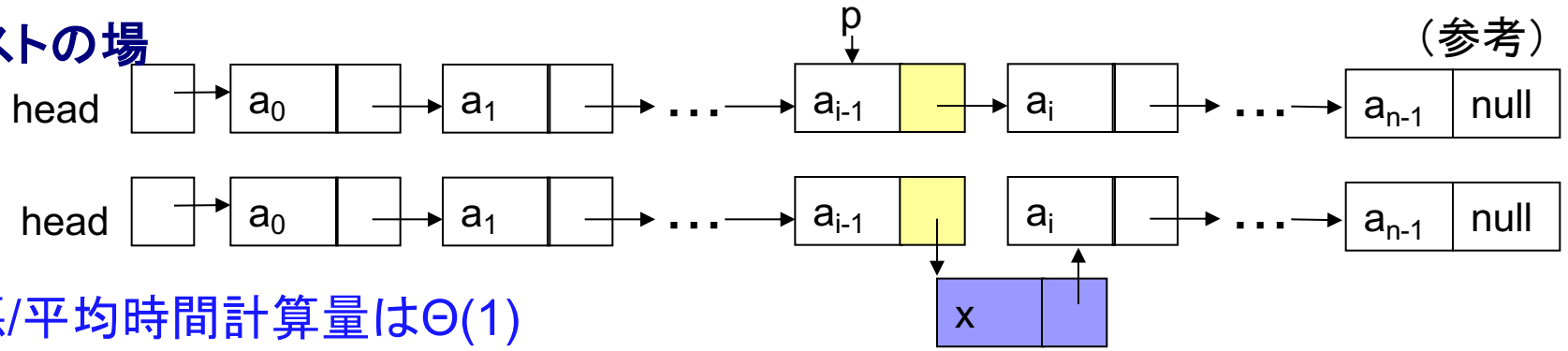
insert(x,p,L) : リストL（要素数n）の位置pの指す次の位置に要素xを**挿入**

配列の場合



最悪/平均時間計算量はΘ(n)

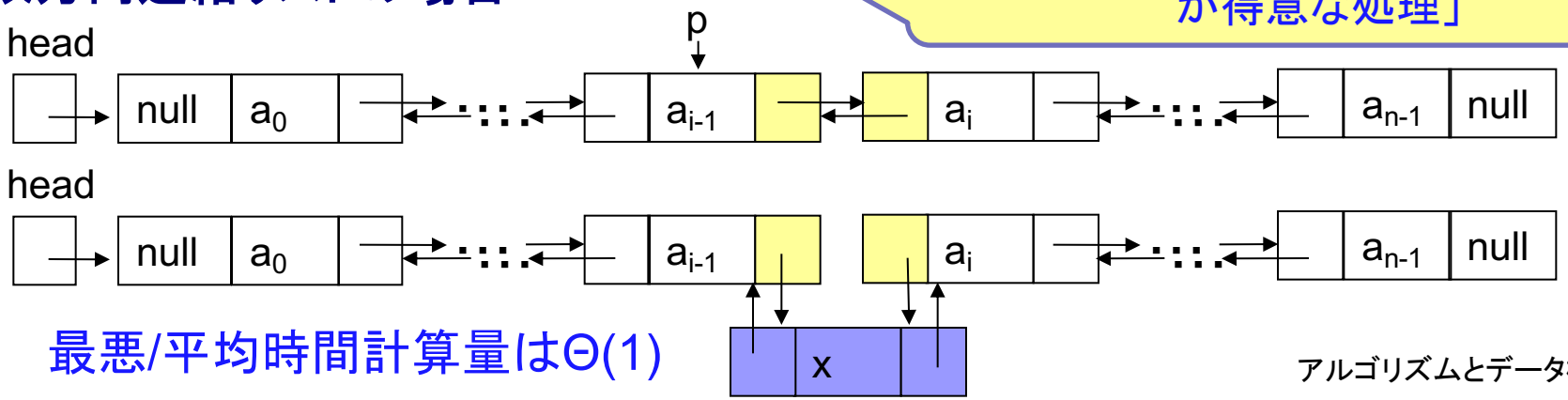
連結リストの場合



最悪/平均時間計算量はΘ(1)

「挿入は，連結リストと双方向連結リストが得意な処理」

双方向連結リストの場合



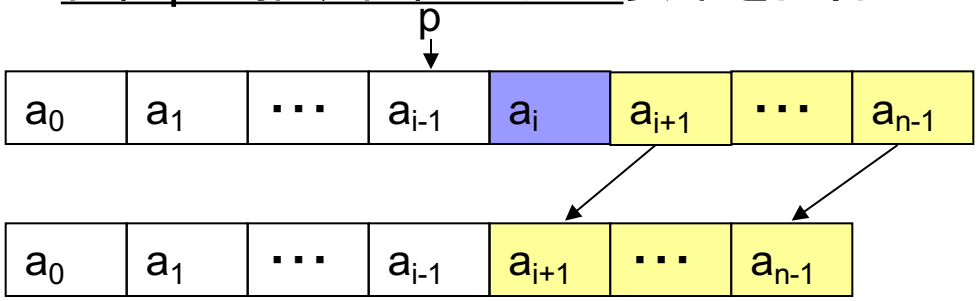
最悪/平均時間計算量はΘ(1)

まとめ：削除演算の計算量

発展

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

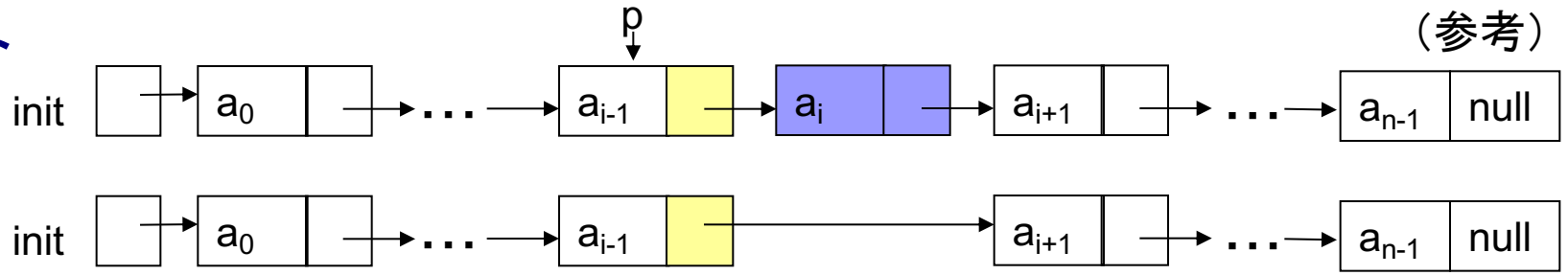
配列の場合



最悪/平均時間計算量はΘ(n)

連結リストの場合

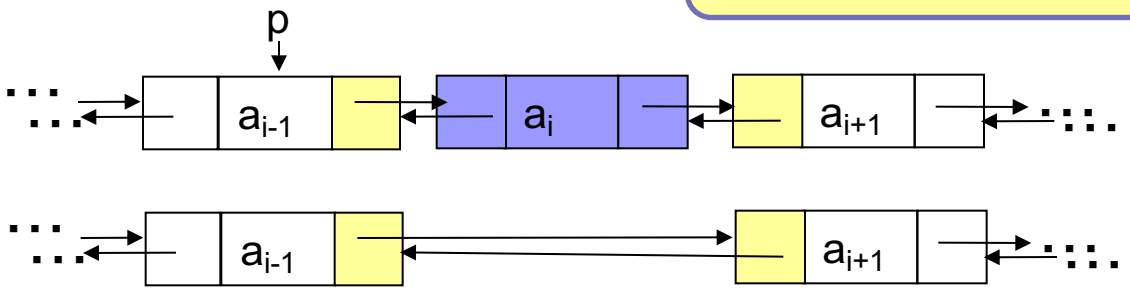
（参考）



最悪/平均時間計算量はΘ(1)

双方向連結リストの場合

「削除も，連結リストと双方向連結リストが得意な処理」



最悪/平均時間計算量はΘ(1)

まとめ：アクセス演算(FIND等)の計算量

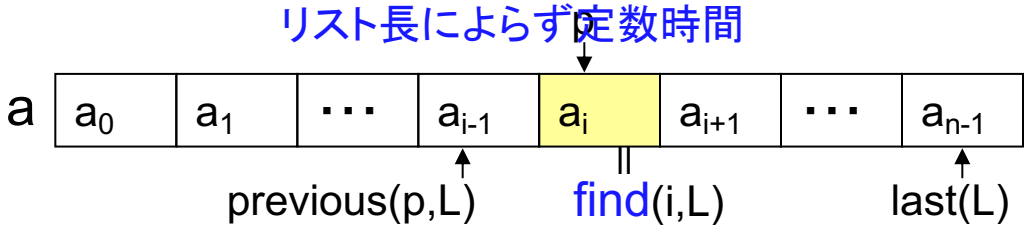
find(L, i) : リストL（要素数n）のi番目のセルの内容を返す
last (L) ：リストLの最後のセルの位置を返す
previous(L, p) : リストLにおいて、位置pの1つ前のセルの位置を返す

発展

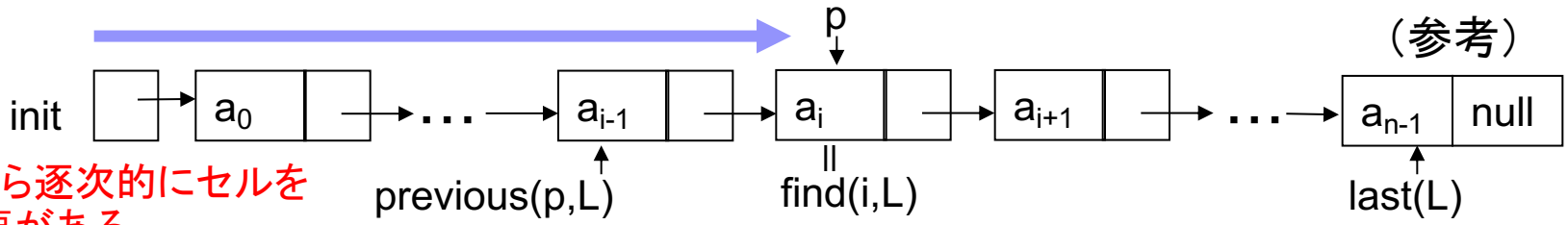
配列の場合

時間計算量 find(L, i): Θ(1),
last(L): Θ(1)
previous(L, p): Θ(1)

長所：ランダムアクセスが可能なので，
リスト長によらず定数時間



連結リストの場合



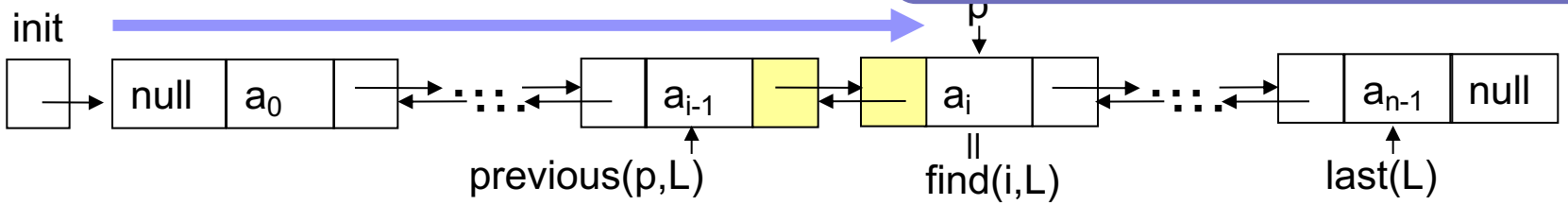
（参考）

短所：先頭から逐次的にセルを走査する必要がある

時間計算量: find(L, i): Θ(n), last(L): Θ(n), previous(L, p): Θ(n)

「アクセス操作は，配列が得意な処理．
連結リストと双方向連結リストは苦手」

双方向連結リストの場合



時間計算量: find(L, i) : Θ(n), last(L) : Θ(n), previous(L, p) : Θ(1)

まとめ

今日のあらすじ

- ✓ 授業のはじめに，抽象データ型としてのリストを紹介する
- ✓ それから特殊なリストであるスタックとキューを紹介する
- ✓ 最後に，ふたたび，リストの実装に戻る

第4回基本データ構造（続）

前回の内容：

- スタック：配列（と構造体）による実装
- キュー ：配列（と構造体）による実装

今日の内容：

- 抽象データ型としてのリスト
- リストの実装：双方向連結リストによる実装(ポインタ)
  - スタックとキューのリストによる別の実装
- 付録：スタックの応用例2：逆ポーランド記法電卓（演習課題2の問4*）

ポイント

- 抽象データ型とその実装方法（プログラム）
- ポインタを用いたデータ構造

第5回「二分探索木」でポインタ構造を再び学ぶ予定