プログラム設計とアルゴリズム

2. リスト構造

水谷　健太郎

非常勤講師

東京大学　大学院新領域創成科学研究科

特任助教

mizutani-kentaro@aoni.waseda.jp

2019年10月7日

前回の内容

- アルゴリズムとは何か
  問題を解く方法

- アルゴリズムの良さを比較するための基準として計算量を学んだ
  - オーダ
    big-o記法：O( )
  - 時間計算量
    アルゴリズムの実行にかかる時間
  - 空間計算量
    メモリの使用量

計算量
（複雑度）

本日の内容

- リスト構造と呼ばれるデータ構造について学ぶ
- 「配列」と「リスト構造」との違いを理解する

名簿データを格納する「配列」

氏名は50音順で並べる

アドレス	氏　名	住　所	電話	電子メール
1	青木春子			
2	石川夏雄			
3	上田秋子			
4	遠藤冬太			
：				
n	山田郁介			

会員に変動（追加・削除）がなければこの形式で十分
会員に変動（追加・削除）がある場合はどうか？

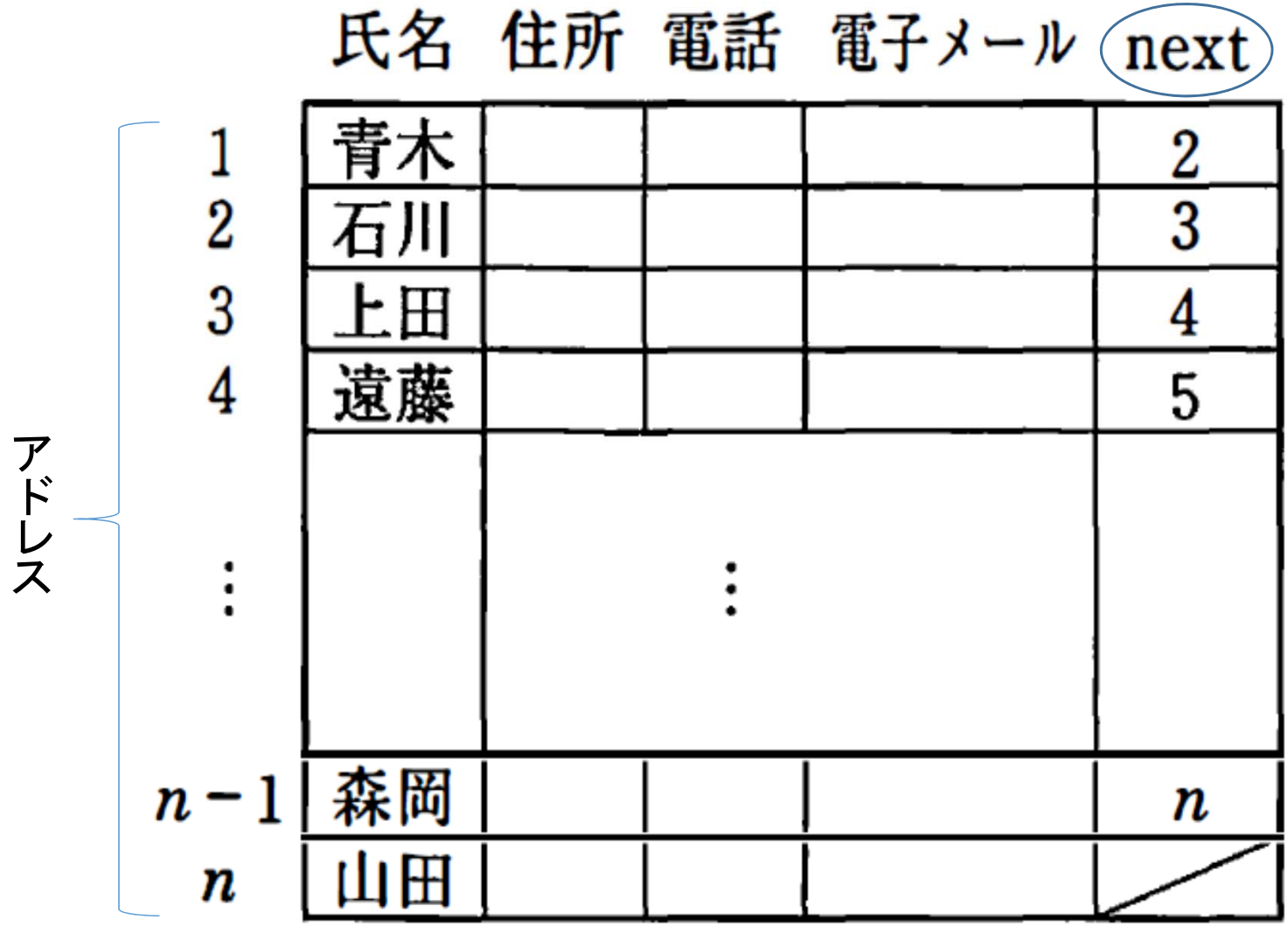
配列へのデータの追加・削除

（a）図2.1に「今井」を追加

（b）図2.1から「上田」を削除

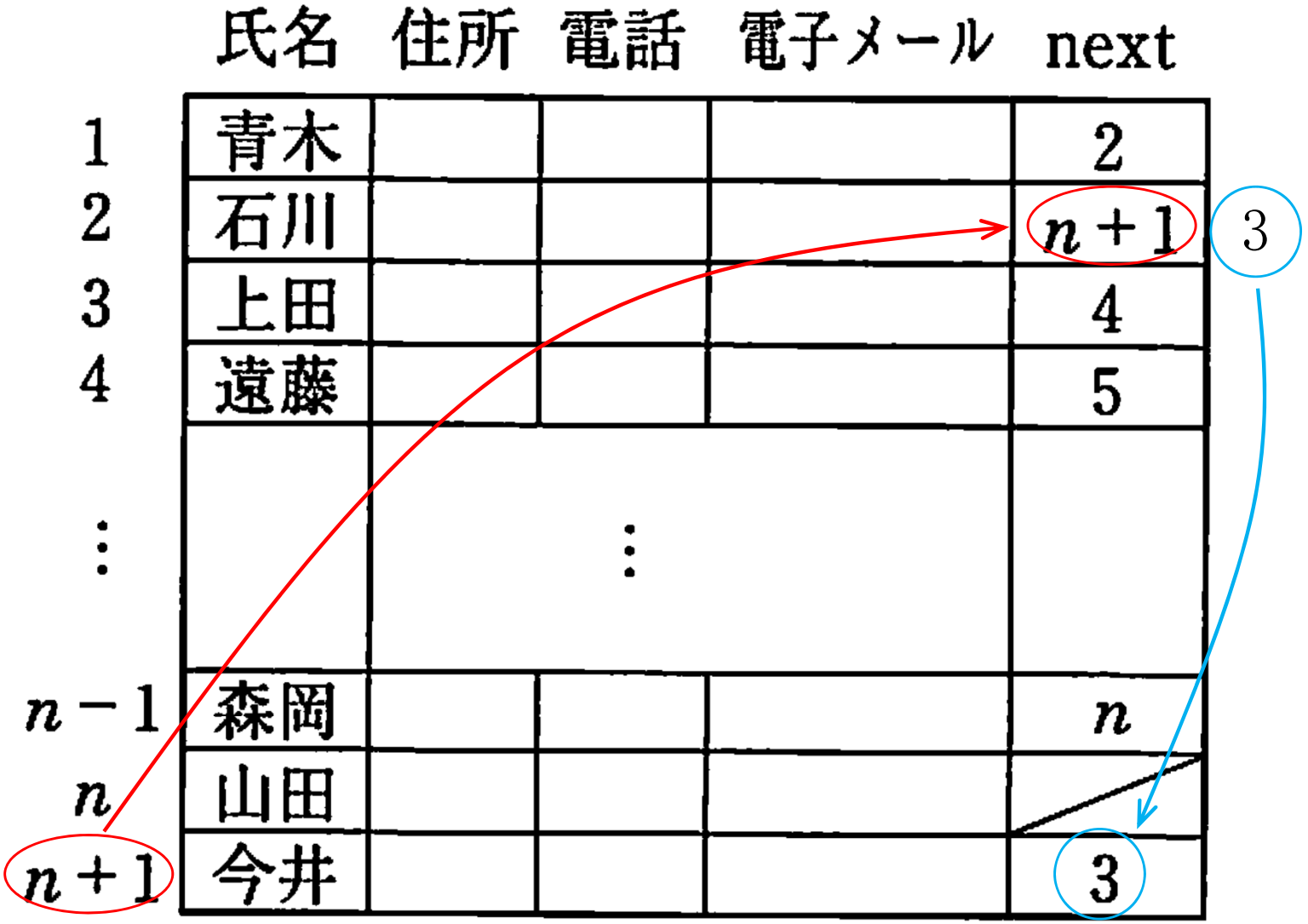
一人の追加・削除のために、データのほぼ全体に変更が及ぶ
変更箇所を小さい領域に限定できないか？

ポインタを表す列の追加



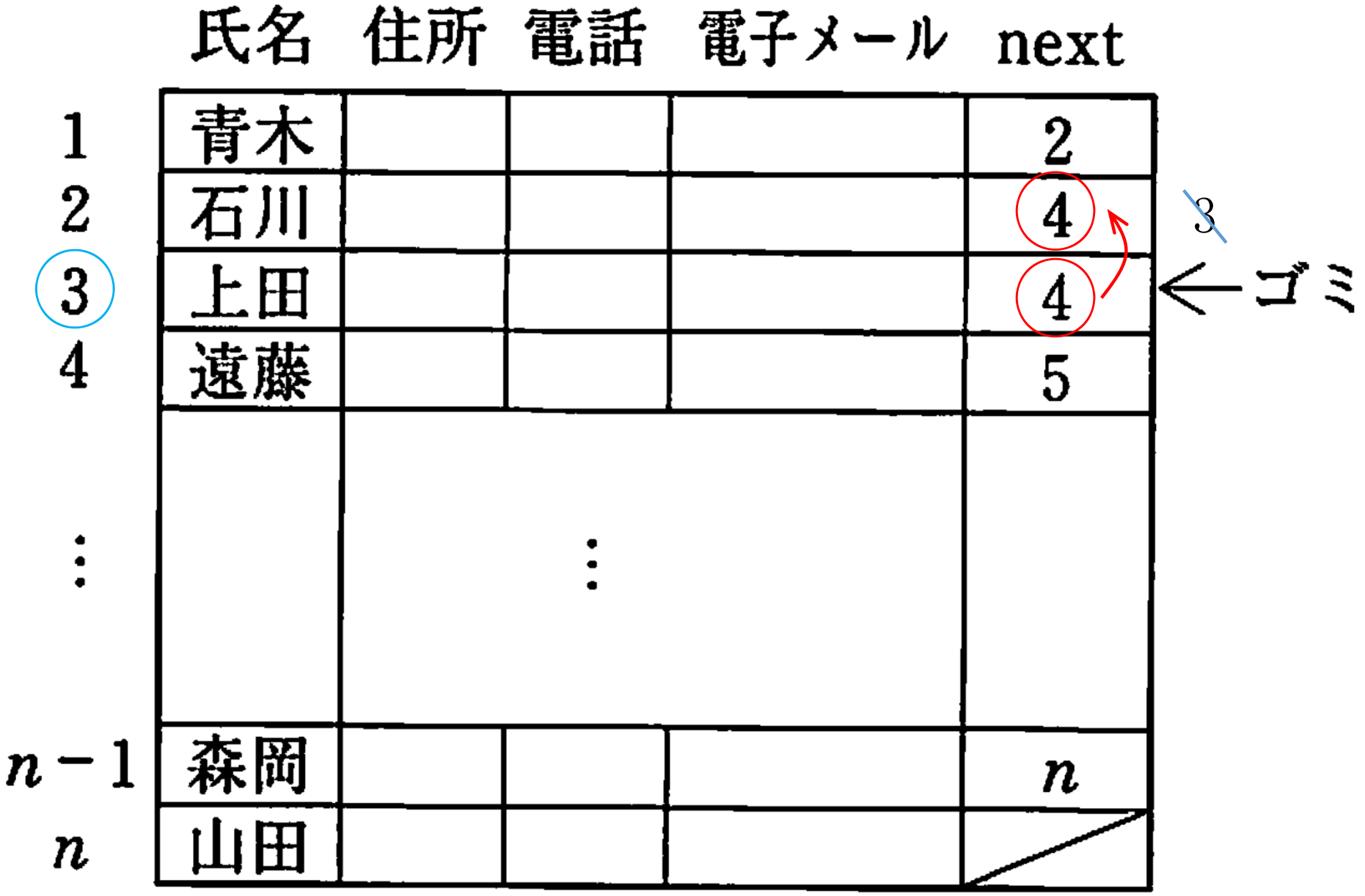
50音順で第i番目の次の会員はアドレスnext[i]に入っている

「今井」を追加



(b) (a) に「今井」を追加

「上田」を削除



追加・削除の処理

- 五十音順でアドレス i の人の次に入れるべき新会員のデータを追加する
  - アドレス（n+1）に新会員のデータを入れる。さらに

$$\mathrm{next}[n+1] \leftarrow \mathrm{next}[i]; \quad \mathrm{next}[i] \leftarrow n+1;$$

- i 行を削除する
  - i=next[j] を満たす j を見つけ

$$\mathrm{next}[j] \leftarrow \mathrm{next}[i]$$

ポインタ

- 記憶したいデータ自身ではなく、データとデータの関係を表現する情報をポインタ（pointer）という
  - 前の例では五十音順で次のデータの場所（アドレス）を示す情報

- ポインタを設けることは、記憶容量を少し余分に使うことになるが、上手に利用すると処理時間が劇的に短縮される

- C言語ではpointerという概念がある
- Java言語ではpointerという概念は明示的には出てこない⇒参照(reference)

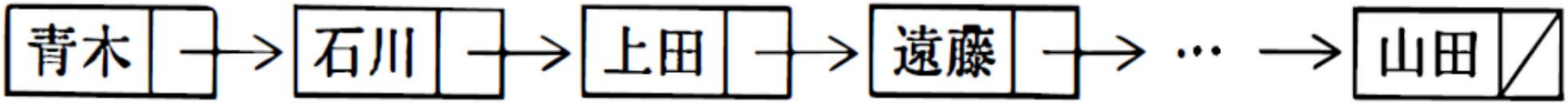
リスト構造（list structure）

- ポインタを用いたデータ構造
- データ＋ポインタ

- 以下では代表的なポインタの使い方（リスト構造）を見ていく

線形リスト（linear list）構造

- 各セルが次のセルを指すポインタを持つ



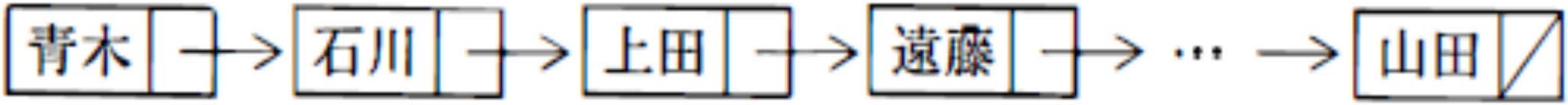
データ　　ポインタ⇒次のセルを指す

セル・ノード
（入れ物の単位）

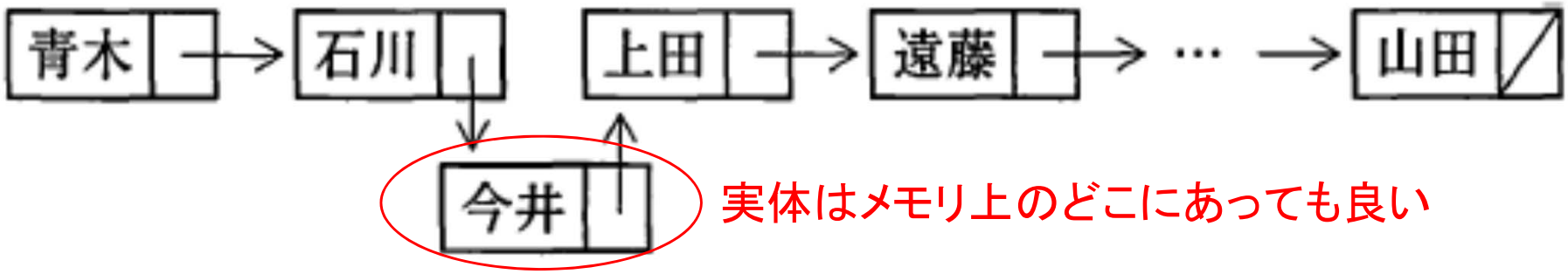
このセルをクラス (class)(Java/C++)や
構造体 (structure)(C)にすることが多い

- 一列に並べられたデータ構造となる

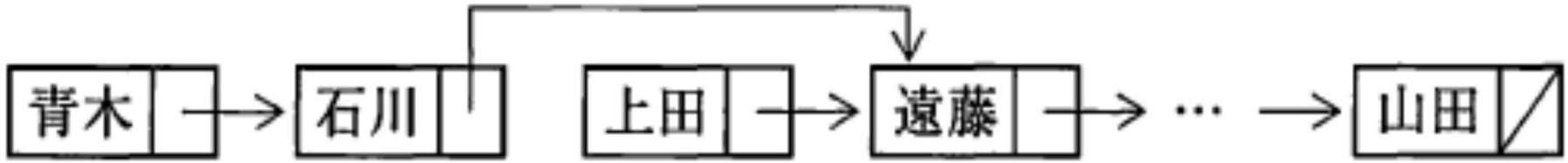
リスト構造への追加と削除



(a) 線形リスト



(b) (a) へ「今井」を追加



(c) (a) から「上田」を削除

リストと配列の違い

- リストは配列のように添え字を使ってデータにアクセスできない
  - E.g. array[5]
  - 配列は連続したメモリ空間にデータを格納
  - リストはデータごとに別々の空間にデータが格納される
- リストはデータの先頭（または末尾）から順に辿っていくことしかできない⇒シーケンシャル・アクセス(sequential-access)⇔ランダム・アクセス(random-access)
- リストはデータの変更（追加・削除）に対する処理が、配列よりも少ない
- 目的に応じて、配列を使うかリストを使うかを選択することが重要

削除にはコストがかかる

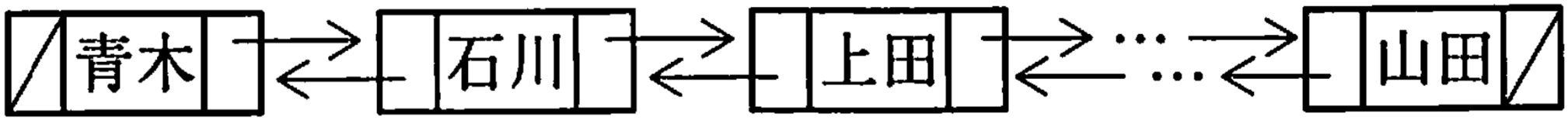
- 「上田」のセルを削除したい場合
- 「上田」を指しているセルのポインタを書き換える必要がある
- よって、どのセルが「上田」を指しているかを探す必要がある

  ⇒これには最悪O(n)の時間計算量がかかる

- これを回避するには、「次のセル」のポインタだけでなく「前のセル」のポインタを持つようにすればよい
  - 定数オーダO(1)でセルの削除が可能

二重線形リスト

- 「双方向リスト」とも呼ばれる



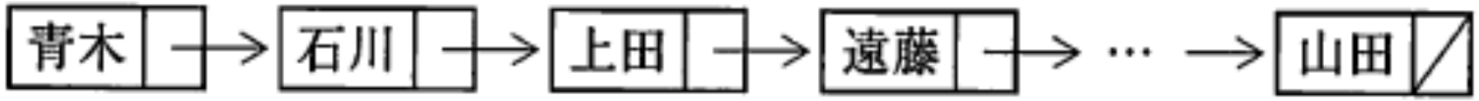
- 各々のノードが
  - 次のノード
  - 前のノード

  の両方に対するポインタを持つ

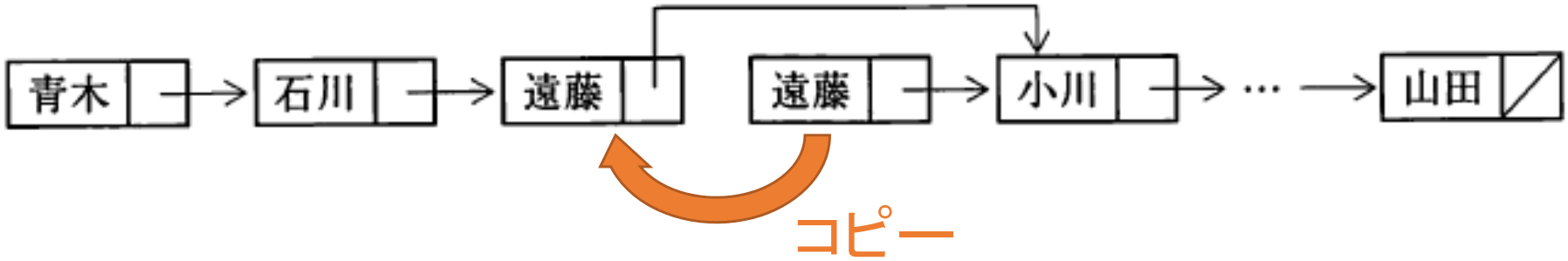
- こうすることにより、あるノードを指しているノードへO(1)で戻れる

O(1)の削除の別方法

- 「上田」のセルを削除したい



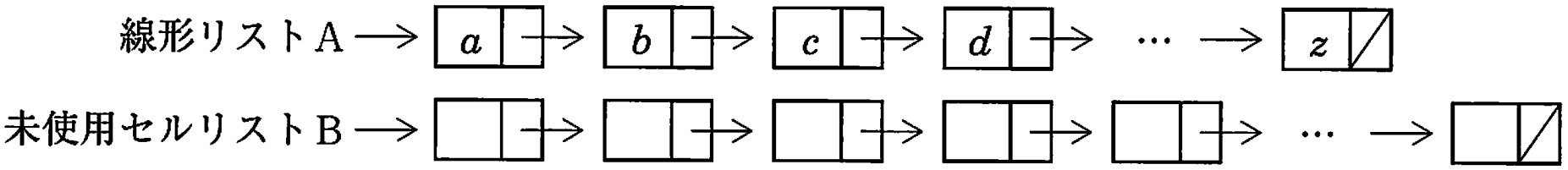
- 「上田」のセルの次の「遠藤」のセルを見つける
  - O(1) で可能
- 「遠藤」のセルに書かれている内容を「上田」のセルにコピーする（ポインタもコピーする）



ゴミ（garbage）の取り扱い

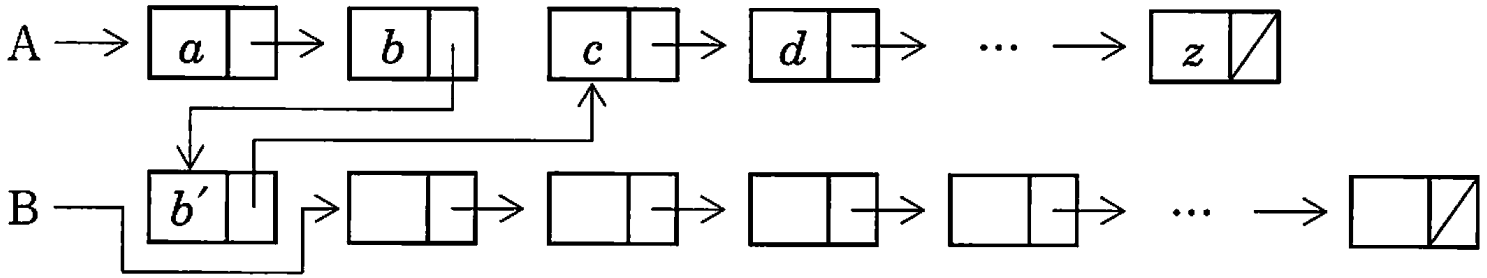
- セルの削除を行うと、どのセルからも指されない（参照されない）セルができる
- このセルをゴミ（garbage）と呼ぶ
- ゴミのセルは本来必要ないものであるので、そのままメモリに残しておくのは無駄
  - メモリーリーク(memory leak)となる
- 次スライドのような方法でゴミを管理することが可能

ゴミ(garbage)の取り扱い



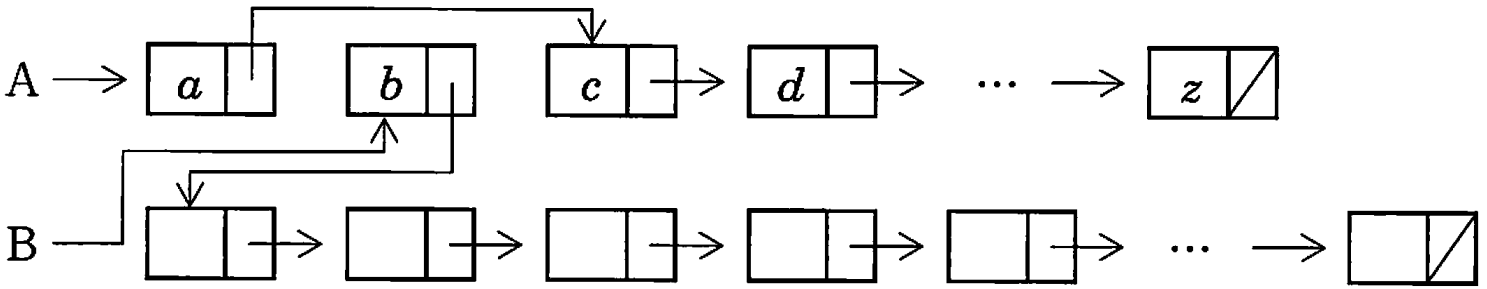
(a)

リスト A の b の次に b′ を挿入したい



(b)

b を消去したい



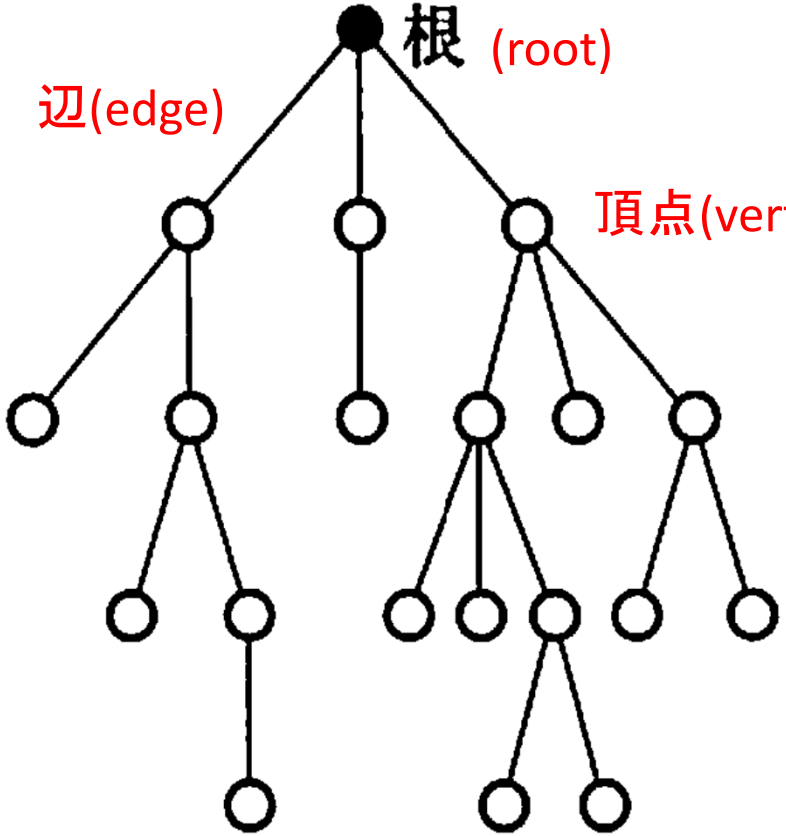
(c)

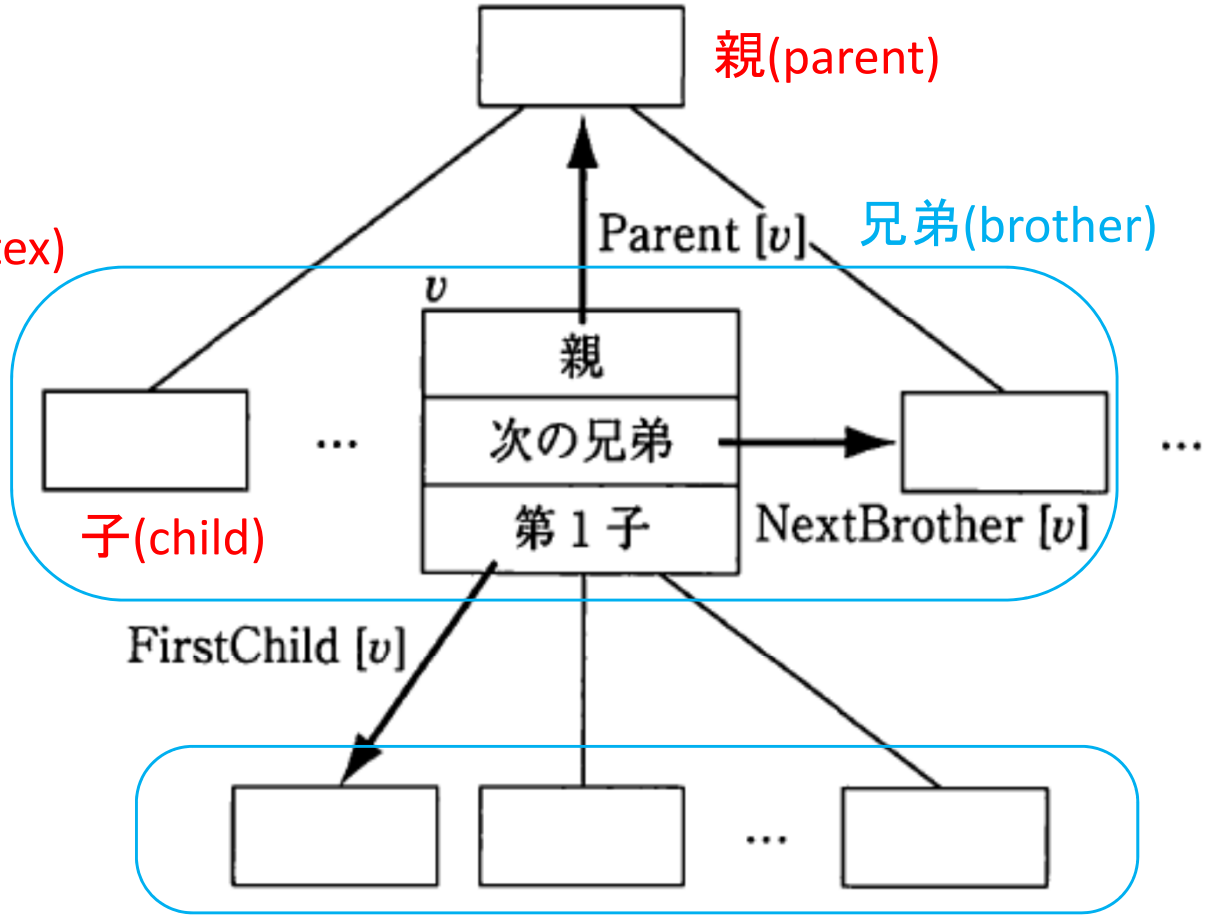
言語によるメモリ管理の違い

- Java言語
  - ほっといても、VMの方で自動的にメモリが開放される
    - ガベージコレクション（garbage collection; GC）
    - なので、未使用セルを明示的に管理しなくても問題ない
- C/C++言語
  - 自ら明示的にメモリ開放をする必要がある
    - やらないとメモリリークとなる
    - 前スライドの方法で管理するか、必要なくなった時点でメモリを開放する(free)かのどちらかを行う

※Windows, Mac OS, Linux等のOS（モダンOS）では、OS側でプログラム（プロセス）の終了時にメモリの解放を行ってくれる（が、ちゃんとメモリの解放を行いましょう）

根つき木





Parent[v]： 頂点 v の親をさすポインタ

FirstChild[v]： 頂点 v の第 1 子をさすポインタ

NextBrother[v]： 頂点 v の次の兄弟をさすポインタ

各頂点につき
3個のポインタ
だけで良い

NIL

- ポインタがさす相手がない場合
  - 根v: Parent[v] = NIL
  - 子を持たない頂点v: FirstChild[v] = NIL
  - vが最後の子: NextBrother[v] = NIL

- プログラム上ではnullで表される場合が多い

頂点 v の子をすべて列挙

$x \leftarrow \mathrm{FirstChild}[v];$

while $x \neq \mathrm{NIL}$ **do**

  report x and $x \leftarrow \mathrm{NextBrother}[x]$

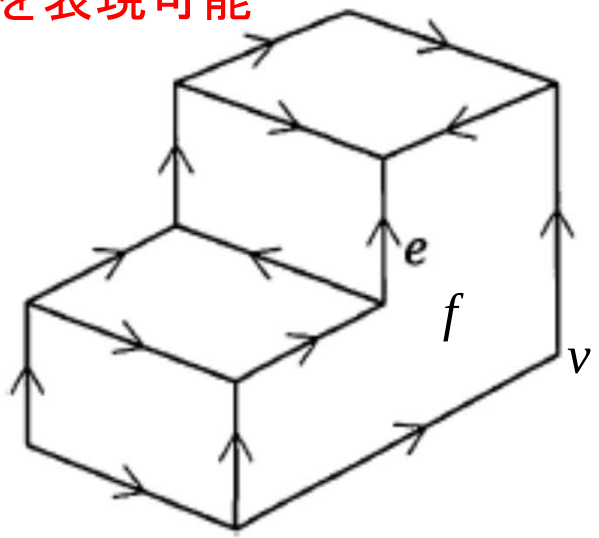
- vの兄弟すべてを列挙したければ、上の手続きの1行目を下記に変える。

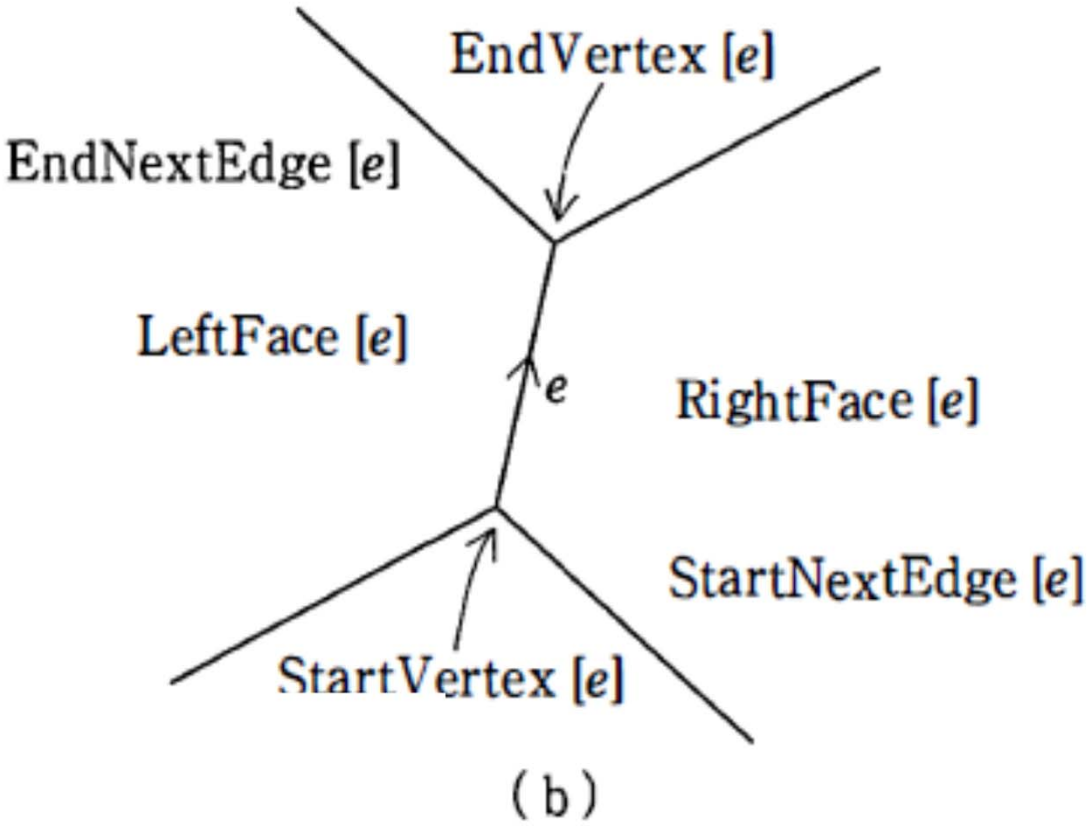
  $x \leftarrow \mathrm{FirstChild}[\mathrm{Parent}[v]]$

  （自分の兄弟＝自分の親の子）

参考：多面体（翼状エッジ構造）

稜線(e)に6つ、頂点(v)に1つ、面(f)に1個の固定の数のポインタで任意の多面体を表現可能





(b)

StartVertex[e]： e の始点をさすポインタ

EndVertex[e]： e の終点をさすポインタ

RightFace[e]： e の右の面をさすポインタ

LeftFace[e]： e の左の面をさすポインタ

StartNextEdge[e]： 始点のまわりで時計回りにまわったとき最初に出会う稜線をさすポインタ

EndNextEdge[e]： 終点のまわりで時計回りにまわったとき最初に出会う稜線をさすポインタ

Edge[v] 接続する稜線の一つを指すポインタ

Edge[f] 境界上の稜線の一つを指すポインタ

参考:面 f を囲むすべての稜線の列挙

$e_0 \leftarrow \mathrm{Edge}[f]$

$e \leftarrow e_0$

1 : **report** e;

if $\mathrm{RightFace}[e] = f$ **then** $e \leftarrow \mathrm{StartNextEdge}[e]$

else $e \leftarrow \mathrm{EndNextEdge}[e]$;

if $e = e_0$ **then** stop **else goto** 1

※翼状エッジ構造 (winged-edge structure)

レポート課題

- 二重線形リストを実装します
- Course N@viからファイルをダウンロードする
  - ２重線形リストを作成して表示を行うプログラム
    - C⇒report2.c
    - Java⇒Report2.java （クラス定義：Node.java）
- まずはこのソースコードを理解する
- その後、このファイルを基に、リスト構造に入っているノードの表示と削除を行うプログラムを作成する
  - 同じ値が複数、リストに含まれている場合は、最初のノードだけが削除されればよい
  - 実行例は次スライド
- 〆切：10月13日（日）23:59
  - ソースコードを提出すること
  - 動作確認をちゃんと行う

プログラム出力例

整数値を入力してください（0を入力すると終了）:1
整数値を入力してください（0を入力すると終了）:2
整数値を入力してください（0を入力すると終了）:3
整数値を入力してください（0を入力すると終了）:4
整数値を入力してください（0を入力すると終了）:5
整数値を入力してください（0を入力すると終了）:0
削除する値を入力してください（0を入力すると終了）:6
6は存在しません
削除する値を入力してください（0を入力すると終了）:7
7は存在しません
削除する値を入力してください（0を入力すると終了）:3
入力された値の中に3が見つかりました。ノードを削除します。
削除する値を入力してください（0を入力すると終了）:3
3は存在しません
削除する値を入力してください（0を入力すると終了）:5
入力された値の中に5が見つかりました。ノードを削除します。
削除する値を入力してください（0を入力すると終了）:0
--残っているのは以下の数です--
1　2　4