アルゴリズムとデータ構造

コンピュータサイエンスコース
知能コンピューティングコース

第4回

バケットソート，基数ソート

塩浦昭義

情報科学研究科 准教授

shioura@dais.is.tohoku.ac.jp
http://www.dais.is.tohoku.ac.jp/~shioura/teaching

データ構造
Data Structures

- アルゴリズムの中で，与えられた問題に関連するデータ集合を管理するための道具
- 良いデータ構造とは？
  - データ管理に必要な計算時間が短い
  - シンプル
  - 必要な領域計算量（記憶容量，領域量）が小さい

集合を管理する
Maintenance of Sets

- 整数の集合が与えられている

  4, 5, 8, 2, 9, 1, 3

- ときどき，新しい整数が追加される

  7を追加 ➔ 4, 5, 8, 2, 9, 1, 3, 7

- ときどき，ある整数が削除される

  9 を削除 ➔ 4, 5, 8, 2, 1, 3, 7

- アルゴリズム（プログラム）の中でどのように表現するか？

配列の利用（その1）
Use of Arrays

配列A[1], A[2], …, A[N]を使って表現（N: 十分大きな数）

集合の中に整数 k が　ある➔ A[k] = 1,　ない➔ A[k] = 0

4, 5, 8, 2, 9, 1, 3 ⇒

1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	0	0	1	1	0

集合の中に整数 k が複数存在する場合も対応可能

A[k] = 集合の中に存在する k の個数

4, 1, 8, 2, 8, 1, 3, 1 ⇒

1	2	3	4	5	6	7	8	9	10
3	1	1	1	0	0	0	2	0	0

整数 k を追加 ➔ A[k] を1増やす --- O(1) 時間で可能

整数 k を削除 ➔ A[k] を1減らす --- O(1) 時間で可能

欠点: 整数 N より大きい整数が追加されるとダメ

実際の集合のサイズより大きい配列が必要

無駄な領域計算量

配列の利用（その2）
Use of Arrays

集合に含まれる整数を配列に代入（非負の整数を仮定）

4, 5, 8, 2, 9, 1, 3 →

1	2	3	4	5	6	7	8	9	10
4	5	8	2	9	1	3	-1	-1	-1

空のところには-1を入れる

7 を追加 →

1	2	3	4	5	6	7	8	9	10
4	5	8	2	9	1	3	7	-1	-1

O(1)時間で可能

5 を削除 →

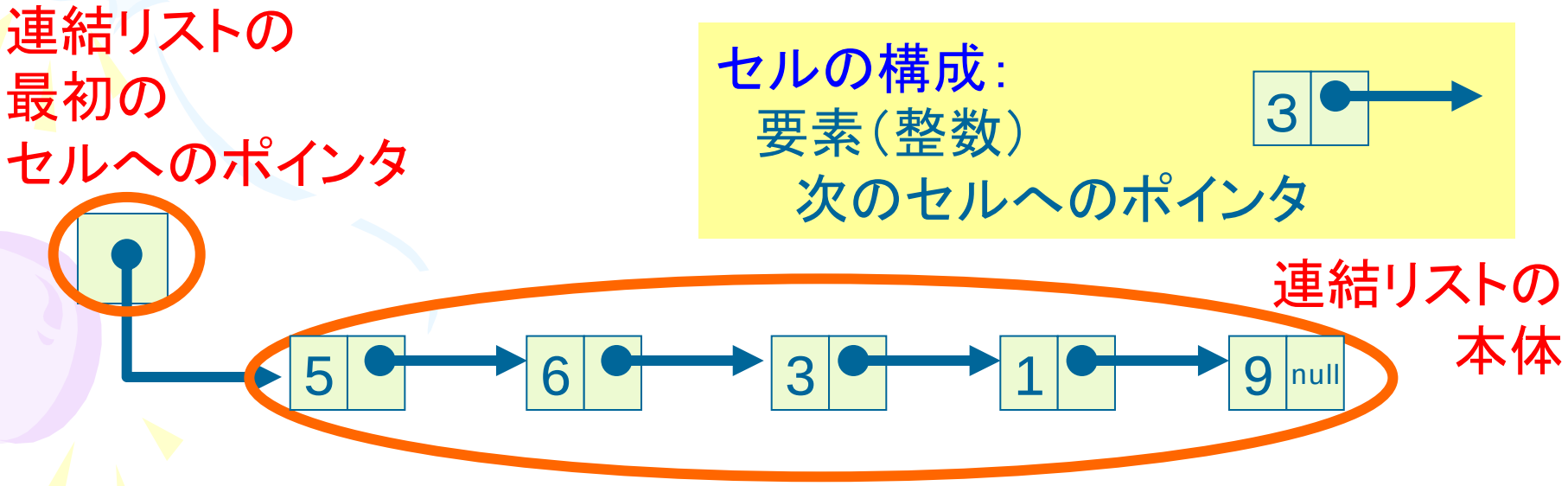
1	2	3	4	5	6	7	8	9	10
4	8	2	9	1	3	7	-1	-1	-1

“5”より後ろの整数を移動させる
➔ O(n)時間（n: 集合のサイズ）

連結リストの利用
Use of Linked Lists

連結リスト:「セル」と呼ばれる基本要素をポインタにより連結したもの

- 必要な領域計算量は集合のサイズに等しい
- 要素の追加はO(1)時間で可能
- 先頭の要素の削除はO(1)時間で可能
- 先頭以外の要素の削除はO(1)時間では不可能

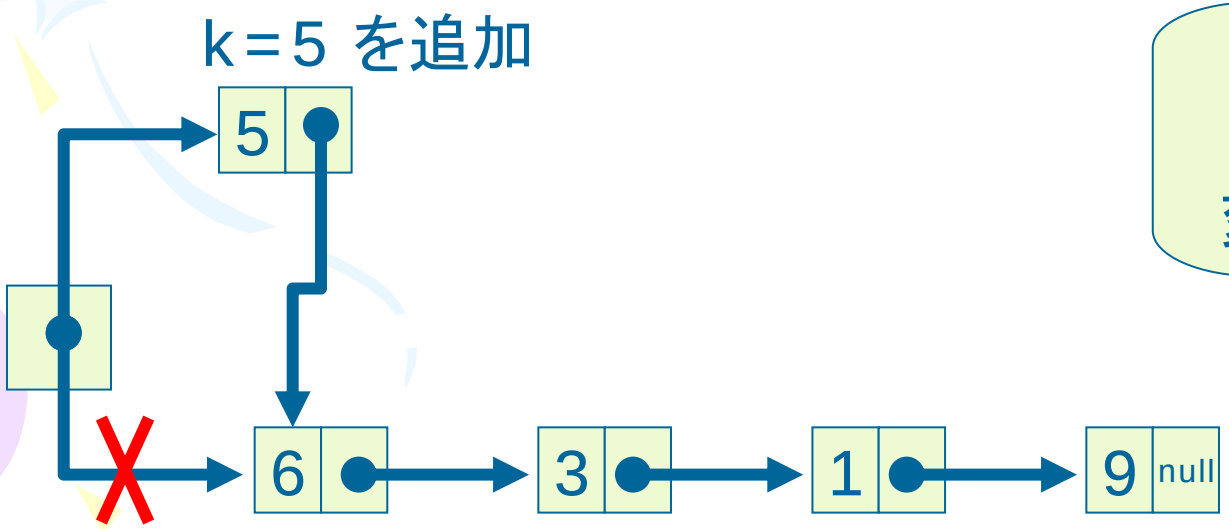


連結リスト:要素の追加
Linked Lists: Addition of Elements

リストの先頭への整数kの追加---O(1) 時間で可能

入力:リスト，追加する整数 k　　出力:新たなセルを追加したリスト

1. 新しいセル C を準備，セルに整数 k と書く
2. Cの次のセルへのポインタを，現在の最初のセルとする
3. 連結リストの最初のセルへのポインタを，Cに変更



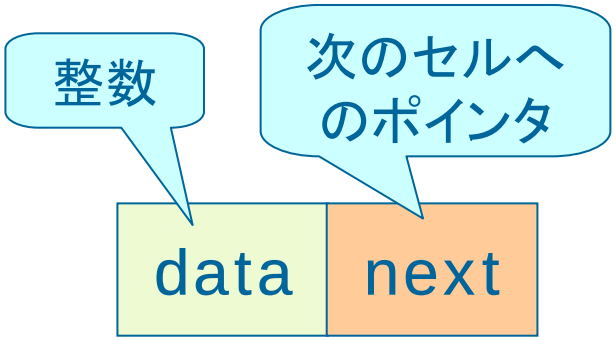
ポインタ変更の
順番を間違えると
変なリストができる

連結リストの実装
Implementation of Linked Lists

C言語での連結リストの実装例

セルを表す構造体の定義

```
struct cell {
  int data;
  struct cell *next;
};
```

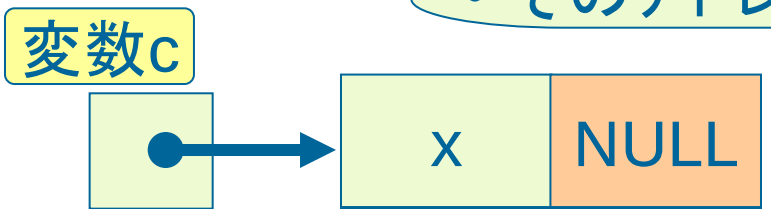


新しいセルを作る関数

```
struct cell *newcell(int x)
{
  struct cell *c;

  c = (struct cell *) malloc(sizeof(struct cell));
  c->data = x;
  c->next = NULL;
  return(c);
}
```

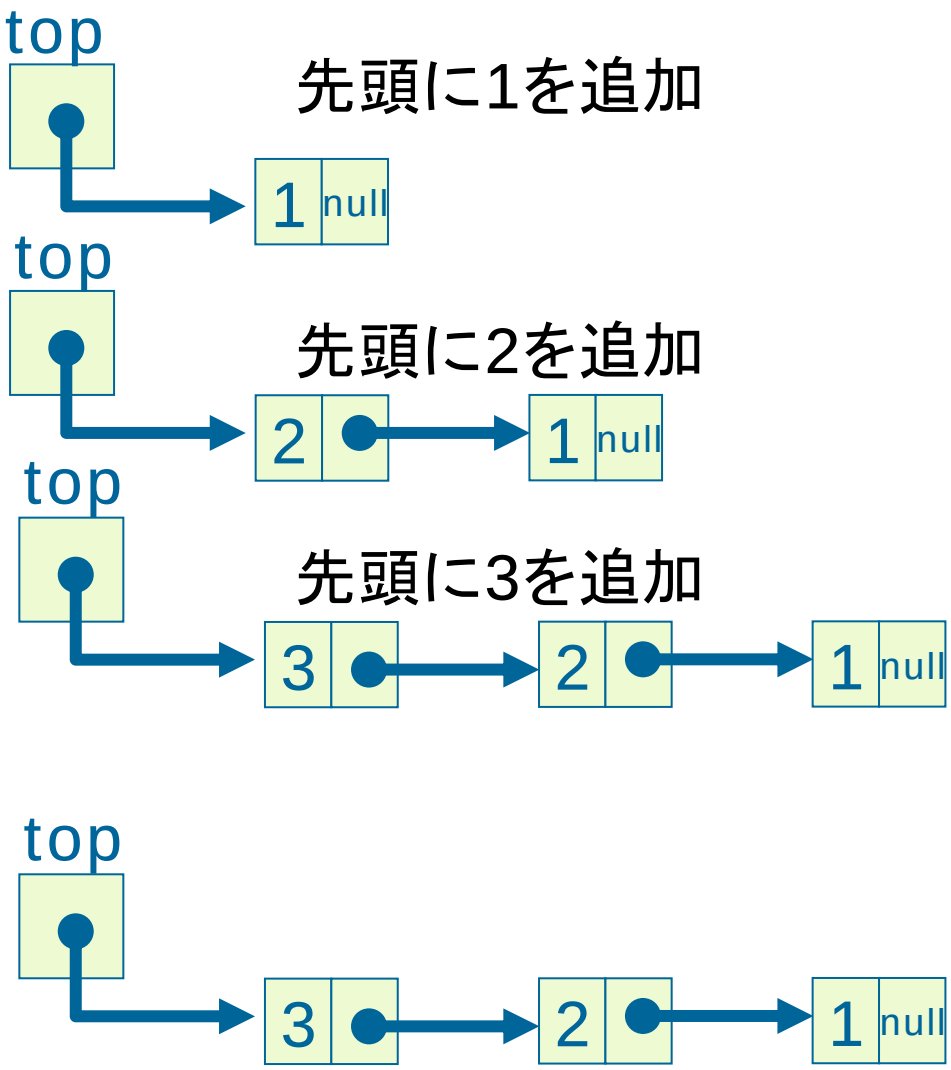
引数 x はセルの要素となる
新しく作ったセルへのポインタを出力

- セルの分の記憶領域を確保
- そのアドレスを c に代入



要素の追加の実装

```
main()
{
  struct cell *top, *cell;

  cell = newcell(1);
  cell->next = NULL; top = cell;

  cell = newcell(2);
  cell->next = top; top = cell;

  cell = newcell(3);
  cell->next = top; top = cell;

  cell = top;
   do {
     printf("%d ", cell->x);
     cell = cell->next;
   } while (cell != NULL);
   printf("¥n");
}
```



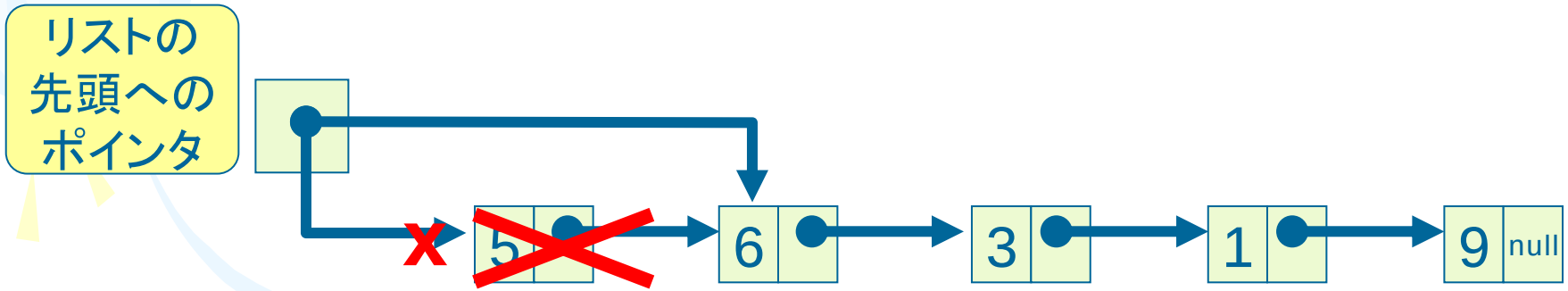
先頭から順番に，各セルの要素を表示

連結リスト:先頭の要素の削除
Linked Lists: Deletion of First Element

先頭にあるセルの削除 --- O(1) 時間で可能

入力:リスト　　出力:先頭のセルを削除したリスト

1. リストの先頭のセルへのポインタを，2番目のセルに変更
2. 元々の先頭のセルを削除

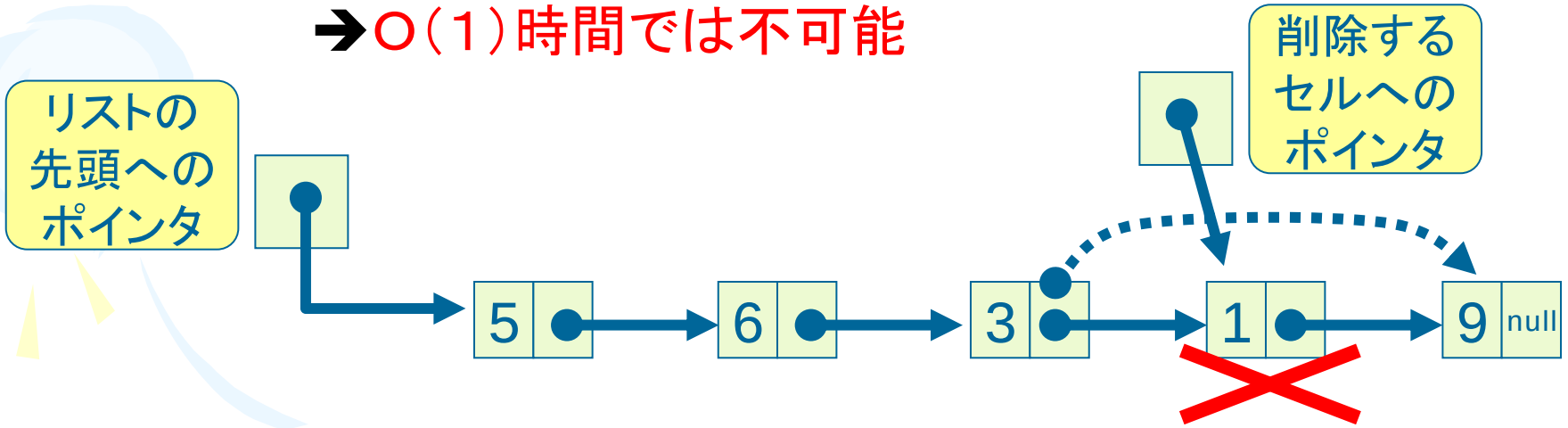


連結リスト：他の要素の削除
Linked Lists: Deletion of Other Elements

先頭以外のセルの削除はなぜ難しい？

- 削除したいセルの前にあるセルがどれか，わからないから
- 前のセルを知るには，リストを先頭から調べる必要有り

➔O(1)時間では不可能



例：1のセルを削除したい

➔1の前のセルから，1の後ろのセルへ，ポインタを修正する必要あり

1の後ろのセルは9とわかる.

1の前のセルは？？？

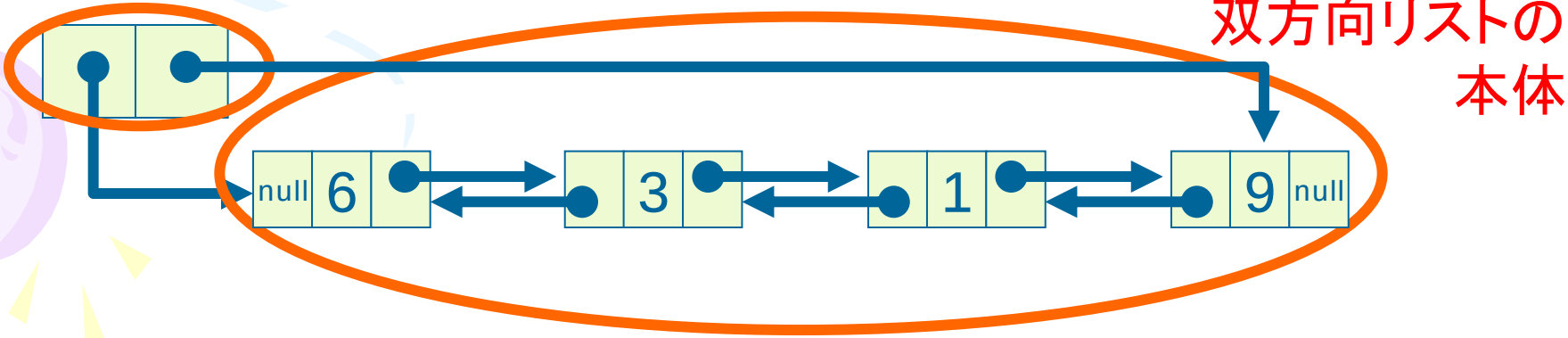
双方向リストの利用
Use of Doubly-Linked Lists

- 双方向リストを利用して集合を表現する
  - 必要な領域計算量は集合のサイズに等しい
  - 整数の追加，削除はO(1)時間で可能

セルの構成：
要素（整数）
前のセルへのポインタ
次のセルへのポインタ

双方向リストの
最初と最後の
セルへのポインタ

双方向リストの
本体

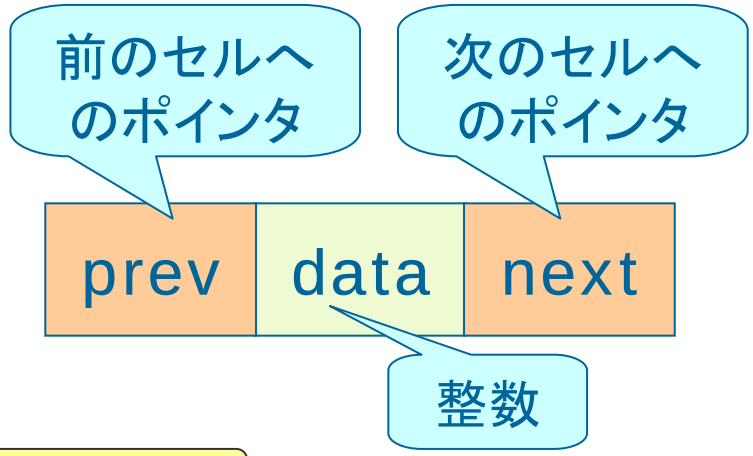


双方向リストの実装
Implementation of Doubly-Linked Lists

C言語での双方向リストの実装例

セルを表す構造体の定義

```
struct cell {
  int data;
  struct cell *prev, *next;
};
```

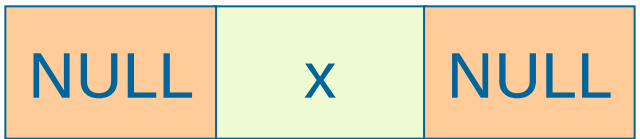


新しいセルを作る関数

引数 x はセルの要素となる
新しく作ったセルへのポインタを出力

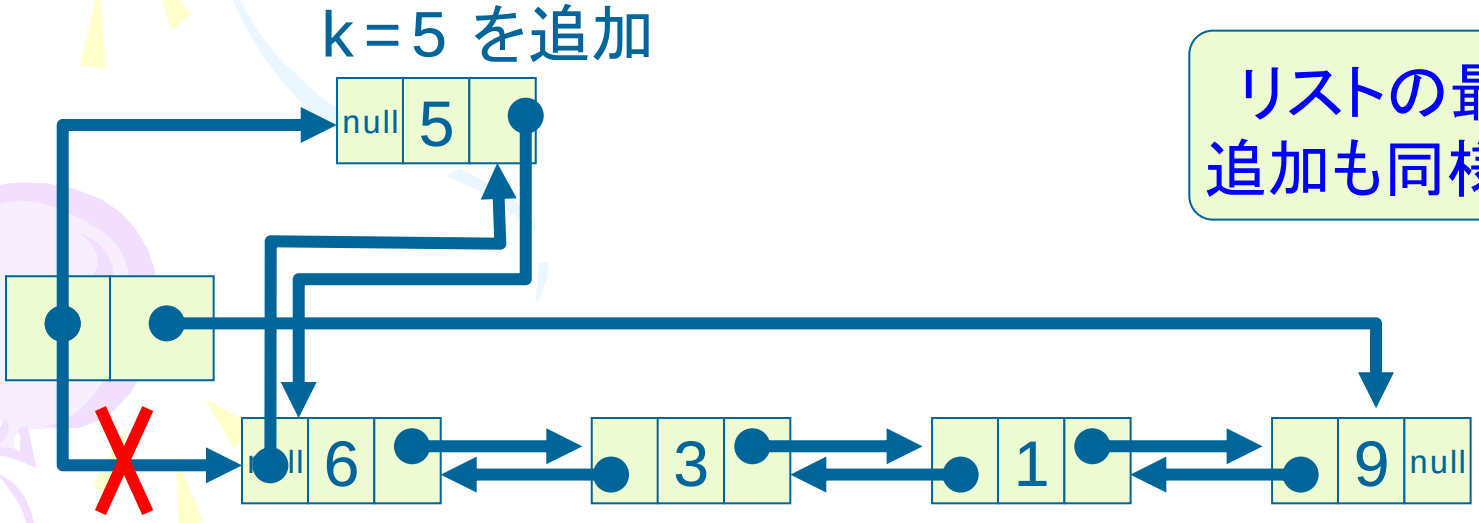
```
struct cell *newcell(int x)
{
  struct cell *c;

  c = (struct cell *) malloc(sizeof(struct cell));
  c->data = x;
  c->prev = NULL;
  c->next = NULL;
  return(c);
}
```

- セルの分の記憶領域を確保
- そのアドレスを c に代入



双方向リスト:要素の追加
Doubly-Linked Lists: Addition of Elements

- リストの先頭への整数kの追加---O(1) 時間で可能
  1. 新しいセル C を準備，セルに整数 k と書く
  2. Cの前のセルへのポインタを，null とする
  3. Cの次のセルへのポインタを，現在の最初のセルとする
  4. 現在の最初のセルの前のセルへのポインタを，Cに変更
  5. 連結リストの最初のセルへのポインタを，Cに変更



リストの最後尾への
追加も同様にして可能

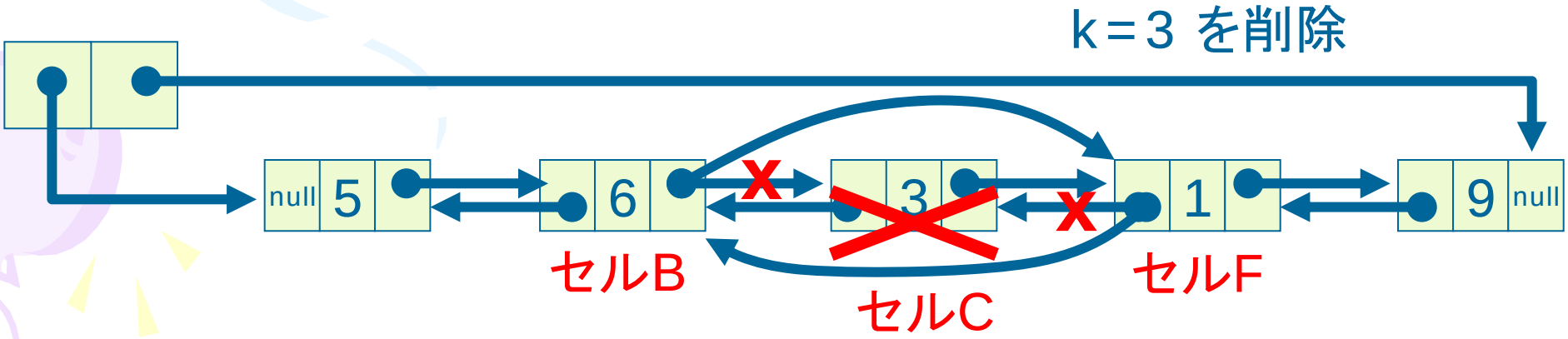
双方向リスト:要素の削除
Doubly-Linked Lists: Deletion of Elements

- 整数 k のセル C の削除(Cへのポインタが既知)

--- O(1) 時間で可能

0. C の前のセルを B, 次のセルを F とする
1. セル B の前のセルを F に変更
2. セル F の次のセルを B に変更
3. セル C を消去

セル C の位置がリストの最初または最後の場合は修正が必要



バケットソート(p89-91)
Bucket Sort

- 整列の対象となっている整数の範囲が事前にわかっている(例:各入力値a_iが$0 \leqq a_i \leqq m-1$を満たす)
- 整数の範囲(mの大きさ)があまり大きくない

バケットソートにより高速に整列が可能
時間計算量 O(m+n)

バケット(bucket)

= バケツ

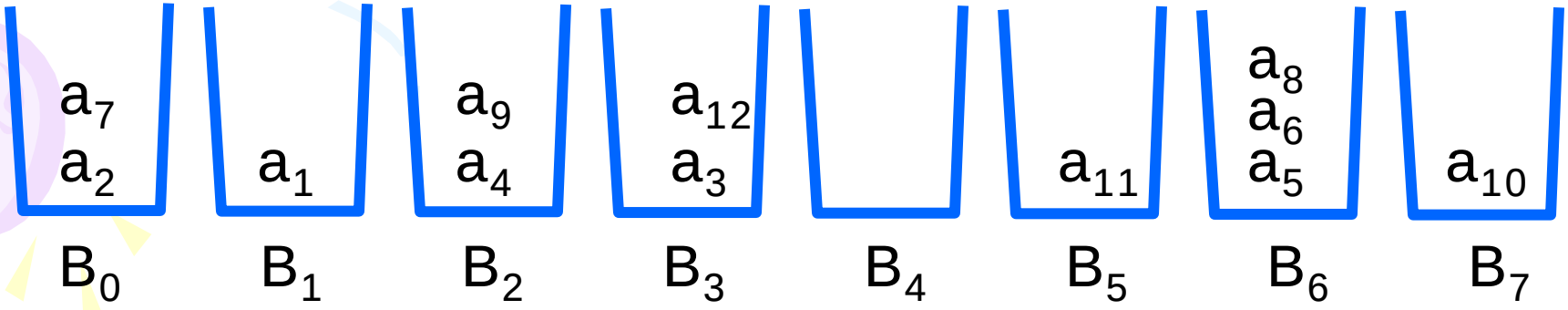
バケットソートのアイディア
Idea of Bucket Sort

- $0 \leqq a_i \leqq m-1$を満たす整数$a_1, a_2, \ldots, a_n$ を整列
- 各 j ($0 \leqq j \leqq m-1$)に対し，空のバケツB_jを用意
- 各 a_i に対し，$a_i = j$ ならば a_i をバケツB_jに入れる
- バケツ $B_0, B_1, \ldots, B_{m-1}$ の中身を（入れた順に）並べる

➔ソートが完了

0以上7以下の整数

1	2	3	4	5	6	7	8	9	10	11	12
1	0	3	2	6	6	0	6	2	7	5	3



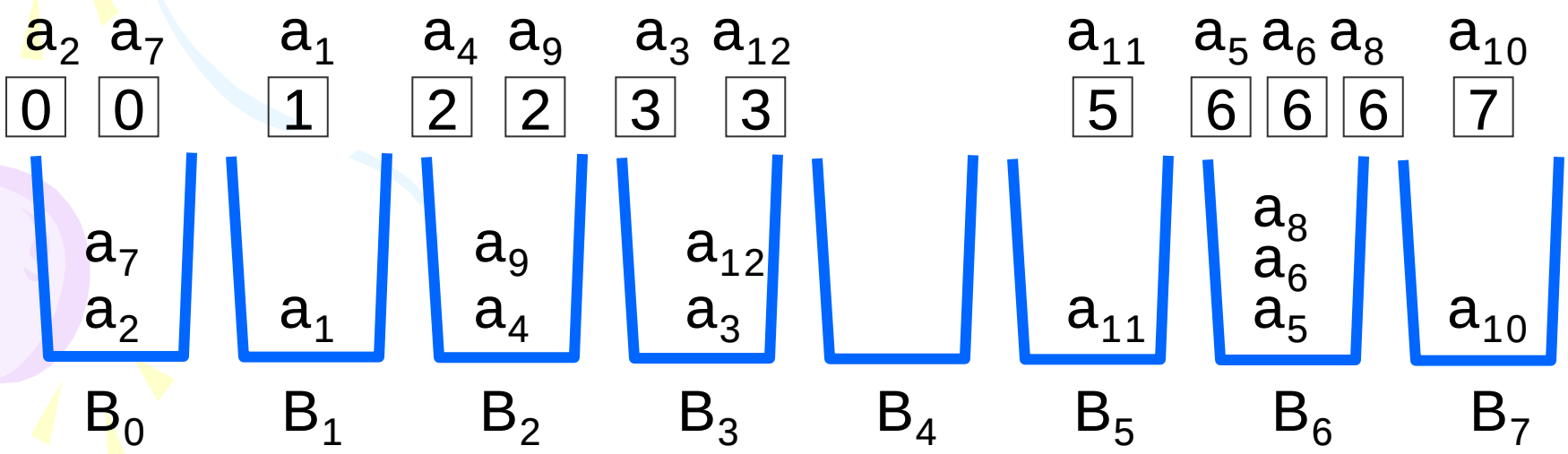
バケットソートのアイディア
Idea of Bucket Sort

- バケツ B_0, B_1, …, B_{m-1} の中身を（入れた順に）並べる

➔ソートが完了

同じバケツの中の要素は
元の順番通りに並べられている

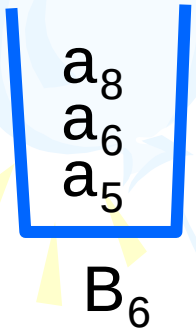
1	2	3	4	5	6	7	8	9	10	11	12
1	0	3	2	6	6	0	6	2	7	5	3

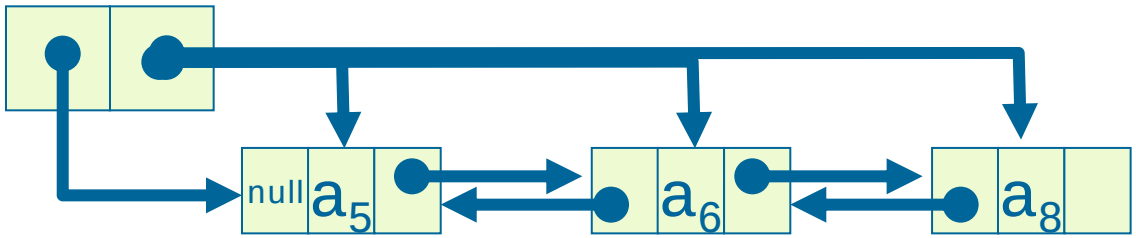


バケットソートの実現
Implementation of Bucket Sort

- 各バケツは双方向リストで実現
- 挿入するときは最後尾に
- 削除するときは先頭から

} 同じバケツの要素は元の順番通りに並ぶ





バケットソートの計算時間
Time Complexity of Bucket Sort

- m個のバケツを準備➔O(m)時間
- n個の要素をバケツに挿入➔O(1)xn＝O(n)
- バケツに入れた要素を取り出す➔O(m＋n)時間

∴ 全体でO(m ＋ n) 時間

空間計算量もO(m ＋ n)

※ 整数の範囲が狭いときには有効

m が大きいときは時間計算量も空間計算量も膨大

基数ソート:カードを並べる方法(p91-94)
How to Sort Cards: Radix Sort

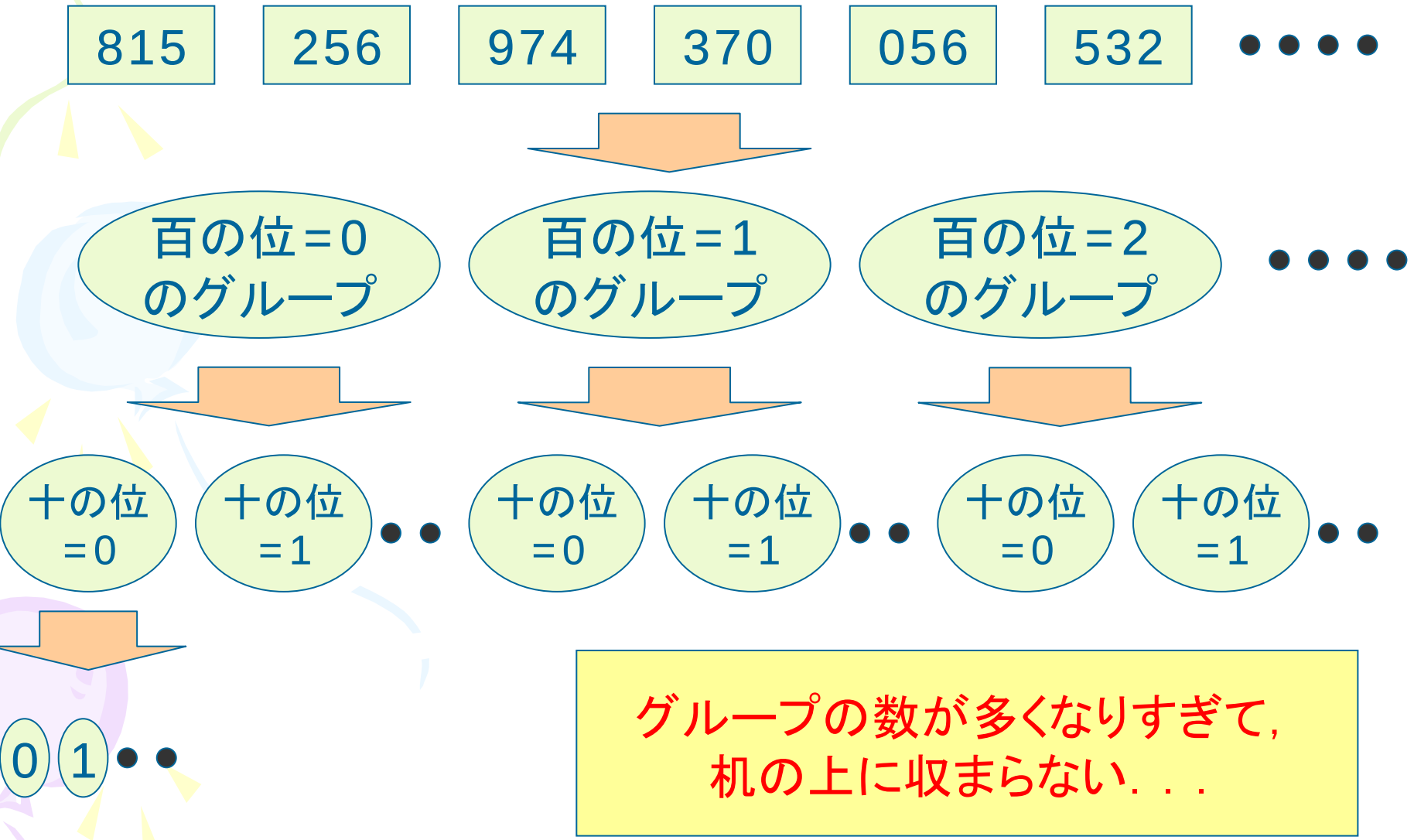
- 例:3桁の数字からなる学籍番号の書かれたカードを番号順に(机の上で)並べたい

815	256	974	370	056	532	●●●●

- よく使われる方法:
  - まず百の位の値によってグループ分け
  - 次に各グループを十の位によってグループ分け
  - 最後に各グループを一の位によってソート
  - ソートされたカードをまとめる

机の上で実現可能か?

カードを並べる
Sorting Cards



基数ソートのアイディア
Idea of Radix Sort

- よく使われる方法:
  - まず百の位の値によってグループ分け
  - 次に各グループを十の位によってグループ分け
  - 最後に各グループを一の位によってソート
  - ソートされたカードをまとめる

- 基数ソートの手順:
  - まず一の位の値によってバケットソート
  - 次に十の位によってバケットソート
  - 最後に百の位によってバケットソート

なぜこの方法で
ソートが
できるのか？

基数ソートの例
Example of Radix Sort

- 簡単のため，各桁の数字は0,1,2,3のみとする

123	013	322	102	021	311	222	110	200

まず一の位の値によってバケットソート

110	200	021	311	322	102	222	123	013

次に各グループを十の位によってバケットソート

200	102	110	311	013	021	322	222	123

バケットソートを利用
➔十の位が同じ数字ならば，元の順番（一の位に関する昇順）通りに並ぶ
➔下2桁に関して昇順に並んでいる

基数ソートの例
Example of Radix Sort

次に各グループを十の位によってバケットソート

200	102	110	311	013	021	322	222	123

下2桁に関して昇順に並んでいる

最後に各グループを百の位によってバケットソート

013	021	102	110	123	200	222	311	322

バケットソートを利用

➔百の位が同じ数字ならば，元の順番（下2桁に関する昇順）通りに並ぶ

➔下3桁に関して昇順に並んでいる

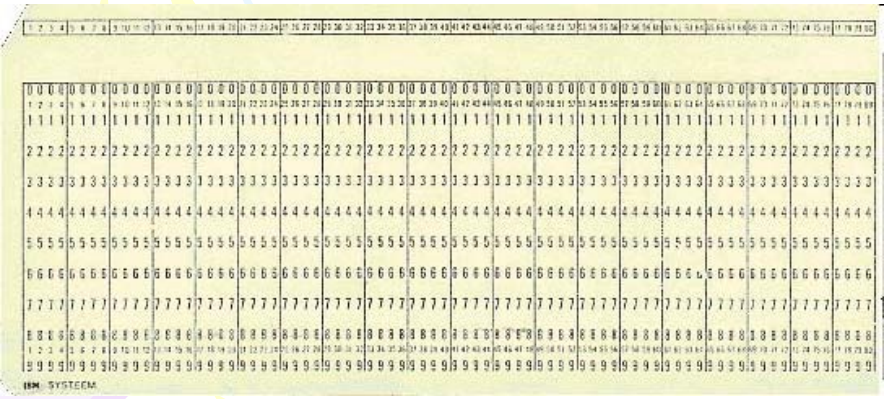
基数ソートの計算時間
Time Complexity of Radix Sort

- 桁数が K 以下の n 個の（非負）整数をソートする場合
  - バケットソートを K 回実行
  - 一回のバケットソートでは10個(0,1,2,..,9)個のバケツを使用➔ O(10 + n) =O(n)時間
  - 全体では O(n) x K = O(Kn)時間
- K 桁以下の m 進数をソートする場合は O(K(m + n)) 時間

- K文字以下のアルファベットで書かれた名前・単語のソートも可能
  - 文字の種類が m，名前・単語の数が n ならば O(K(m+n)) 時間

実際に使われていた基数ソート
Radix Sort Used in Real Life

- 基数ソートは実際にカードを機械を使ってソートするときに使われていた
- カードには，各桁ごとに対応する数字のところに穴を開ける
- 基数ソートをするときには，各桁ごとに0,1,2,…,9に対応する穴を参照する



http://www.museumwaalsdorp.nl/computer/en/punchcards.html

http://smoto.mii.kurume-u.ac.jp/~smoto/edu/ala/history/p0.html

レポート(プログラム作成)

連結リストを用いた下記のプログラムを作成しなさい.

問1:キーボードから入力された数字を，次々に連結リストに追加していくプログラムを作成しなさい．ただし，入力される数字は正の整数とし，0が入力されたらプログラムは終了するものとする.

問2:「3，2，1」という数字が入っている連結リストのセルの順番を，逆順に並び替えなさい．スライド「要素の追加の実装」に書いてあるプログラムを修正してください.

問3:連結リストを適当に作った後，リストの中のセルを数字の小さい順に並び替えるプログラムを作成しなさい.

締切:5月27日(木)午後7時まで

レポート(紙で提出)
締切:5/20 AM9:00

問1:以下の英単語にして基数ソートを適用せよ.

COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX

なお, 結果だけでなく, ソートの途中の計算過程についても(授業で行なったように)きちんと書くこと.

問2:双方向リストの最後尾に要素を追加することはO(1)時間で出来る. 一方, 連結リストの最後尾に要素を追加するにはどうすればよいか？その方法と計算時間を説明せよ.

レポート(紙で提出)
締切:5/20 AM9:00

問3:正しくない命題「1+2+…+(n-1)+n=O(n)」に対する以下の証明のうち，どこが間違っているか説明しなさい.

証明:

n に関する数学的帰納法により証明する.

まず，n=1 のときは 1 = O(1) なので成立する.

次に，n=k で成り立つと仮定して， n=k+1のときに成り立つことを証明する.

仮定より， 1+2+…+(n-1)=O(n-1)

したがって，1+2+…+(n-1)+n = O(n-1)+n=O(n)

以上より，命題が証明された(証明終わり)