

# 第6回のキーワード

1

## アルゴリズム関係

- 文字列探索  
(string search/matching)
- 力まかせ法 (brute force)
- $O(n) \sim O(nm)$
- ボイヤー・ムーア (BM) 法  
(Boyer-Moore, Boyer-Moore-Horspool (簡易版))
- $O(n/m) \sim O(nm)$
- 動的配列
- コレクション

## Java関係

- charAt
- Arrays.sort  
Arrays.binarySearch
- ジェネリクス (総称型)
- 自然な順序
- Comparable<E>
- Comparator<E>
- ArrayList<E>
- ラッパークラス
- Collections.sort  
Collections.binarySearch

# 文字列探索

2

## □ 文字列探索とは

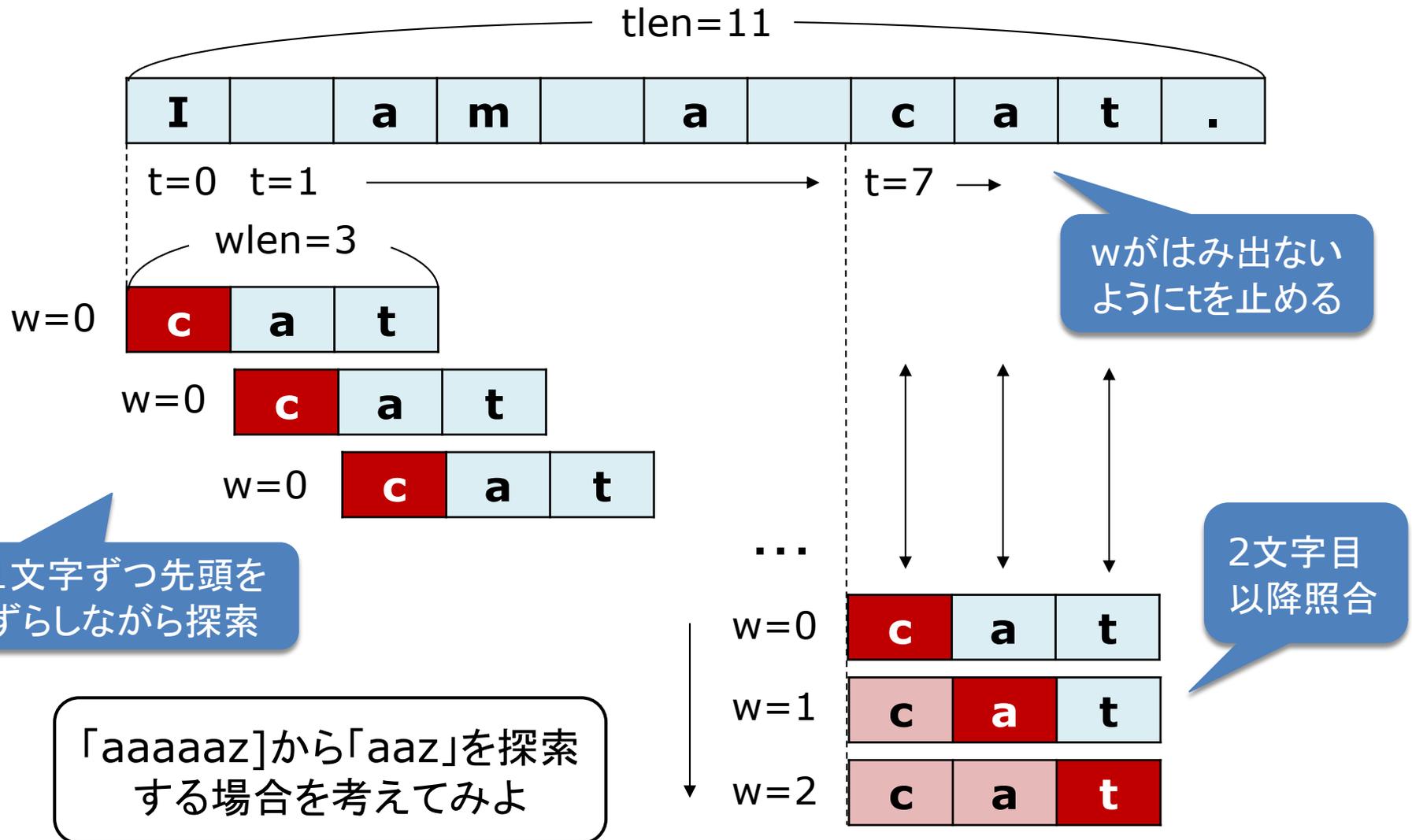
- 前回までは、データ列から要素1個を探す問題を扱った
- 今回は、文字「列」の中から、文字「列」を探す
- 文字列だけでなく、DNA配列の探索などにも応用される

## □ 力まかせ法

- 探索対象のテキストを $n$ 文字、探索文字列を $m$ 文字とする
- 最初は $t=0$ として、テキストの中の位置 $t$ から位置 $t+m$ まで、1文字ずつ順に探索文字列と照合する
- もし、 $m$ 文字すべてが一致したら、位置 $t$ で発見となる
- そうでなければ、 $t$ を1だけ進めて同様の処理を繰り返す
- ただし、テキストの残りが $m$ 文字未満になったら終了とする

# 文字列探索(力まかせ法)

3



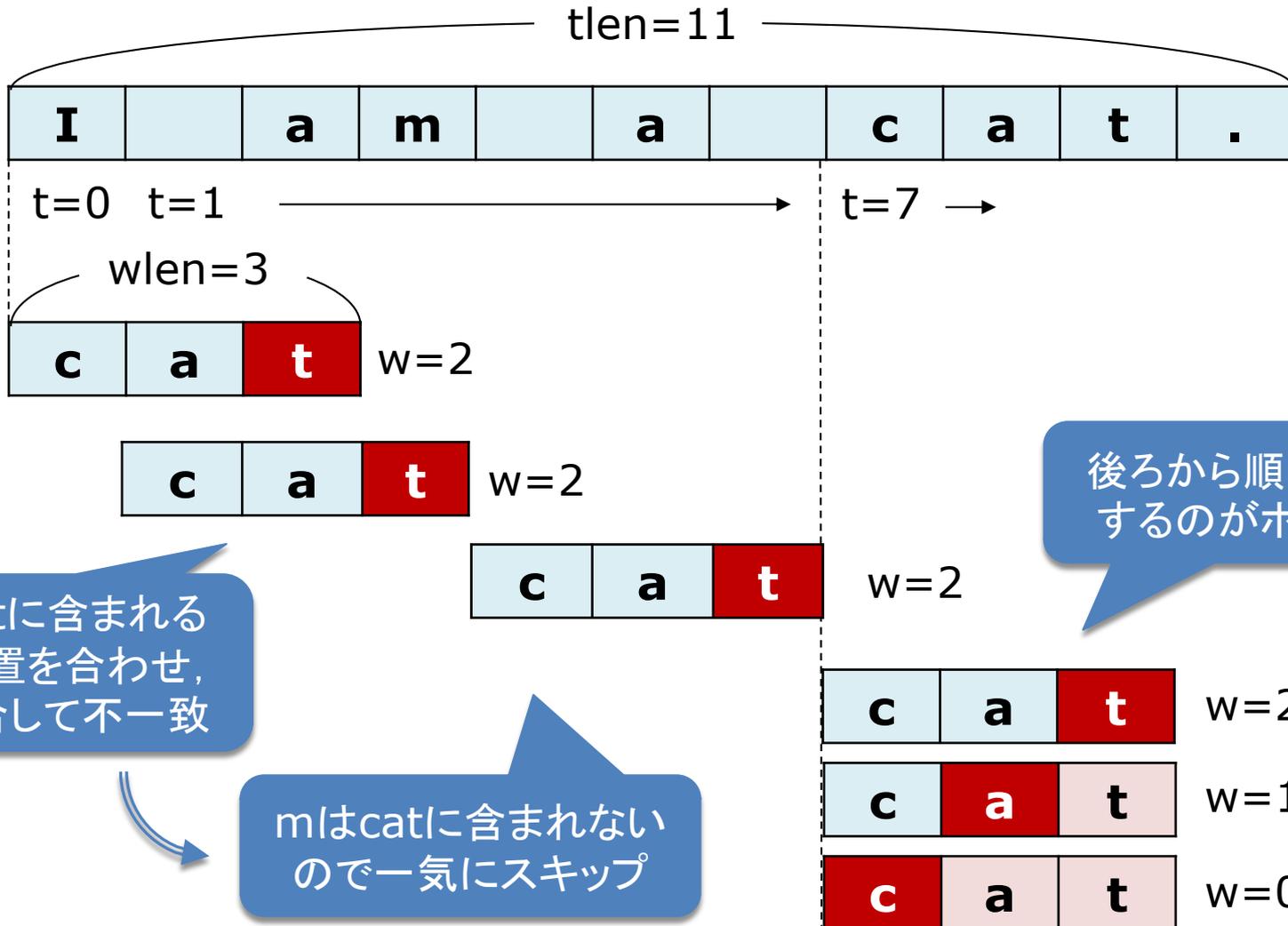
# ボイヤー・ムーア法(簡易版)

4

- 基本的なアイデア
  - 文字列の先頭から照合するよりも、末尾から照合した方が、探索位置を大きくスキップできる
  
- アルゴリズムの概要
  - 対象のテキストを $n$ 文字、探索する文字列を $m$ 文字とする
  - 最初は $t=0$ として、テキストの中の位置 $t+m$ と探索文字列の末尾の位置 $m$ から、1文字ずつ逆順に照合する
  - もし、 $m$ 文字すべてが一致したら、位置 $t$ で発見となる
  - そうでなければ、テキスト側の不一致文字が探索文字列に含まれない場合は、 $t$ を $m$ だけ進め、探索を続ける
  - テキスト側の不一致文字が探索文字列に含まれる場合は、その文字同士が合うように $t$ を適切に進め、探索を続ける

# ボイヤー・ムーア法 (簡易版)

5



後ろから順に比較  
するのがポイント

aはcatに含まれる  
ので位置を合わせ、  
tを照合して不一致

mはcatに含まれない  
ので一気にスキップ

# 文字列探索の計算量の概算

6

## □ カマかせ法

- 最善の場合：テキストの中に、探索文字列の先頭文字が1回以下しか含まれない ⇒ 線形探索と同じなので  $O(n)$
- 最悪の場合：tを進めるごとに、探索文字列の末尾直前まで照合する ⇒ 2重ループをほぼ全て回るので  $O(nm)$

## □ ボイアー・ムーア法

- 最善の場合：tを進めるごとに、探索文字列の末尾文字と不一致で、探索文字列全体を照合するのは1回だけ ⇒ n文字の中でm文字ずつスキップするので  $O(n/m)$
- 最悪の場合：tを進めるごとに、探索文字列の先頭直前まで照合する ⇒ 2重ループをほぼ全て回るので  $O(nm)$

# クラス型の配列（復習）

7

- クラス型の配列の作成
  - ▣ `class Item { int code; String name; }`
  - ▣ `Item [] data = new Item[10];`
  - ▣ `for (int i = 0; i < data.length; i++) data[i] = new Item();`
  
- 配列要素のメンバのアクセス
  - ▣ `if (data[i].code == code)`
  
- 配列要素の（位置の）交換
  - ▣ `Item t; t = data[i]; data[i] = data[j]; data[j] = t;`
  - ▣ Javaの配列の構造やクラス型変数の代入について再確認

# Javaによる探索とソート

8

## □ 配列の探索とソート

- `java.util.Arrays`の静的メソッドが使用できる
- 線形探索: `Arrays.asList(array).indexOf(key)`
  - ただし, これはクラスの配列でしかうまく動かない
  - `int`や`double`など基本型の配列では期待通りに動作しないので注意
- 2分探索: `Arrays.binarySearch(array, key)`
- ソート: `Arrays.sort(array)`

## □ Comparableインタフェース

- 2分探索やソートでは, 要素が比較できなければならない
- そのためには, 要素のクラスはComparableインタフェースを実装し, `compareTo`メソッドを持つことが必要

# 動的配列とジェネリクス

9

## □ ArrayList<E>

- 要素数を動的に変更できる配列(のようなクラス)
- *E*に要素のクラス名を当てはめて使う(ジェネリクス)

例) `ArrayList<String> alist = new ArrayList<String>();`

- 要素の追加/取得/変更には, `add/get/set`メソッドを使う
- 例) `alist.add(str)` / `alist.get(i)` / `alist.set(i, str)`

## □ ArrayListの探索とソート

- `java.util.Collections`の静的メソッドが使用できる
- 線形探索: `alist.indexOf(key)`
- 2分探索: `Collections.binarySearch(alist, key)`
- ソート: `Collections.sort(alist)`

# ラッパークラス

10

- intやdouble(基本型)はクラスとして使えない
  - ▣ 昔, Javaの開発時にパフォーマンスを重視したためだが...
  - ▣ ArrayListなどのコレクションに入れられなくて不便
  
- Integer, Doubleなどの対応するクラスが使える
  - ▣ ラッパークラス(=包むクラス)と呼ばれる
  - ▣ 基本型の値をクラスのインスタンスとして使えるようにする
  - ▣ 例) `Double x = new Double(1.4142);`
  
- 自動的に基本型と相互変換(autoboxing, unboxing)
  - ▣ ほぼintやdoubleと同じように使える(四則演算も可能)
  - ▣ 例) `Integer i = 10; i += 20; System.out.println(i);`