

アルゴリズムとデータ構造

第6回 文字列探索とジェネリクス

第6回のキーワード

2

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
- ジェネリクス (総称型)
- コレクション

Java関係

- charAt
- indexOf, lastIndexOf
- ArrayList<E>
- ラッパークラス
- Arrays.sort
Arrays.binarySearch
- Collections.sort
Collections.binarySearch
- 自然な順序
- Comparable<E>
- Comparator<E>

文字列探索

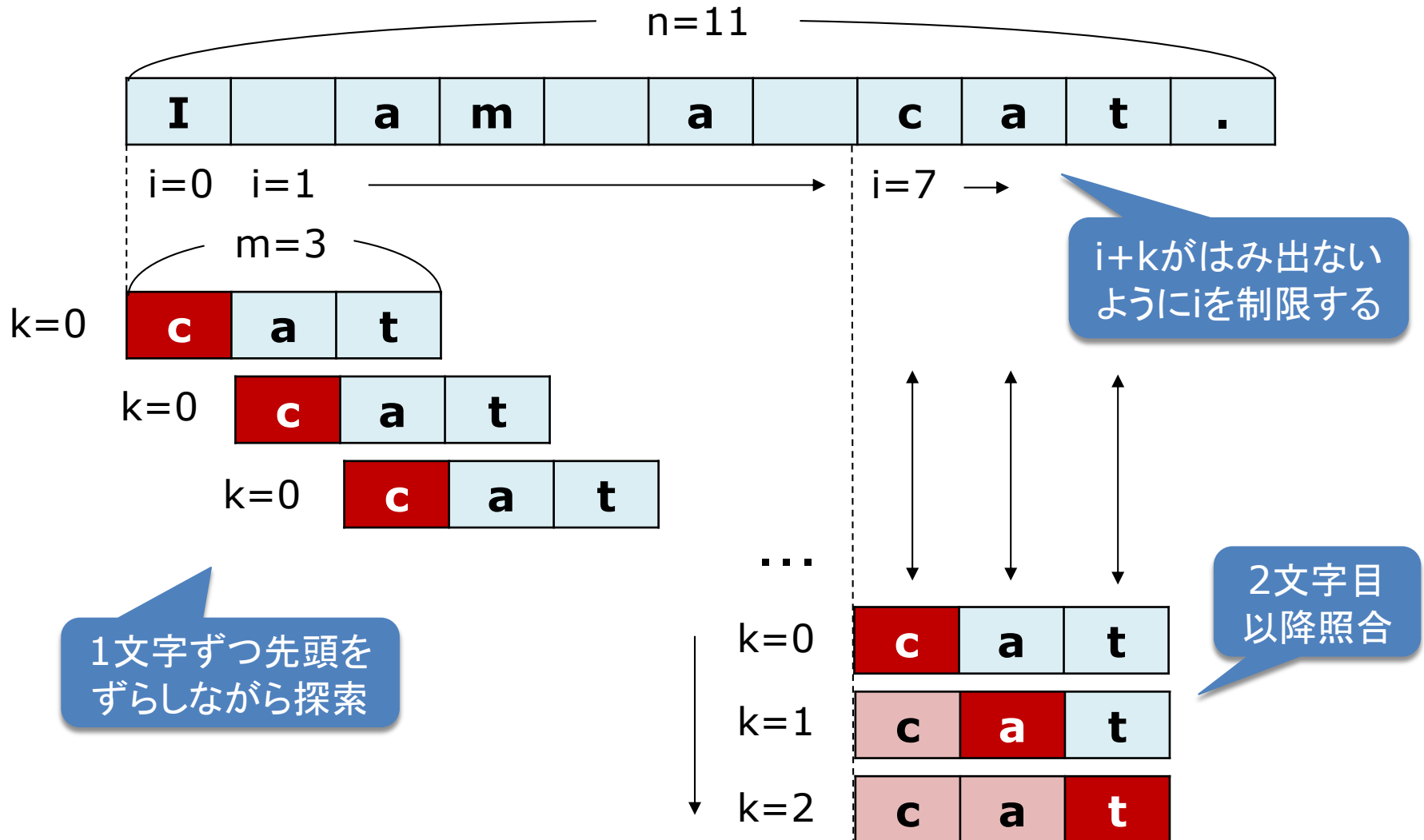
3

- 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

- カマかせ法
 - ▣ 対象のテキストを n 文字、探索文字列を m 文字とする
 - ▣ まず $i = 0$ として、テキスト内の位置 i から1文字ずつ順に m 文字分を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 i で発見となる
 - ▣ そうでなければ、 i 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 文字未満なら終了とする

文字列探索(力まかせ法)

4



確認問題

5

- 力まかせ法
 - ▣ 「accent」 から「cent」を探索する場合について考える
 - ▣ n と m を示せ
 - ▣ 探索中の i と k の値の組み合わせ (i, k) を全て順に示せ

- 文字列の中の文字の比較
 - ▣ 文字列 text の中の位置 i から始まる m 文字が、文字列 key の最後まで一致しているか調べる処理を完成させよ

```
int m = key.length();
int k; // kはkeyの中の位置
for (k = 0; k <          ; k++) {
    if (text.charAt(          ) != key.charAt(          ))
        break; // 不一致
}
```

ボイヤー・ムーア法(簡易版)

6

□ 基本的なアイデア

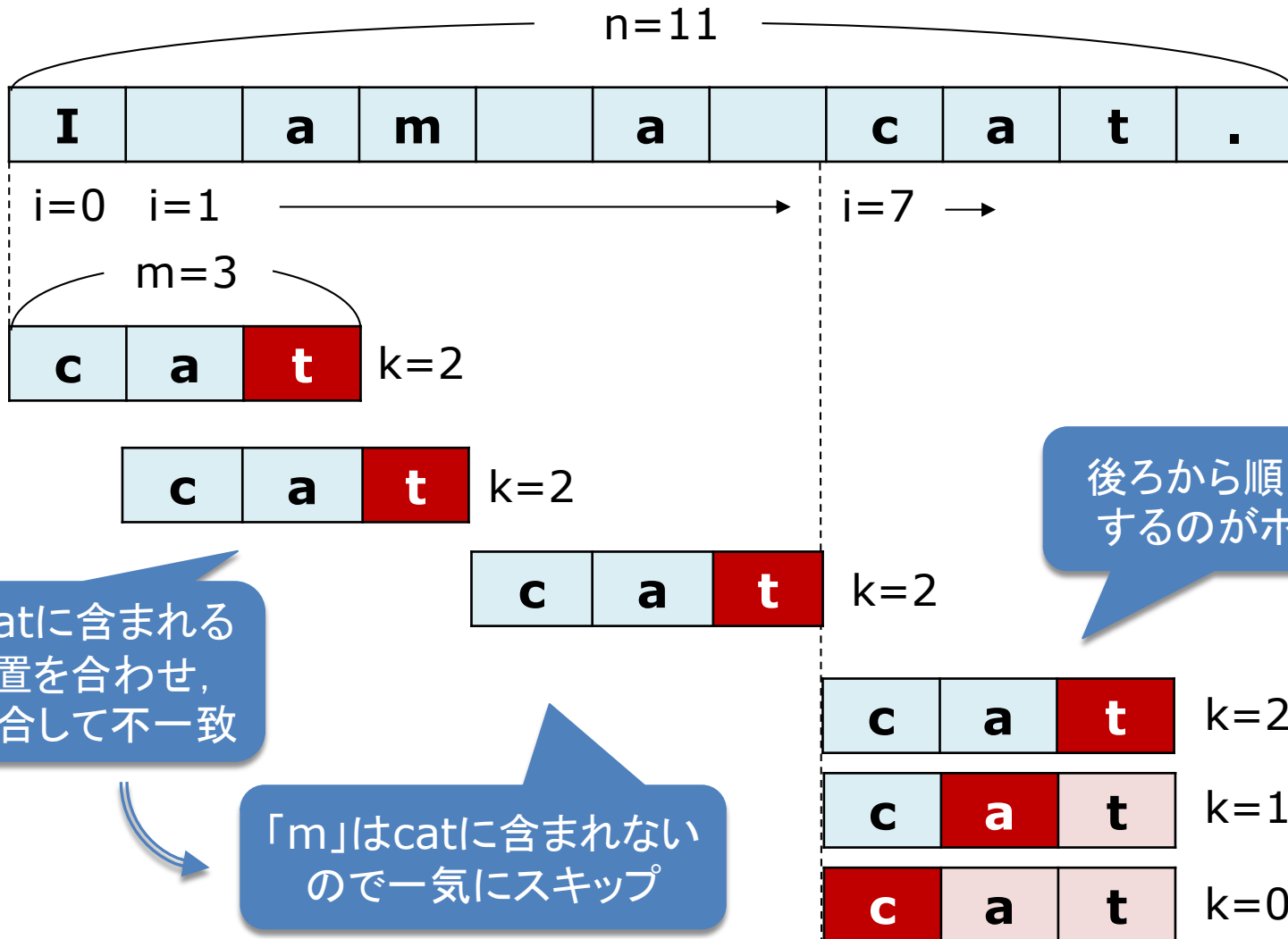
- 文字列の先頭から照合するよりも、末尾から照合した方が、探索位置を大きくスキップできる

□ アルゴリズムの概要

- 対象のテキストを n 文字, 探索文字列を m 文字とする
- まず $i = 0$ として, テキストの位置 $i + (m - 1)$ の文字と探索文字列の末尾の文字から, 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 i で発見となる
- 一致しなかったら, テキストの位置 $i + (m - 1)$ の文字が探索文字列に含まれない場合は, i を m だけ進め, 探索を続ける
- 含まれる場合は, テキストの位置 $i + (m - 1)$ に探索文字列のその文字を合わせるように i を最小限だけ進め, 探索を続ける

ボイヤー・ムーア法 (簡易版)

7



文字列探索の計算量の概算

8

□ 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 \Rightarrow 線形探索と同じなので $O(n)$
- 最悪の場合: i を進めるごとに, 探索文字列の末尾直前まで照合する \Rightarrow 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: i を進めて照合すると毎回完全に不一致で, 探索文字列全体を照合するのは1回だけ
 $\Rightarrow n$ 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: i を進めるごとに, 探索文字列の先頭直前まで照合する \Rightarrow 2重ループをほぼ全て回るので $O(nm)$

Javaの機能による文字列操作

9

□ 文字列の探索

- ▣ `str.charAt(pos)` `str`の中の位置`pos`の文字を取得
- ▣ `str.contains(str2)` `str`に`str2`が含まれていれば真
- ▣ `str.indexOf(str2)` `str`の先頭から`str2`を探索
- ▣ `str.indexOf(str2, pos)` `str`の位置`pos`以降で`str2`を探索
- ▣ `str.lastIndexOf(str2)` `str1`の末尾から`str2`を探索

□ 文字列の構築

- ▣ Stringクラスによる文字列操作(+演算子による連結等)は、文字列を複製して作り直しているなので効率が悪い
- ▣ 文字列を組み立てるときは、`String.format`メソッドを使うか、`StringBuilder`(または`StringBuffer`)クラスを使う

クラス型の配列（復習）

10

- 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

確認問題

11

□ Javaの配列とfor文

- 下記のA)やB)のような記述はできるが、C)のような記述はできない理由を述べよ

A)

```
for (int i = 0; i < data.length; i++)  
    data[i] = new Item();
```

B)

```
for (Item e : data)  
    System.out.println(e.code);
```

C)

```
for (Item e : data)  
    e = new Item();
```

□ Javaの配列とコピー

- クラスのインスタンスを要素とする配列 data について、下記のA)とB)の処理の違いを図解で説明せよ

A)

```
data[i] = data[j];
```

B)

```
data[i].code = data[j].code;  
data[i].name = data[j].name;
```

中間試験の範囲
はここまで

動的配列とジェネリクス

12

- ArrayList<E>クラス
 - ▣ 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(ジェネリクスという)
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)`

- 要素(E)はクラス型のみ
 - ▣ 基本型の代わりに, 対応する「ラッパークラス」を使う
 - ▣ `int → Integer, double → Double, char → Character`
 - ▣ 例) `ArrayList<Double> alist = new ArrayList<Double>();`

Javaの機能による探索とソート

13

- 配列の探索とソート (`java.util.Arrays`)
 - ▣ 2分探索: `Arrays.binarySearch(array, key)`
 - ▣ ソート: `Arrays.sort(array)`

- `ArrayList`の探索とソート (`java.util.Collections`)
 - ▣ 線形探索: `alist.indexOf(key)`
 - ▣ 2分探索: `Collections.binarySearch(alist, key)`
 - ▣ ソート: `Collections.sort(alist)`

- `Comparable` (比較可能) 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
 - ▣ 要素は `java.util.Comparable` インタフェースの実装が必要