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

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

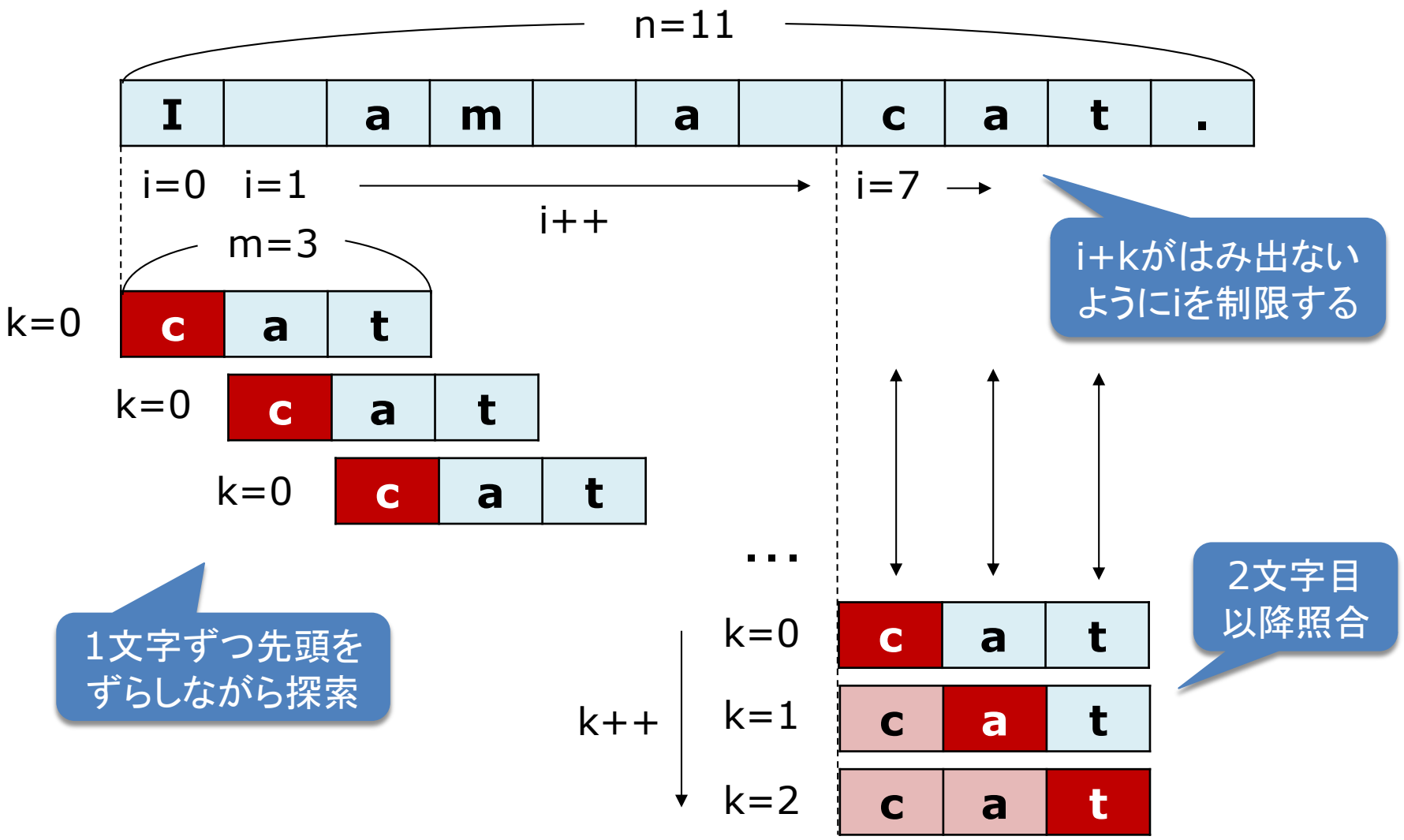
- charAt
- indexOf, lastIndexOf
- ArrayList<*E*>
- ラッパークラス
- Arrays.sort
  Arrays.binarySearch
- Collections.sort
  Collections.binarySearch
- 自然な順序
- Comparable<*E*>
- Comparator<*E*>

文字列探索

- 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
  - 対象のテキストを n 文字，探索文字列を m 文字とする
  - i の初期値を 0 として，テキストの位置 i の文字から順に探索文字列と照合する（i の位置から最大 m 文字照合）
  - もし，m 文字全部が一致したら，位置 i で発見とする
  - 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）


n=11
I a m a c a t .
i=0 i=1
i++
i=7
m=3
k=0 c a t
k=0 c a t
k=0 c a t
i+kがはみ出ないようにiを制限する
1文字ずつ先頭をずらしながら探索
...
k=0 c a t
k++
k=1 c a t
k=2 c a t
2文字目以降照合

確認問題

- 力まかせ法
  - 「address」から「dress」を探索する場合について考える
  - n と m を示せ
  - 文字の照合の組み合わせと i の値の変化を順に示せ

- 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

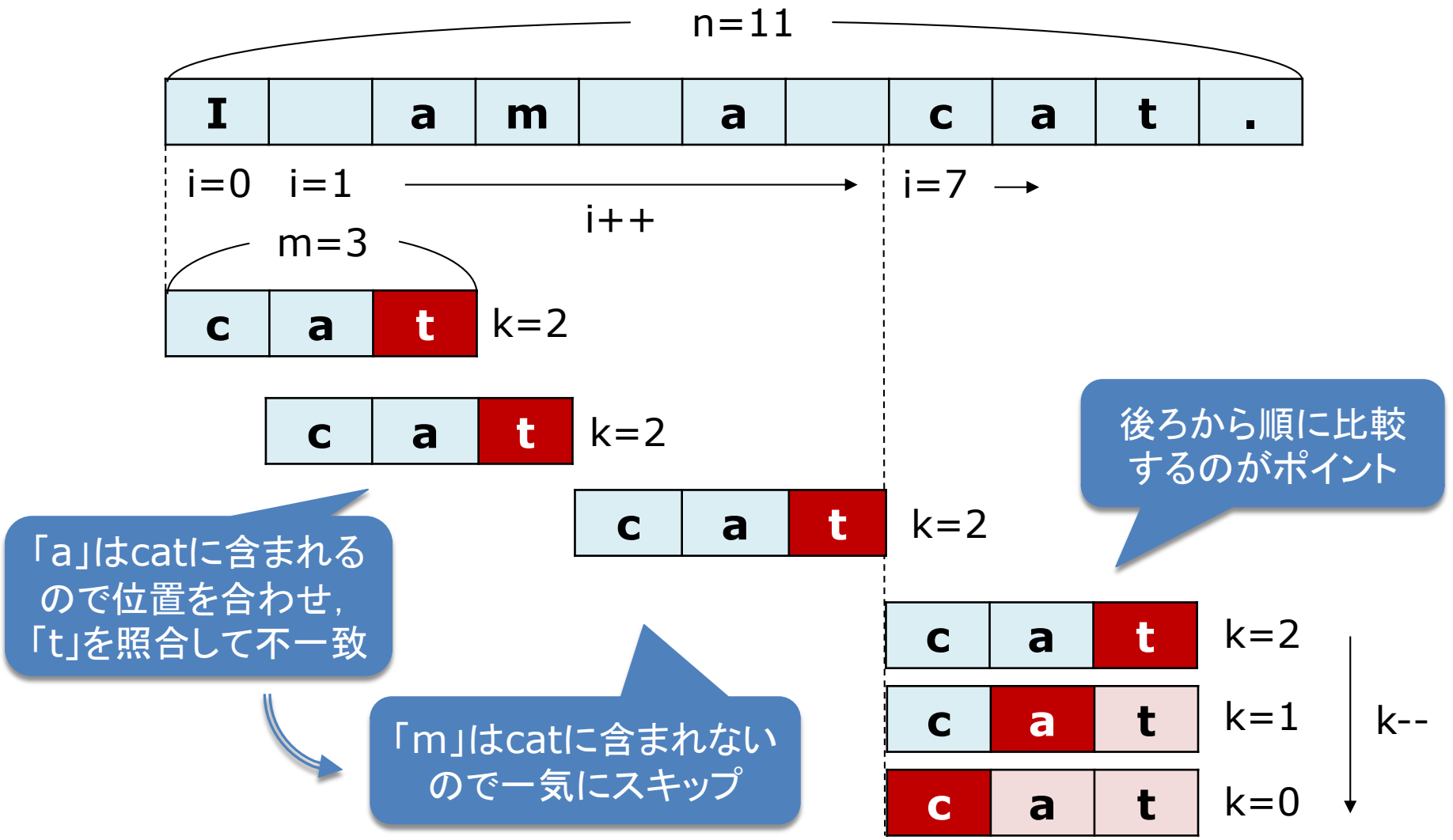
```
int m = key.length();
int k;  // kはkeyの中の位置
for (k = 0; k <          ; k++) {
    if (text.charAt(          ) != key.charAt(          ))
        break;  // 不一致
}
```

ボイヤー・ムーア法（簡易版）

- 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
  - 事前の準備で，探索文字列に含まれる文字を把握しておく

- 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
  - i の初期値を 0 として，テキストの位置 i + (m - 1) から位置 i の文字まで，探索文字列の末尾の文字から逆順に照合していく
  - もし，m 文字全部が一致したら，発見位置を i として終了する
  - 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）



文字列探索の計算量の概算

- 力まかせ法
  - 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：i を進めるごとに，探索文字列の末尾まで照合して不一致 ⇒ 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回以下
⇒ 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，探索文字列の先頭まで照合して不一致 ⇒ 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

- 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）

- 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();
  - 各要素に対して，個別にnew処理が必要なことに注意せよ
- 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;
  - クラス型（参照型）の変数やその配列の構造を再確認せよ

確認問題

- Javaの配列とfor文
  - クラス型の配列dataに関して，下記のA)やB)の処理は意味があるが，C)の処理は意味がない理由を述べよ

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

- 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
  - 基本型の代わりには，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

- 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分探索やソートは，要素が大小比較できることが条件
  - 要素クラスは java.util.Comparable インタフェースを実装