アルゴリズムとデータ構造

第11回　ハッシュテーブル

第11回のキーワード

アルゴリズム関係

- ルックアップテーブル（lookup table）
- ハッシュテーブル / ハッシュ表（hash table）
- ハッシュ関数
- ハッシュ値（hash value） / ハッシュコード（hash code）
- バケット（bucket）
- 衝突（collision）
- $O(1)$
- キー（key），値（value）
- チェーン法（separate chaining）
- オープンアドレス法（open hashing）

Java関係

- 文字コード（Unicode）
- hashCodeメソッド（Objectクラスのメソッド）
- 整数型のオーバーフロー（最大値を超えても計算）
- 負数に対する%演算子

確認問題

- 配列によるルックアップテーブル
  - 以下は1桁の整数を英単語に“翻訳”するプログラムである
  - 空欄(1箇所)を埋めて，配列が「整数からデータを探す対応表」(ルックアップテーブル)に使えることを理解せよ

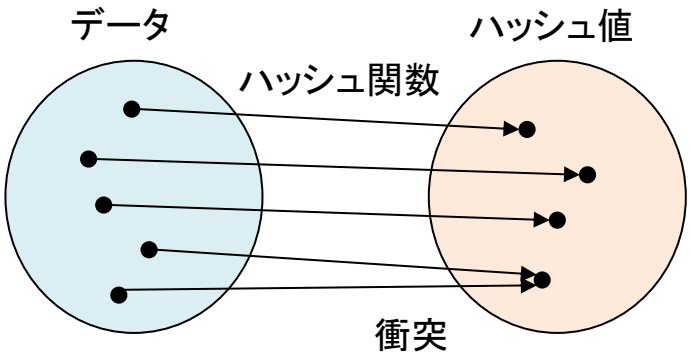
```
import java.util.Scanner;

public class Main {
    final static String[] ewords = { "zero", "one", "two", "three",
        "four","five", "six", "seven", "eight", "nine" };

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("number? ");
        int n = sc.nextInt();
        if (0 <= n && n <= 9)
            System.out.println(ewords[    ]);
    }
}
```

プログラムを入力して実行させずに動作を考えて理解すること

ハッシュ関数

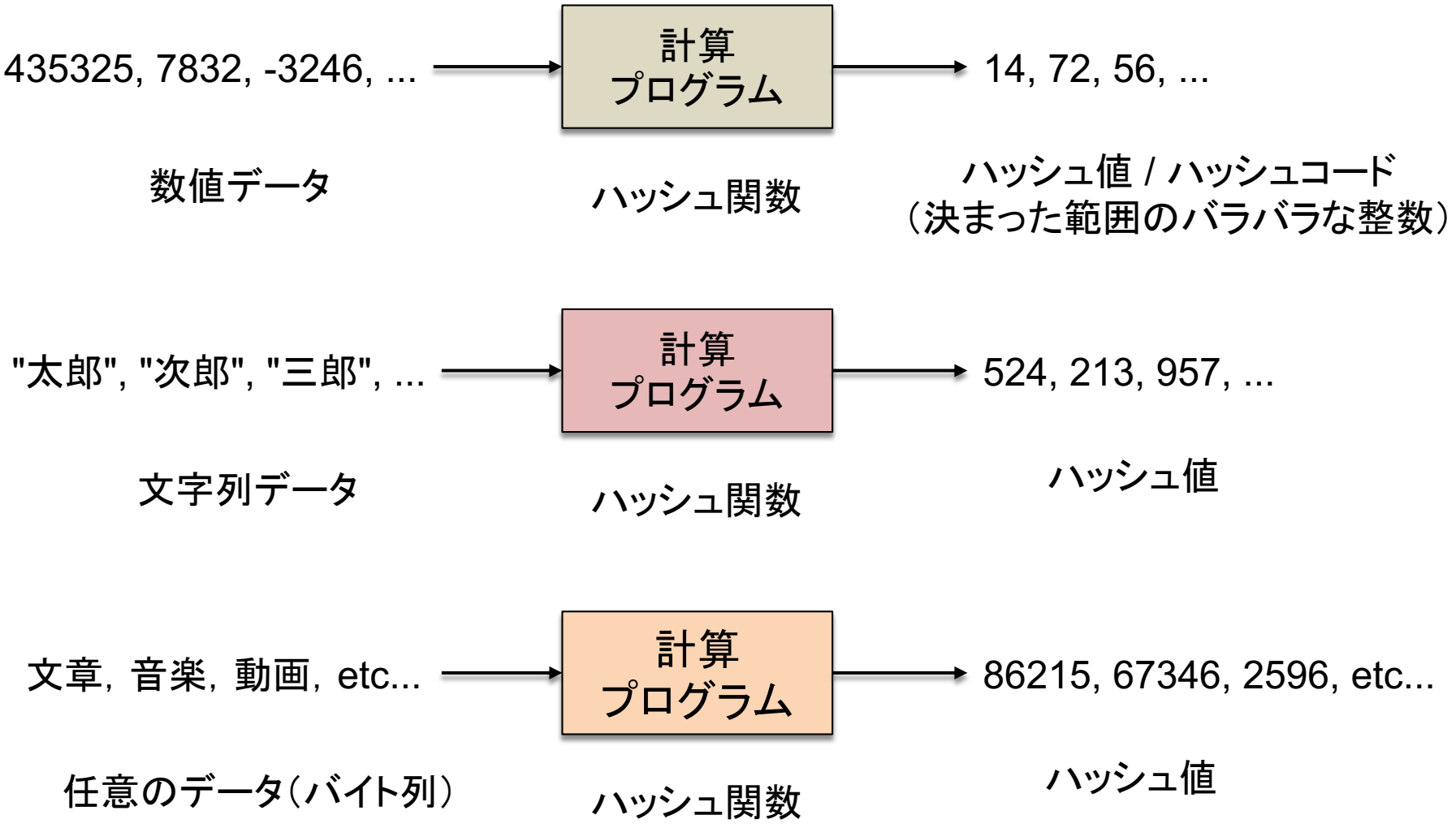
- ハッシュ関数
  - 任意のデータ(x)から，「ハッシュ値」と呼ばれる指定範囲のバラバラな整数(h)を算出する関数

$$h = hash(x)$$



- ハッシュ関数の特徴
  - 数学の関数の性質により，
    ハッシュ値が異なれば，データは必ず異なる
  - 異なるデータのハッシュ値が一致することは，稀に起きる
    ⇒ ハッシュ値の「衝突」という
  - コンピュータでは，文字や画像も2進数の形で記録されているので，任意のデータのハッシュ値を算出できる

ハッシュ関数の概念


435325, 7832, -3246, ...
計算
プログラム
14, 72, 56, ...
数値データ
ハッシュ関数
ハッシュ値 / ハッシュコード
（決まった範囲のバラバラな整数）
"太郎", "次郎", "三郎", ...
計算
プログラム
524, 213, 957, ...
文字列データ
ハッシュ関数
ハッシュ値
文章，音楽，動画，etc...
計算
プログラム
86215, 67346, 2596, etc...
任意のデータ（バイト列）
ハッシュ関数
ハッシュ値

確認問題

- ハッシュ関数
  - 下記は，任意の文字列を整数に変換する「ハッシュ関数」の実装例である
  - hash("one") の値を表す計算式を示せ

```
public static int hash(String key) {
    final int X = 37, HASHSIZE = 31;
    int L = key.length();
    int h = 0;
    for (int i = 0; i < L; i++) {
        int c = key.charAt(i);
        h = h * 37 + c;
    }
    return Math.abs(h % 31);
}
```

hash("one")の計算に必要な文字コード

'o' = 111

'n' = 110

'e' = 101

まずプログラムを実行せずに頭で考えて解答すること（計算をPCでするのは可）

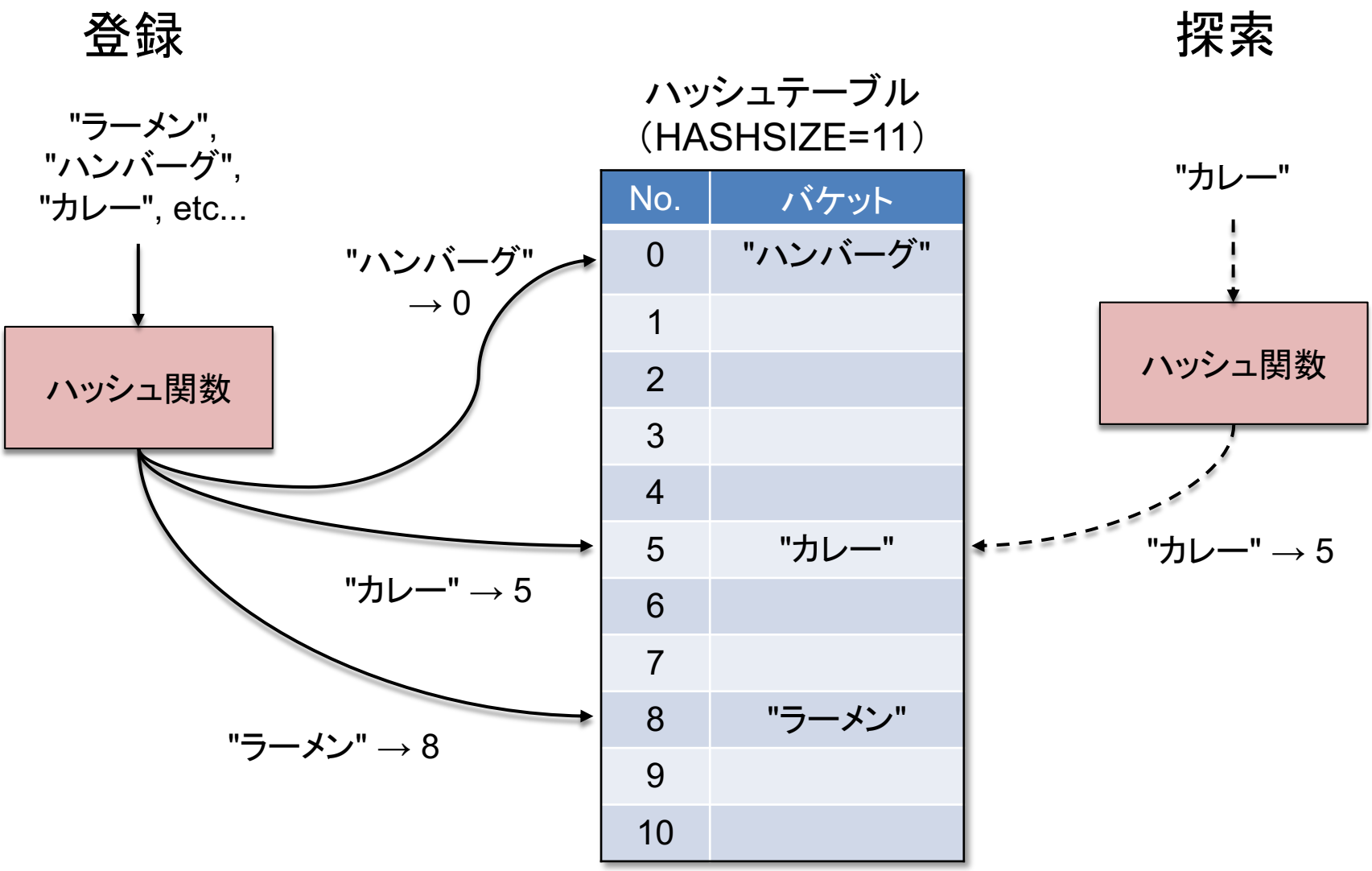
  - 今回の演習課題の2.を解答せよ

ハッシュテーブル

- ハッシュテーブル（ハッシュ表）
  - データの内容自体から，それを格納・探索する場所（配列の中の位置）を即時に計算で決めるデータ構造
  - データを「ハッシュ関数」によって整数 h に変換し，配列の要素 hashtable[h] をデータの格納場所（バケット）とする
  - 「衝突」がない理想的な条件では，$O(1)$ と極めて高速
  - 配列の中身を順に埋めるのではなくて，スカスカに使う

- よいハッシュ関数の条件
  - 格納場所が重ならないように衝突がない（または少ない）
  - データが少しでも異なれば，ハッシュ値も全く異なる
  - ハッシュ値が分散し，バケットを全体的にまんべんなく使う

ハッシュテーブルの仕組み



確認問題

- ハッシュテーブルへの登録
  - 演習問題の4.を解答し，ハッシュテーブルによるデータの登録と探索の原理を理解せよ
  - このハッシュテーブルの問題点を説明せよ

```
hashtable[   ] = "rei (0)";
hashtable[   ] = "ichi (1)";
hashtable[   ] = "ni (2)";
hashtable[   ] = "san (3)";
hashtable[   ] = "shi (4)";
```

```
hashtable[   ] = "go (5)";
hashtable[   ] = "roku (6)";
hashtable[   ] = "shichi (7)";
hashtable[   ] = "hachi (8)";
hashtable[   ] = "kyuu (9)";
```

- ハッシュテーブルからの探索
  - データを探索するときの計算量を考察せよ

```
int h = hash(eng);
String entry = hashtable[h];
```

ハッシュテーブルの利用

- キー（key）と値（value）
  - ハッシュテーブルは，通常データベースのように使う
  - キーは，ハッシュ値を計算して探索に使うデータ（例：氏名）
  - 値は，キーに結びつけて格納するデータ（例：成績方法）
- ハッシュ関数の作り方
  - データの全体（または十分な長さ）を使って算出する
  - 値を分散させるには，素数による乗除算を繰り返すとよい
- ダメなハッシュ関数の例
  - 先頭文字だけを使う ⇒ 残りの文字が異なっても値が衝突
  - ハッシュ値が必ず偶数になる ⇒ バケットの半数が無駄

衝突への対処方法

- 完全ハッシュ
  - 全ての登録データが分かっている場合に，ハッシュ関数の計算方法を，必ず衝突が起きないように設計しておく
- チェーン法
  - ハッシュテーブルの各要素（バケット）を連結リストにする
  - 例）class Node { String key; int value; Node next; }

    Node [ ] hashtable = new Node[HASHSIZE];
- オープンアドレス法
  - もし衝突が起きたら，別のアルゴリズムで計算しなおした位置（単純な方法の場合，次の隣のバケット）に格納する

チェーン法

- 各バケットを連結リストにして複数の値を格納する
  - バケットには，リストの先頭要素への参照（head）が入る