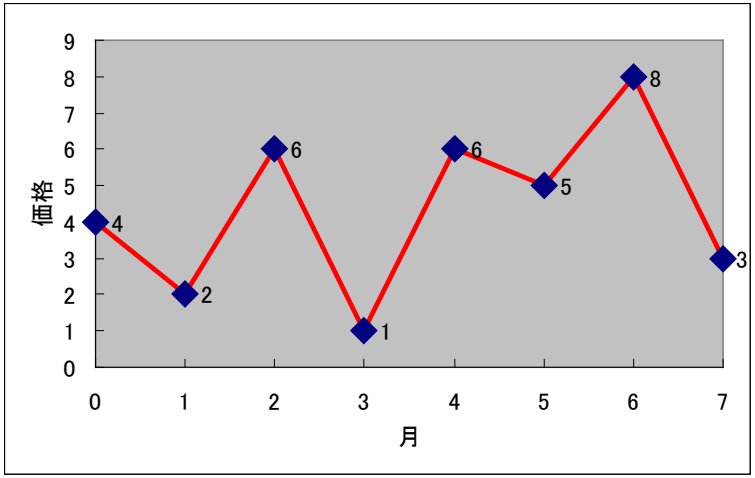
データ構造とアルゴリズム
第3週

掛下　哲郎
kake@is.saga-u.ac.jp

前回のポイント

- アルゴリズムの効率を，具体例を用いて検討
  - 株価データから「最大売却益」をもとめる例
  - データ数 n
    - 素直なアルゴリズム
      → $O(n^2)$
    - 頭を使って改良
      → $O(n)$
  - n次多項式の計算



今日学ぶこと

探索（search）問題

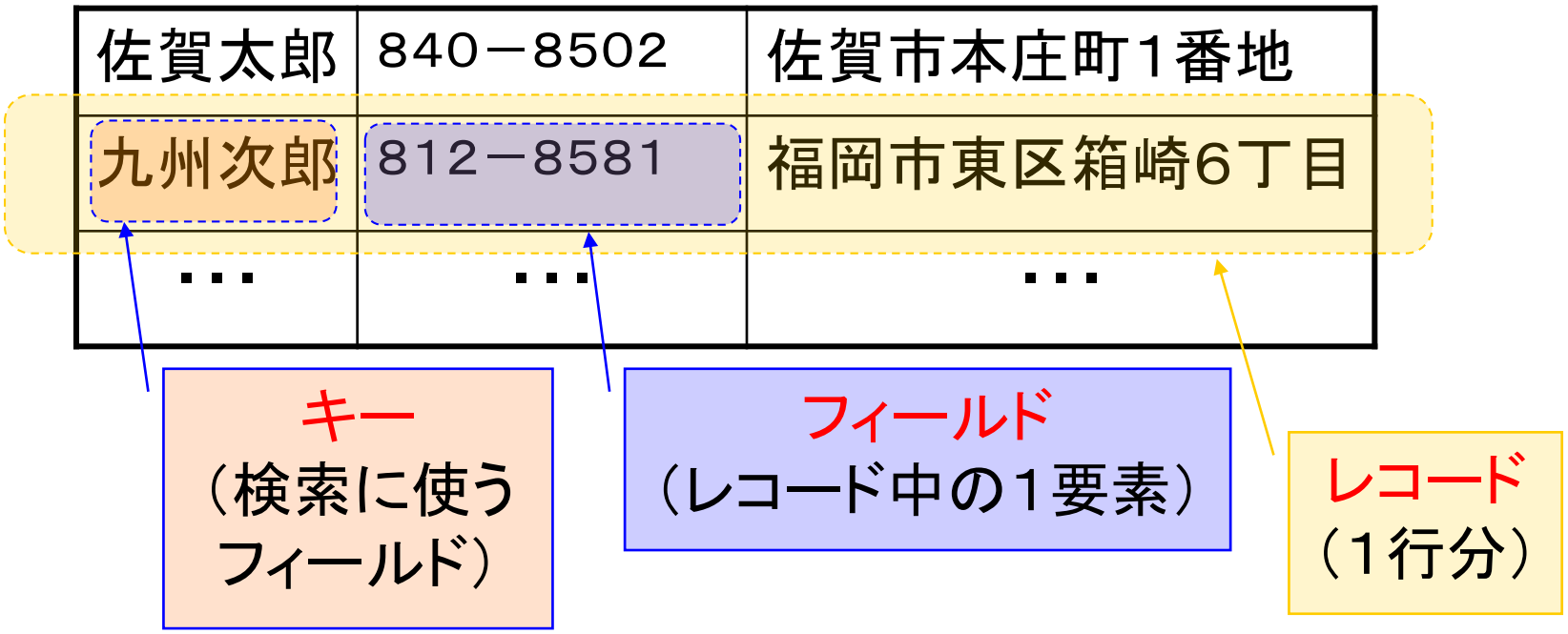
- データ集合から，自分の欲しいデータを探す

さまざまな探索方法

- 逐次探索
- 順序関係を利用した探索
- m-ブロック法
- 2分探索
- ハッシュ

探索問題

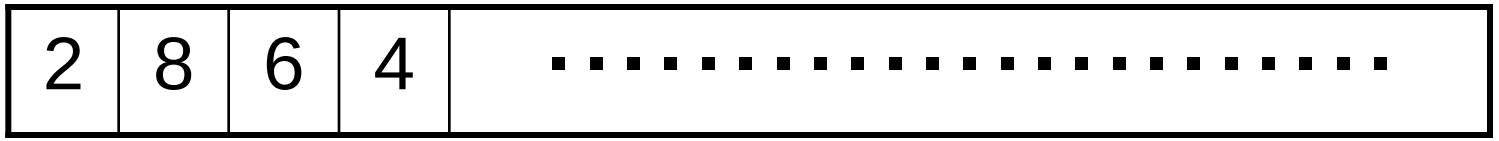
- 例：住所録からｘさんの住所を探す．



探索：キー フィールドを探して，一致するレコードを探す

探索問題

- 簡単のため，以下を仮定
  - 整数のキーフィールドが一つだけのデータ
  - データ集合中に，指定された整数が「有る」か「無い」かを返す問題

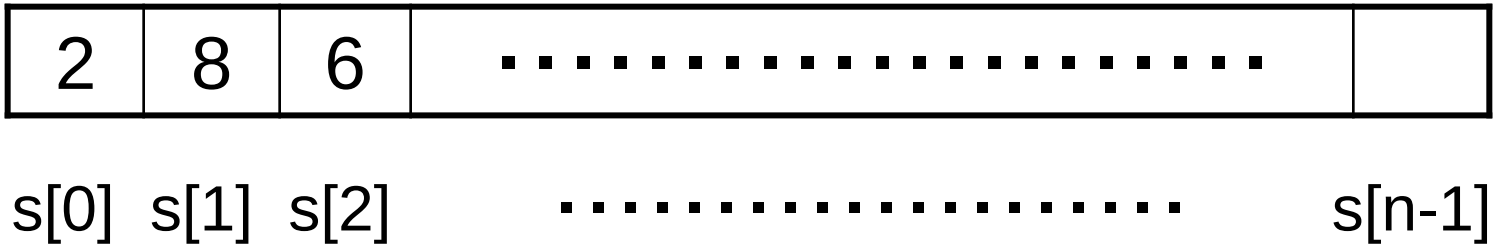


（探索対象のデータ集合）

ここではデータ構造として配列を仮定

- データ構造として配列（1次元）を使用

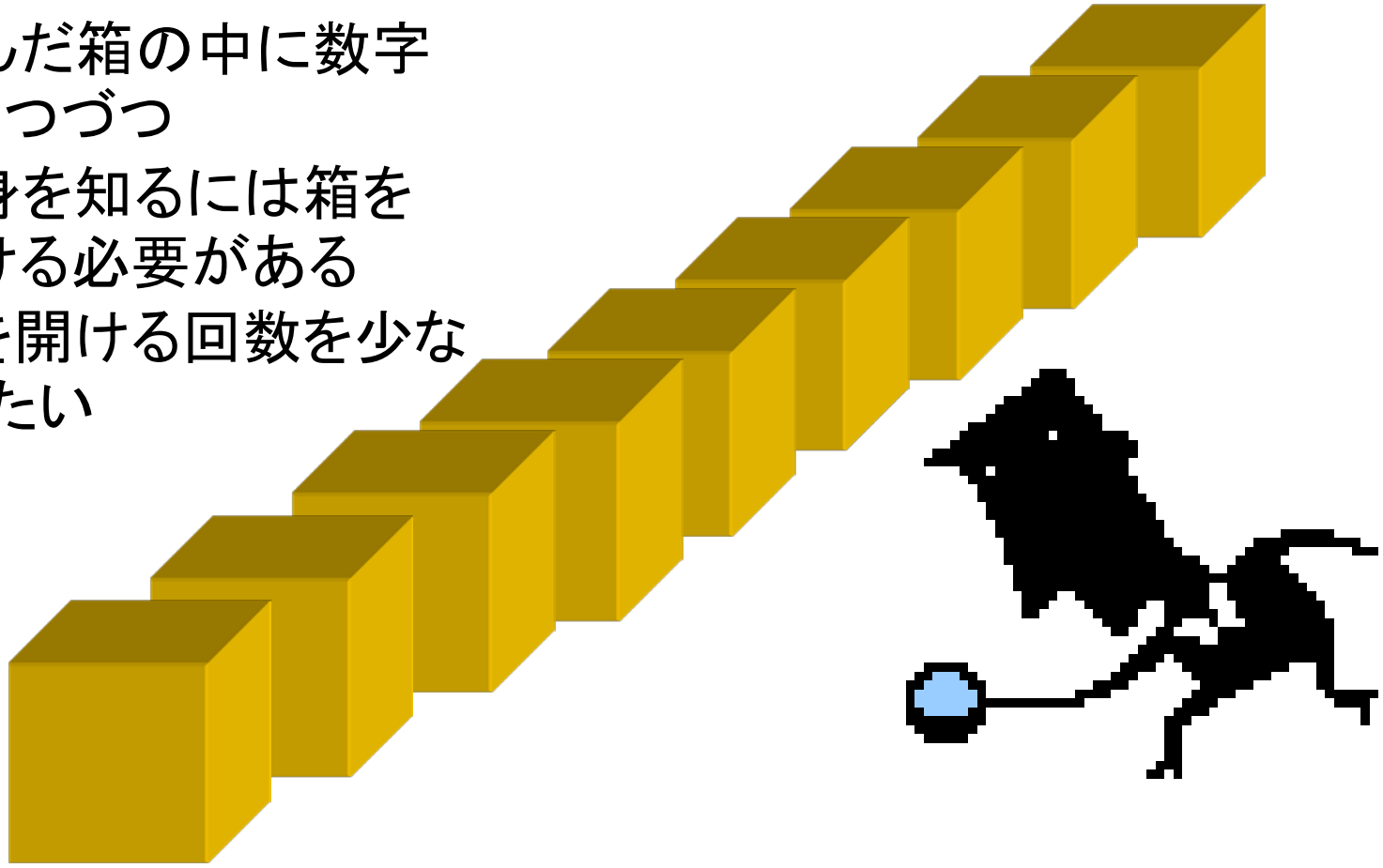
int s[*n*]



どんな探索アルゴリズムを考える？

イメージ

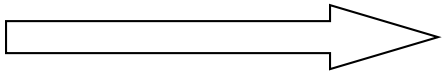
- 並んだ箱の中に数字が1つづつ
- 中身を知るには箱を開ける必要がある
- 箱を開ける回数を少なくしたい



逐次探索

- 配列要素を先頭から順に調べてゆく

9	4	1	8	3	7	10	5	6



手間数(配列要素を調べる回数)は？

3を探索：　5回
5を探索：　8回
9を探索：　1回
2を探索：　9回

アルゴリズム(逐次探索)

入力: 質問データ x

1. 探索位置を配列の先頭とする.
2. 配列の末尾に到達するまで以下の処理を繰り返す.
   2-1. 探索位置の要素がxと一致したならば，探索位置を返して終了する.
   2-2. そうでなければ，探索位置を1つ進める.
2. 検索失敗を返して終了する.

要書き取り

正しさの確認

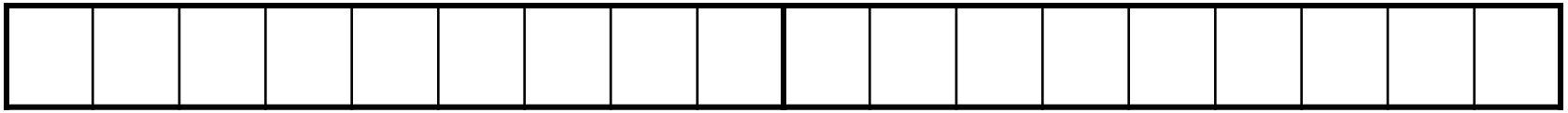
- 停止性(「停止するか？」)
  - 配列の先頭からスタート
  - 配列の末尾に到達するまで繰り返し
  - 探索位置は 1ずつ増加

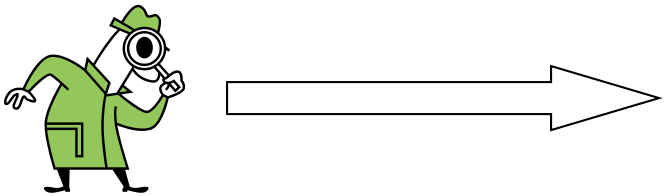
  → 必ず停止
- 正当性(「正しく解くか？」)
  - 全てのデータを調べる
  - 見つけたときはその位置を返す
  - 見つからなかったときは失敗を返す

  → 正しく動作

要書き取り

逐次探索の効率





見つからないとき
・・・ n回　O(n)

- 見つかるとき
  - 最良・・・1回　O(1)
    - 先頭で見つけたとき
  - 最悪・・・n回　O(n)
    - 最後で見つけたとき
  - 平均・・・(n+1)/2回　O(n)
    - 出現確率が等確率と仮定。期待値を計算すれば良い

$\Sigma_{k=0\sim(n-1)} (1/n)\ (k+1)$

整理

- n要素の配列における探索問題
  - 基本的にO(n)　(平均、最悪ともに)
  - つまり、データの数に比例する時間計算量
    - データ数が10倍になれば手間も10倍。100倍になれば手間も100倍。・・・
- 身の回りの「探索問題」
  - アドレス帳、辞書における「探索」
  - 何か違いは？

順序関係がある場合

- 「順序」が定義され、その順番にデータが並んでいる場合 (数字の大小、あいうえお順、etc.)

15	4	25	8	39	7	10	5	6

（バラバラ）

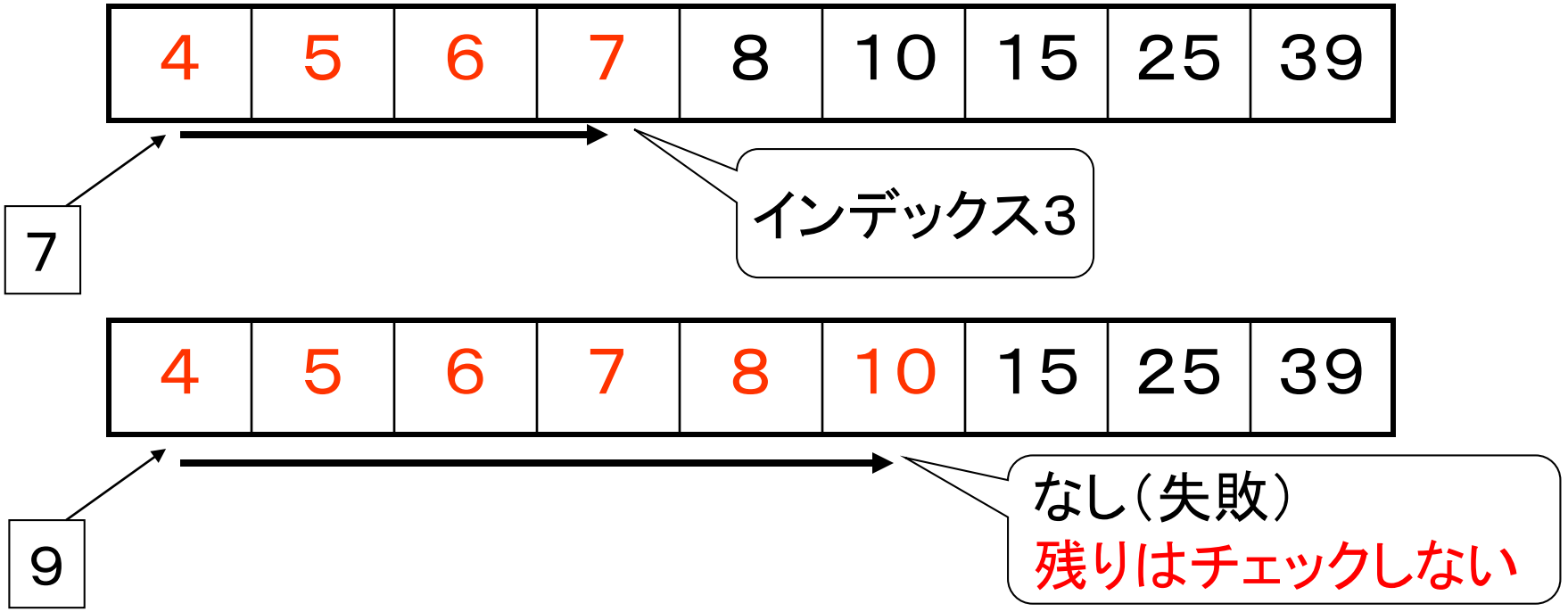
4	5	6	7	8	10	15	25	39

（大小順）

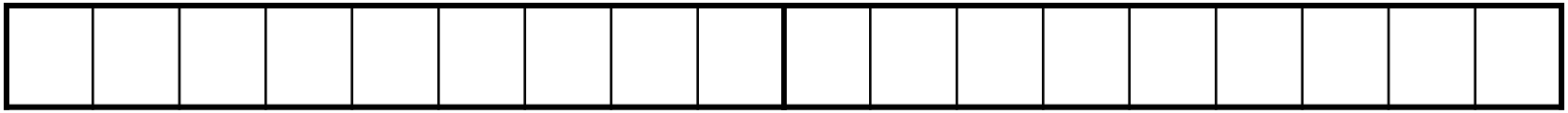
「順番通りに並んでいる」
という性質を、どう利用する？

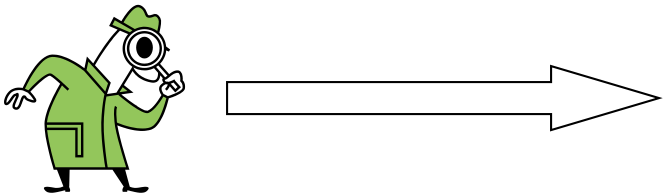
順序関係を利用した探索（1）

- 逐次探索の改良版
  - 「探す数字」>「配列中の数字」 になったら処理をやめる（→ 無駄な探索を省く）



順序関係を利用した逐次探索の効率





- 見つかるとき　・・・通常の逐次探索と同じ
- 見つからないとき
  - 最良　1回　　O(1)
    - 最初の要素より小さい
  - 最悪　n回　　O(n)
  - 平均　$(n+1)/2$ 回　　O(n)

少しはマシになるが，
オーダーは変わらない

整理

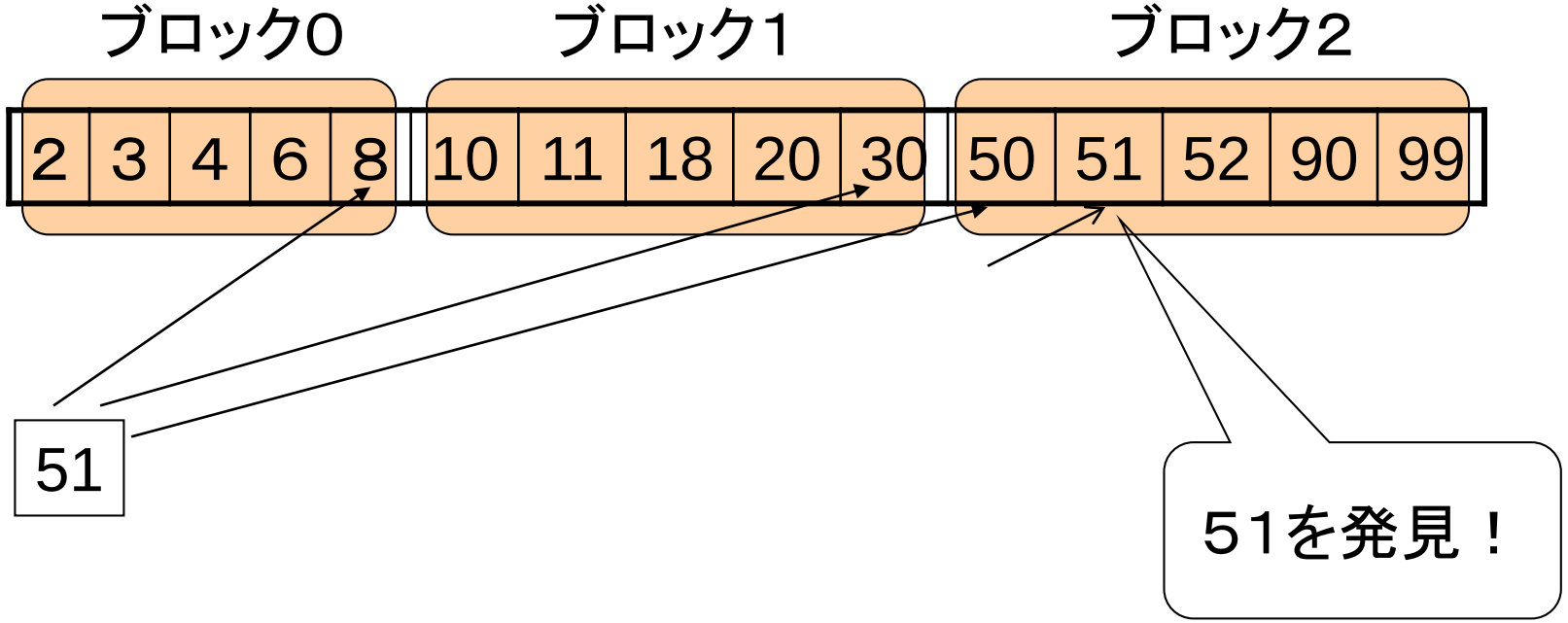
- 逐次探索
  - 平均、最悪ともに O(n)
- 逐次探索（順序関係を利用）
  - 平均、最悪ともに O（n）

順序関係をもっと賢く利用できれば、もっと改善できる？

要書き取り

順序関係を利用した探索(2)

- m-ブロック法
  - *n*個のデータを*m*個のブロックに分割。
    - 各ブロックには*n*/*m*要素を保持。
  - *n*/*m*個とびにチェックし，可能性のあるブロックを見つける
  - 可能性のあるブロックの中にある*n*/*m*個を逐次探索



m-ブロック法のアルゴリズム

- 入力：質問データx

1. ブロック数mを定める.
2. ブロック長kをn/mとする（小数点以下切り上げ）.
3. 探索位置を配列の先頭ブロックとする.
4. 末尾を除く各ブロックに対して以下の処理を繰り返す.
    4-1. 当該ブロックの末尾の要素がxの値以上ならば，ループから出る.
    4-2. そうでなければ，探索位置を次のブロックに進める.
5. 探索位置を当該ブロックの先頭とする.
6. 当該ブロックの末尾に到達するまで以下の処理を繰り返す.
    6-1. 探索位置の要素がxの値以上ならば，ループから出る.
    6-2. そうでなければ，探索位置を1つ進める.
7. 探索位置の要素がxと一致したならば，探索位置を返す.
8. そうでなければ，探索失敗を返す.

m-ブロック法の効率

ステップ4　ステップ6

- 比較回数　≦　$(m\text{-}1) + [n/m]$

  $[\alpha]$ ･･･α以上の最小整数

  $[n/m] \leqq n/m + 1$ より

  比較回数 $\leqq m + n/m$

- 相加平均と相乗平均

  $\sqrt{a \times b} \leqq (a+b)/2$　（等号は $a=b$ のとき）

- 比較回数最小となるのは、$n/m = m$ のとき

  つまり、$m = \sqrt{n}$ のときが最小で、$O(\sqrt{n})$

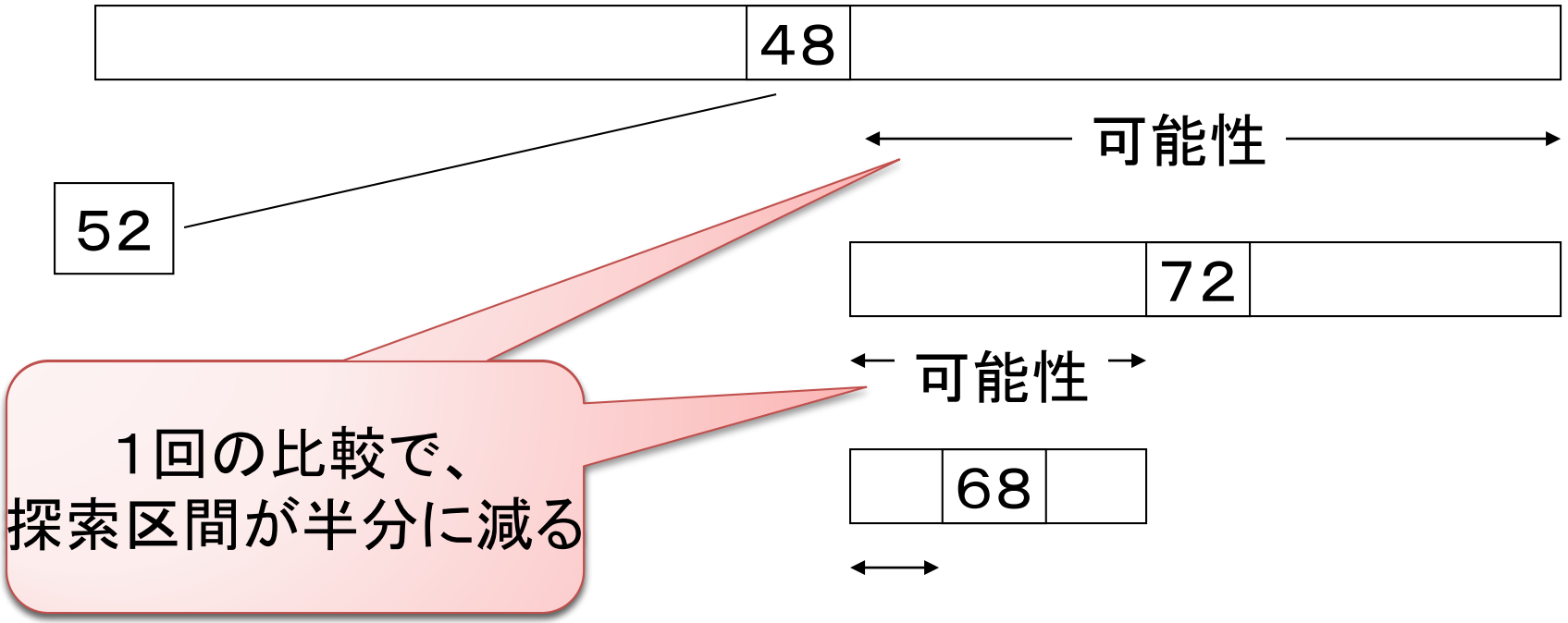
整理

- 逐次探索
  - 平均、最悪ともに $O(n)$
- 逐次探索（順序関係を利用）
  - 平均、最悪ともに $O(n)$
- m-ブロック法
  - 平均，最悪ともに $O(\sqrt{n})$

要書き取り

順序関係を利用した探索(3)

- 二分探索
  - 対象区間の真ん中に見当をつけながら探す



二分探索法のアルゴリズム

- 入力：質問データx

1. xが配列の先頭要素よりも小さいか，末尾要素よりも大きい場合，探索失敗を返す．
2. 探索区間を配列の全域とする．
3. 探索区間が空でない限り以下の処理を繰り返す．
   3-1. 探索区間の中央の要素を求める（小数点以下切り捨て）．
   3-2. xが中央の要素よりも小さいならば，探索区間の末尾を中央-1とする．
   3-3. そうでなければ，探索区間の先頭を中央+1とする．
4. 探索区間の末尾の要素がxと一致したならば，末尾の位置を返す．
5. そうでなければ，探索失敗を返す．

要書き取り

二分探索法の効率

- 最悪でもO(log n)
  - 探索区間は、高々$\log_2 n$ 回の分割で大きさが1になる
  - なぜなら・・・

    (k回分割後の探索区間の幅)=n/2/2/2/・・・/2 ≦ 1

    (2でk回割る)

$$n \times 2^{-k} \leqq 1$$

$$\log_2 n - k \log_2 2 \leqq \log_2 1$$

$$\log_2 n \leqq k$$

整理

- 逐次探索
  - 平均、最悪ともに O(n)
- 順序関係を利用した逐次探索
  - 平均、最悪ともに O(n)
- m-ブロック法
  - 平均，最悪ともにO($\sqrt{n}$)
- 二分探索法
  - 平均，最悪ともにO(log n)

要書き取り

ハッシュ法

- 平均計算時間 O(1)
- n個のデータを格納するのに、1.5～2倍程度の大きさの配列s[m]を用いる。
- アイデア
  - データ x の格納位置を、関数hash(x)で求める
    - x：格納したいデータ
    - i = hash(x)：0～m-1の整数を返す関数
    - s[i] にデータxを格納
    - 異なるxで同じiになったとき ⇒ 次の位置から空きを探して格納
  - 探索は、hash[x]の位置から順次探索

ハッシュ法の具体例

- 格納したいデータ

  10, 15, 20, 33, 44　(n=5)

- m = 10とし、s[0],s[1],…,s[9] にデータを格納（ハッシュ表と呼ぶ）
- hash(x) = x % m で定義

xをmで
割った余り

(ハッシュ表)

10	20	—	33	44	15	—	—	—	—
0	1	2	3	4	5	6	7	8	9

ハッシュ表を作る(1)

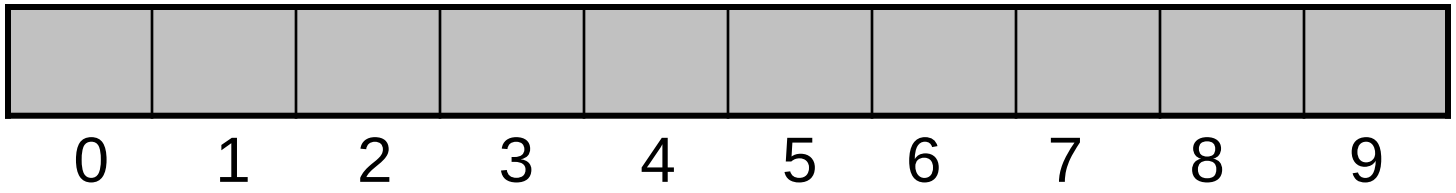
(探索対象のデータ)　n = 5

27　11　13　10　41

hash(27) = 7
より，位置7に格納

ハッシュ関数は
hash(x) = x % 10
とする

(ハッシュ表)　m = 10



ハッシュ表を作る(2)

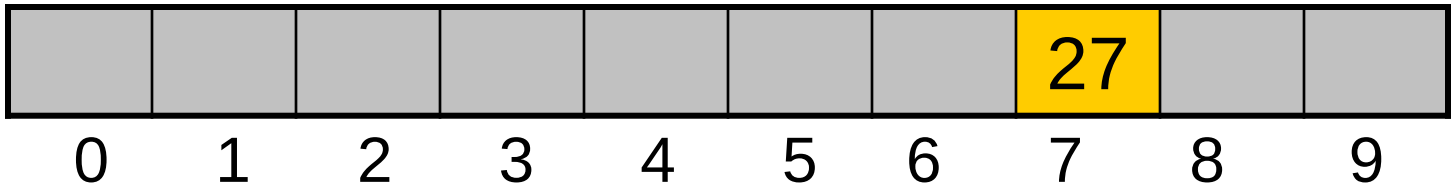
(探索対象のデータ)　n = 5

11　13　10　41

hash(11) = 1
より，位置1に格納

ハッシュ関数は
hash(x) = x % 10
とする

(ハッシュ表)　m = 10



ハッシュ表を作る(3)

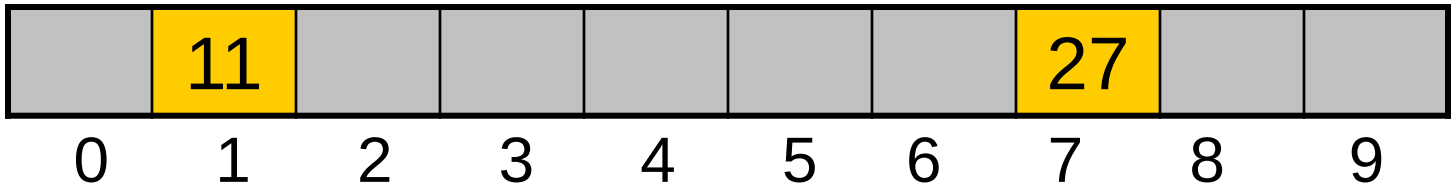
(探索対象のデータ)　n = 5

13　10　41

ハッシュ関数は
hash(x) = x % 10
とする

hash(13) = 3
より，位置3に格納

(ハッシュ表)　m = 10



ハッシュ表を作る(4)

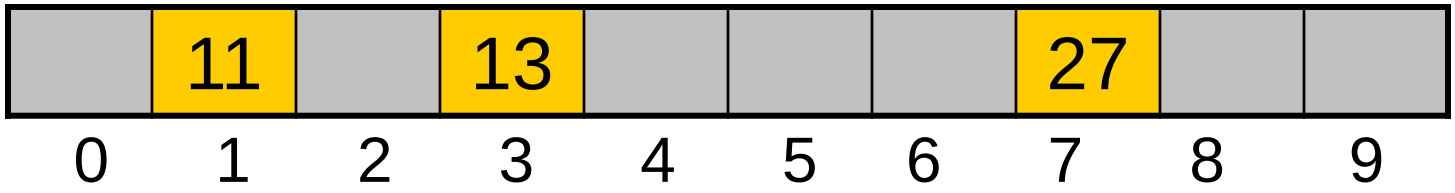
(探索対象のデータ)　n = 5

10　41

ハッシュ関数は
hash(x) = x % 10
とする

hash(10) = 0
より，位置0に格納

(ハッシュ表)　m = 10



ハッシュ表を作る(5)

(探索対象のデータ)　n = 5

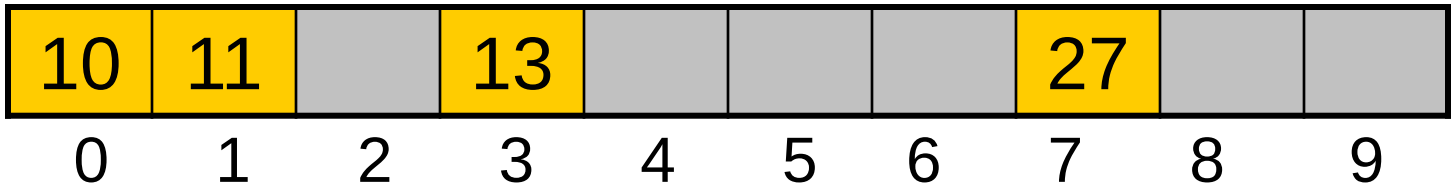
41

hash(41) = 1
より，位置1に格納

先客がいるので
右隣へ

ハッシュ関数は
hash(x) = x % 10
とする

(ハッシュ表)　m = 10

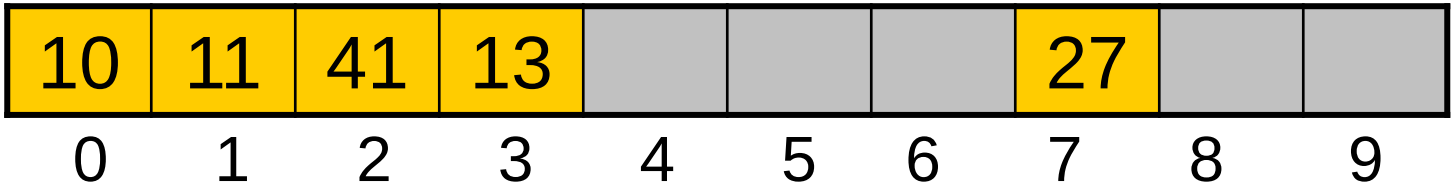


完成したハッシュ表

(探索対象のデータ)　n = 5

ハッシュ関数は
hash(x) = x % 10
とする

(ハッシュ表)



データ登録アルゴリズム

1. ハッシュ表の各要素を空にする.
2. 登録対象の各データdについて以下の処理を繰り返す.
   2-1. dのハッシュ値jを求める.
   2-2. ハッシュ表の位置j以降で空の要素を探し，最初の空き要素にdを格納する.

ハッシュ法の具体例

(ハッシュテーブル)

10	20	—	33	44	15	—	—	—	—
0	1	2	3	4	5	6	7	8	9

- データ34を探索
  - hash(34) = 34 % 10 = 4
    s[4], s[5], s[6](空き)と順に調べ、
    「データ無し」と返す（失敗）
- データ44を探索
  - hash(44) = 44 % 10 = 4
    s[4] から調べ、一発で見つける。

ハッシュ法で探索(1)

探索例:

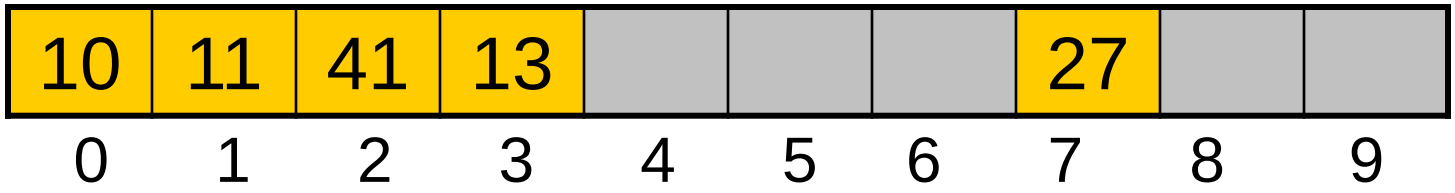
13を探したい！

ハッシュ関数は
hash(x) = x % 10
とする

hash(13) = 3
より，位置3を探す

発見！

(ハッシュ表)



ハッシュ法で探索(2)

探索例:

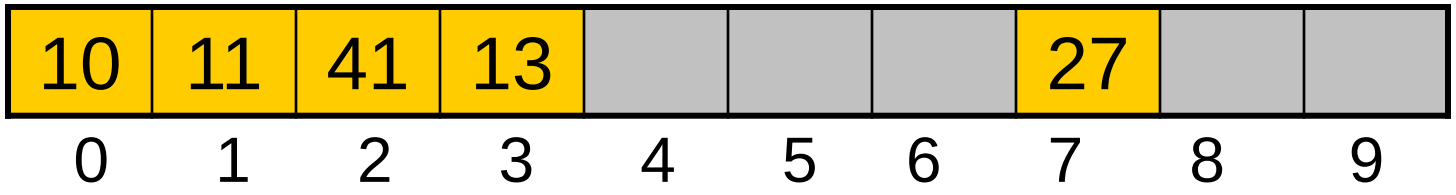
41を探したい！

ハッシュ関数は
hash(x) = x % 10
とする

hash(41) = 1
より，位置1を探す

発見！

(ハッシュ表)



ハッシュ法で探索(3)

探索例:

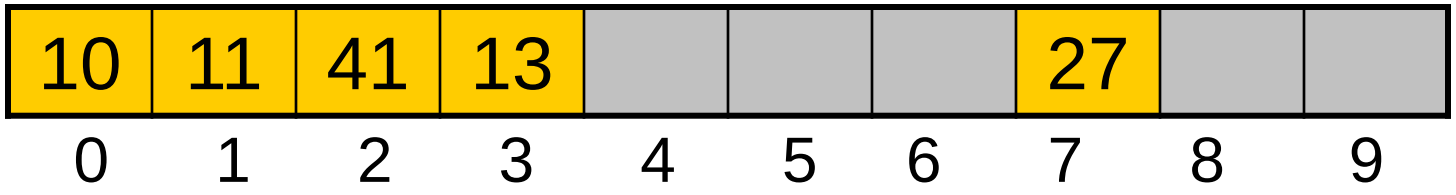
29を探したい！

ハッシュ関数は
hash(x) = x % 10
とする

hash(29) = 9
より，位置9を探す

無し！

(ハッシュ表)



ハッシュ法で探索(4)

探索例:

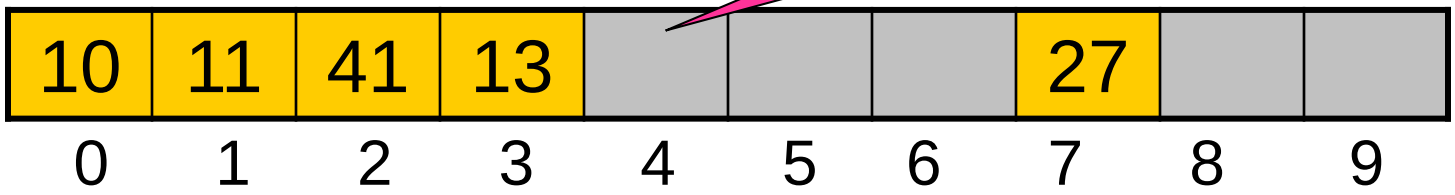
12を探したい！

ハッシュ関数は
hash(x) = x % 10
とする

hash(12) = 2
より，位置2を探す

無し！

(ハッシュ表)



データ探索
アルゴリズム

入力:質問データx
1. xのハッシュ値jを求める.
2. ハッシュ表の位置jを探索位置とする.
3. 探索位置が空でない限り以下の処理を繰り返す.
    3-1. 探索位置の要素がxと一致したならば，探索位置を返す.
    3-2. そうでなければ，探索位置を1つ進める.
4. 探索失敗を返す.

ハッシュ法の効率

- 平均成功探索回数　～　　$1/2 \{ 1 + 1/ (1-a) \}$
- 平均不成功探索回数 ～　　$1/2 \{ 1 + 1/ (1-a)^2 \}$
- a は ハッシュ表の占有率。

  データ量には依存していない

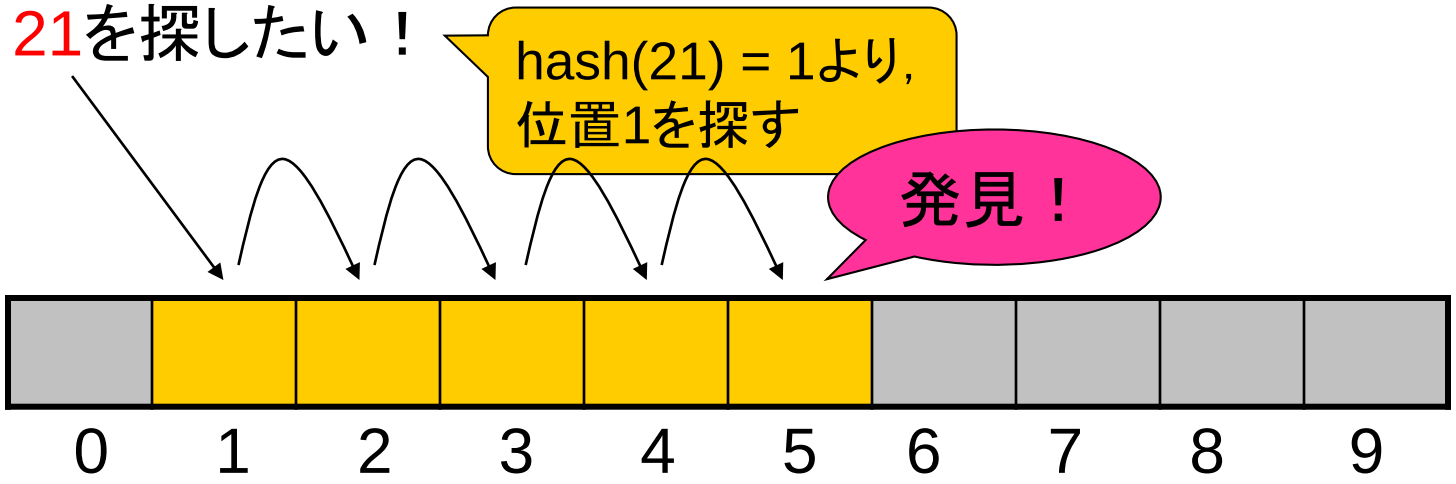
  ⇒ 1にくらべて充分小さければ、平均1回の比較で済む

どんなとき効率が悪くなるか？

(探索対象のデータ)　n = 5

11　51　31　71　21

ハッシュ関数は
hash(x) = x % 10
とする



(ハッシュ表)　m=10

結局，n個全ての要素のチェックが必要　→ 手間O(n)

本日のまとめ

- 逐次探索
  - 平均、最悪ともに O(n)
  - 未登録データを探索した場合，全要素の確認が必要
- 順序関係を利用した逐次探索
  - 平均、最悪ともに O(n)
- m-ブロック法
  - 平均、最悪ともにO($\sqrt{n}$)
- 二分探索法
  - 平均、最悪ともにO(log n)
- ハッシュ法
  - 平均O(1)。最悪時は?

未登録データを探索した場合でも，全要素の確認は不要

確認テスト

1. データが 1, 2, 4, 6, 7, 8, 9, 10,13,15,17 と昇順に格納されているとする。
   (a) m=4でm-ブロック法を用いて探索する。8を探索するときと、11を探索する場合のそれぞれについて、どのような順番で要素をチェックするか説明せよ。
   (b) 13, 6, 3を二分探索するとき、それぞれ，どのような順番で要素をチェックするか説明せよ。

確認テスト

2. ハッシュ法での探索を考える.
   - データの集合は 1, 3, 16, 8. (n=4. この順番でハッシュ表に入れる)
   - ハッシュ表のサイズ m = 8
   - ハッシュ関数 hash(x) = x % 8

   データ登録後のハッシュ表の状態を示せ.

   4つのデータ１，５，８，27 を探索するとき，どのようにハッシュ表をチェックしていくのか，それぞれ示せ.
3. ハッシュ法の最悪時の時間計算量は？

   ヒント：ハッシュ表へ格納するとき衝突ばかり起こったら？