

# 連絡事項

---

- 次回(11/8)は演習棟に集合してください
  - パスワードや PIN コードを忘れないように
- 次々回(11/15)は休講にします

# 本日の話題：探索問題

---

- コンピュータを使った探索
- 実世界の探索
- 探索問題の定式化
- 線形探索
- 計算量の評価
- 2分探索
- 現実のコンピュータが扱える問題のサイズ

# 探索 (1/3)

- (大量の)データの中から、特定の条件にあうものを探す
  - e.g. Web ページの中から「コンピュータサイエンス」というキーワードを含むものを探す
  - e.g. オンライン書店のコンピュータが、ISBN 番号 4781909949 の本の在庫状況を調べる
  - e.g. 銀行のコンピュータが、口座番号 XXXXXXXX の顧客の残高を調べる
  - e.g. ロサンゼルス往復の一番安いチケットを探す
  - e.g. 今日上映している一番面白い映画を探す

# 探索 (2/3)

- 探索はいたるところで行われる

- コマンドの実行

- \$ ls -l main.c

- main.c に関する情報を探す

- \$ man ls

- ls コマンドのマニュアルを探す

- \$ grep Hello main.c

- main.c の中の文字列 Hello を含む行を探す

- \$ find . -name '\*.c'

- 名前が .c で終了するファイルを探す

# 探索 (3/3)

- 探索はいたるところで行われる (cont.)
  - コンピュータへの login
    - ユーザ名をキーとして, パスワードのチェック, ホームの設定などに必要な情報を探す
  - Word での入力
    - スペルチェックに必要な情報等を探す
  - Web の閲覧
    - URL のホスト部 (e.g. `www.titech.ac.jp`) から, Web サーバの IP アドレス (e.g. `131.112.125.60`) を探す

# 図書の並べ方 (1/2)

- 図書館で本を書架に並べる方法により、探す手間や追加する手間に大きな差が生じる
  - 10冊なら簡単だが、100万冊だと大変
- 以下のような並べ方の得失を考える
  1. 購入順に並べる
  2. 書名に関して、あいうえお順に並べる
  3. 著者名に関して、あいうえお順に並べる
  4. ISBN番号順に並べる
  5. 分野別に分類し、各分野内ではあいうえお順に並べる

# 図書の並べ方 (2/2)

- 一般利用者が本を探す
  - 購入順, ISBN順: 購入時期やISBN番号は知らない可能性大
  - 書名順, 著者名順: 書名や著者名は, 知っているかもしれないし, 知らないかもしれない
  - 分野別 & 書名順: 分野を知っている可能性は高い
- 司書が本を追加する
  - 購入順: 比較的簡単
  - その他: 隙間を作る作業が大変

# 探索問題

- 探索問題をモデル化する
  - こういう場合, 一般性と単純さのトレードオフが大事
- 探索の対象となるデータ項目は, キー(key)と値(value)の対に限定
  - この対をレコードと呼ぶ
  - e.g. (cat, {猫, Unix のコマンドの一種})
  - e.g. (www.titech.ac.jp, 131.112.125.60)
  - キーは一意であれば何でもよい
  - 値は何でもよい (複雑な構造を持つ値も可)
- 与えられたキーに対応する値を探す問題と捉える

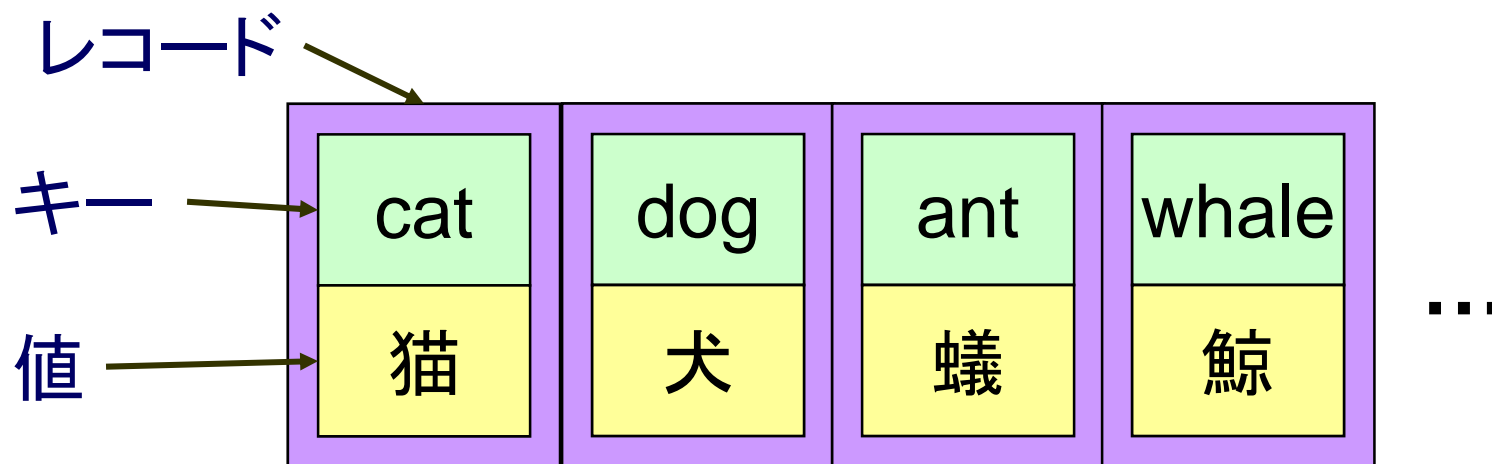
# 探索問題の例

- 日英単語の対応表から、英単語に対応する日本語の単語を探す
  - キーは英単語, 値は対応する日本語の単語
    - 簡単のため, 対応する日本語の単語は一つとする
  - 前から順番に調べれば探し出せることは確実
    - cf. でも, 英和辞典を前から順番に探したりはしない

|     |     |     |       |     |
|-----|-----|-----|-------|-----|
| cat | dog | ant | whale | ... |
| 猫   | 犬   | 蟻   | 鯨     | ... |

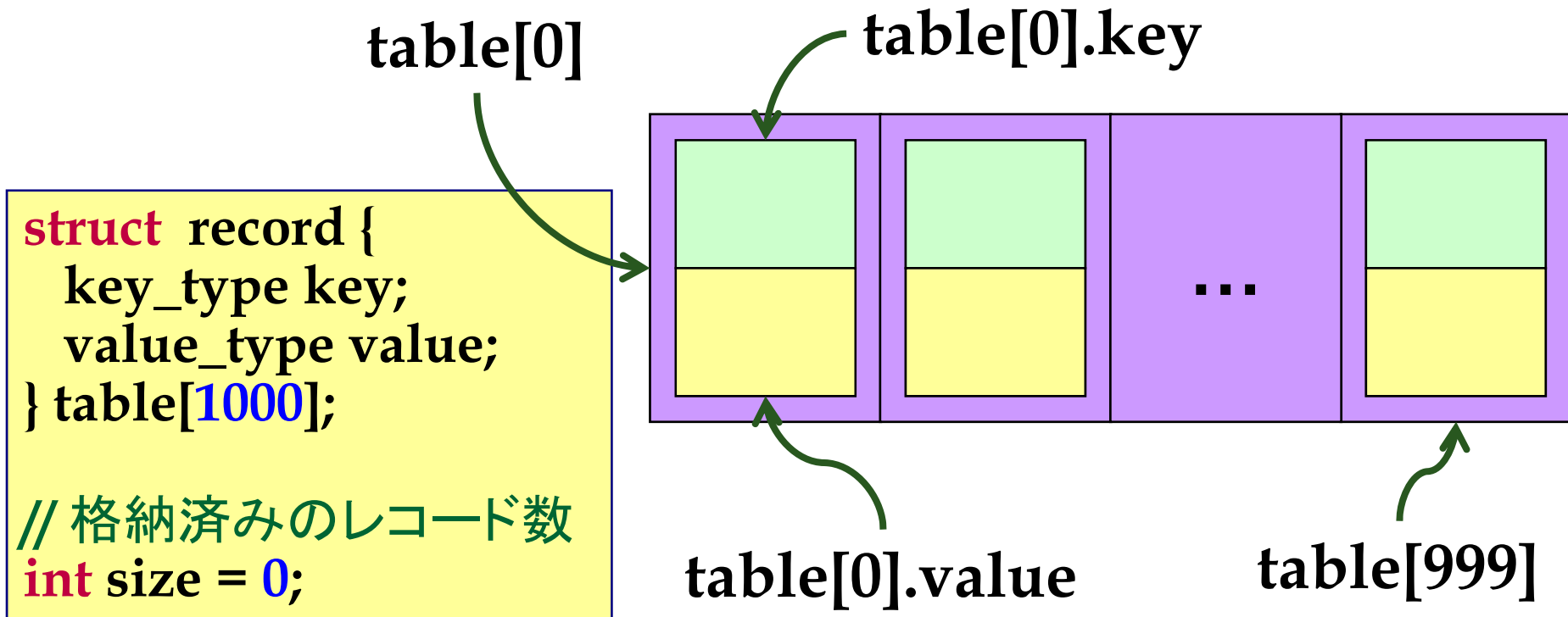
# データ構造 (1/2)

- 全体はレコードを複数並べた構造
  - e.g. レコードの配列で表す
- 各レコードはキーと値の対
  - e.g. キーと値を含む構造体で表す
- キーと値の構造は具体的な問題により異なる



# データ構造 (2/2)

- C 言語の配列は固定長
  - 長さを必要に応じて変えられた方が便利だが...
  - C++, Java などにはそのようなデータ構造もある

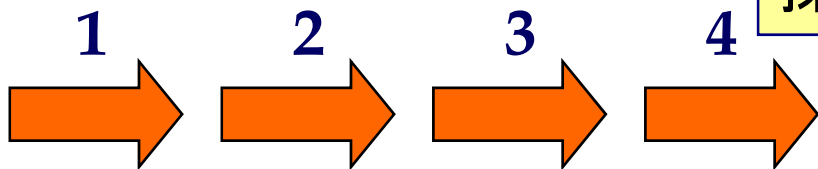


# 線形探索 (1/2)

- 基本的なアルゴリズム

- 前から順番に探す

```
int i;  
for(i = 0; i < size; i++) {  
    if (table[i].key が求めるキー) {  
        探索成功; 終了;  
    }  
}  
探索失敗; 終了;
```



|     |     |     |       |     |
|-----|-----|-----|-------|-----|
| cat | dog | ant | whale | ... |
| 猫   | 犬   | 蟻   | 鯨     | ... |

# 線形探索 (2/2)

- レコードの追加・削除の可能性も無視できない
  - cf. 図書の追加
- レコードを削除すると空いた場所ができる
  - 詰める必要アリ
- レコードを追加するには、末尾に空いた場所が必要
  - 場所さえあれば、末尾に追加するだけ

|     |   |     |       |  |
|-----|---|-----|-------|--|
| cat | ← | ant | whale |  |
| 猫   | ← | 蟻   | 鯨     |  |

# 計算量の評価 (1/2)

- プログラムの効率を分析・予測したい
  - データ構造やアルゴリズムの選択によって効率が変わる
  - cf. 本の並べ方にもいろいろな方法がありえた
- 実験と理論
  - 実験により, 特定のコンピュータで特定のデータを処理するために要する時間とメモリ量がわかる
  - 理論により, データサイズの増え方と計算に必要な時間やメモリ量の増え方の関係がわかる
  - ただし, 実験でも理論でも誤差は避けられない

# 計算量の評価 (2/2)

- 計算量の理論的分析を行う場合, コンピュータの動作を単純化したモデルを使うことが多い
  - e.g. 計算時間が, ループを回る回数に比例
  - e.g. 計算時間が, キーの比較回数に比例
  - e.g.  $i$  番目のレコードを読む時間と  $j$  番目のレコードを読む時間は同じ
  - 分析は簡単になるが, 現実とのギャップは広がる

# 線形探索の時間計算量 (1/2)

- レコード数を  $N (> 0)$  とする
- 探索のループを回る回数は, 1 以上  $N$  以下
  - 探索に成功する場合, 平均  $(N + 1)/2$  回
  - 探索に失敗する場合, 常に  $N$  回
- 単純化したモデルのもとで, 計算時間  $t(N)$  は  $N$  にほぼ比例
  - オーダー記法を用いると  $O(N)$ 
    - $O(N)$  は「 $N$  に比例」を意味するわけではない

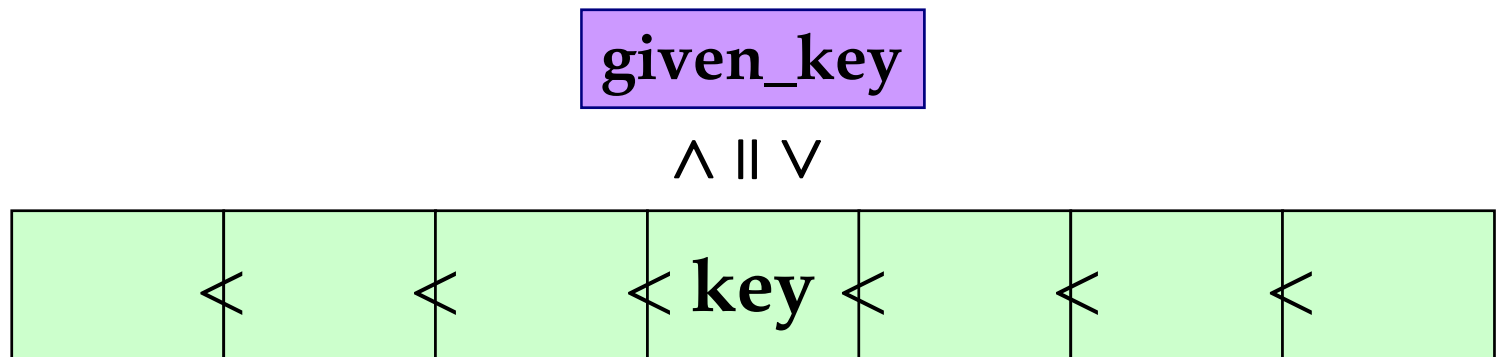
$$\exists c : \lim_{N \rightarrow \infty} \frac{t(N)}{N} < c$$

# 線形探索の時間計算量 (2/2)

- レコード追加に要する時間は  $O(1)$ 
  - $i$  番目のレコードを読み書きする時間は,  $i$  に依存しないと仮定
- レコード削除に要する時間は  $O(N)$ 
  - 削除するレコードを探す手間
    - キーを指定して探すなら  $O(N)$
    - 番号を指定して探すなら  $O(1)$
  - 空いた隙間を埋める手間
    - 最大  $N-1$  個のレコードを移動すると  $O(N)$
    - 末尾のレコードで埋めれば  $O(1)$

# 2分探索 (1/3)

- 探索の高速化のため, 1回の比較で探索範囲をなるべく絞り込みたい
  - 線形探索では, 1回の比較で, 最悪レコード1個分しか探索範囲が減らない
  - 2分探索では, 1, 2回の比較で, 探索範囲が半減
- 2分探索は, レコードがキーに関してソートされた配列を対象とする



# 2分探索 (2/3)

- 探索時間は  $O(\log N)$
- ソートすることが前提

```
int low = 0;
int high = size;
while (low < high) {
    int mid = (low + high)/2;
    if (table[mid].key が求めるキー) {
        探索成功; 終了;
    } else if (table[mid].key が求めるキーより大) {
        high = mid;
    } else {
        low = mid + 1;
    }
}
```

# 2分探索 (3/3)

- レコード追加に要する時間は  $O(N)$ 
  - 挿入位置を探す手間
    - 2分探索のアイデアを使って探せば  $O(\log N)$
  - 隙間を作るためにレコードを移動する手間
    - 最大  $N-1$  個のレコードを移動する必要があり  $O(N)$
- レコード削除に要する時間も  $O(N)$ 
  - 削除するレコードを探す手間
    - キーを指定して探すなら  $O(\log N)$
    - 番号を指定して探すなら  $O(1)$
  - 空いた隙間を埋める手間
    - $O(N)$

# コンピュータで扱える データと計算の量 (1/2)

- 今日の下限に近いもの: 演習室の iMac
  - メインメモリ: 2.5GB (?)
    - 32ビット整数で6.7億個強(=  $2.5 \times 2^{30} \div 4$ )
    - 一般家庭用PCよりは多い
  - クロックサイクル: 2GHz
    - (調子が良いとき)毎秒数十億機械語命令を処理
    - 四則演算性能は, 地球上の全人類が算盤の修行をし, 束になってかかった場合, どちらが勝つか微妙
- 今日の上限に近いもの: TSUBAME
  - 総メモリサイズ, 総演算性能ともに(非常に大雑把だが) iMac の1万倍くらい

# コンピュータで扱える データと計算の量 (2/2)

- 全国民データベースを作る場合
  - 1人あたり日本語1000文字程度の情報を格納する  
なら, 約 $1.2\text{億} \times 1000 \times 2 = 2400\text{億}$ バイト程度
    - iMac のメインメモリの100倍くらい
    - ディスク1本に収まる分量
- 全地球人の顔写真データベースを作る場合
  - 人数が日本人の50倍程度, 顔写真のデータ量が1  
人2MBとすると, 上の例の1000倍程度
    - iMac のメインメモリの500万倍くらい
    - 1TB のディスクで2万本以上
    - TSUBAME のディスクストレージの10倍以上

# 探索の計算量 (1/2)

- $N = 100$ 万の場合
  - 普通のパソコンのメモリに納まる可能性が高い
    - 1レコードが100バイトなら、全体で100MB程度
  - 1回の探索なら、線形探索でも、普通のパソコンで一瞬の可能性が高い
  - 100万回探索を行うと大きな差が出る
    - 平均で、線形探索なら  $5 \times 10^{11}$  回程度の繰り返し
    - 2分探索なら  $2 \times 10^7$  回程度の繰り返し
    - もちろん、線形探索と2分探索で1回の繰り返しの内容は異なる

# 探索の計算量 (2/2)

- コンピュータの性能が向上するとどうなるか？
  - 問題のサイズが固定なら、計算時間は短くなる
    - 線形探索と2分探索の差が縮まる(比は縮まらない)
    - e.g. 10秒 vs. 1ミリ秒  $\Rightarrow$  10ミリ秒 vs. 1マイクロ秒
  - より大きな問題を解きたくなるのが人情
    - cf. 人間の欲望には限りがない
    - $N$ が大きくなると、線形探索と2分探索の比が大きくなる
    - e.g.  $5 \times 10^{11}$ 回 vs.  $2 \times 10^7$ 回  $\Rightarrow$   $5 \times 10^{17}$ 回 vs.  $3 \times 10^{10}$ 回
- コンピュータの性能が向上すると、速い(オーダーの低い)アルゴリズムが求められるようになる