データ構造とアルゴリズム　第２回

1. アルゴリズムの重要性
2. 探索問題
3. 基本的なデータ構造
4. 動的探索問題とデータ構造
5. データの整列
6. グラフのアルゴリズム
7. 文字列のアルゴリズム
8. アルゴリズム設計手法

第２章　探索問題

2.1　探索問題とは
2.2　逐次探索の効率
2.3　順序関係を利用した探索
2.4　m-ブロック法
2.5　２分探索法
2.6　ハッシュ法

今日の学習目標

- 探索問題とは何か、応用例を用いて説明できる
- 探索アルゴリズムを説明できる
  - 逐次探索、m-ブロック法、２分探索法、ハッシュ法
- 探索アルゴリズムの計算時間を説明できる
- 探索アルゴリズムを用いたプログラムを書ける

2.1 探索問題とは

- 探索問題
  - 数値データの集合 S が与えられているときに，S の中からデータ x を探す
    （x が S にない場合，ないことが分かる）
- 探索問題の例
  - 顧客リストから特定の顧客を探す
    氏名，住所，電話番号などから探索
  - ネットワークから特定の情報を探す
    Google, Yahoo などによる検索
    ＊ネットワークでの探索は講義の範囲外

2.1 探索問題とは

- 数値データの集合 S が与えられているときに，
  S の中からデータ x を探す
  （x が S にない場合，ないことが分かる）
- 数値データの集合 S の保管方法
  - この章では**配列 s に格納**
    - 要素の**挿入・削除がない**から
      → 挿入・削除は４章で扱う

s
72
7
6
65
97
9
74
37

第２章　探索問題

逐次探索法

- $s[0..n-1]$：集合 S（n 個のデータ）
- x：探索するデータ
- 逐次探索法

```
xを入力する；
i = 0;
do {
  if (x == s[i]) i を返して終了;
  else i=i+1;
} while (i < n);
-1 を返して終了;
```

s
72
7
6
65
97
9
74
37

65

比較回数：**最小　１回**　　　**最大 n 回**
平均 $(n+1)/2$ 回
x が見つかる場合

逐次探索の平均比較回数

- **成功探索**（x が見つかる場合）
  - S の全要素が同確率で探索されるなら
    $(n+1)/2$ 回
    （$= (1+2+3+\cdots+n)/n$）
- **失敗探索**（x が見つからない場合）
  n 回
- **成功確率 p** の場合
  - S の全要素が同確率で探索されるなら
    $p(n+1)/2+(1-p)n$ 回

番兵：有名な高速化技法（テキスト外）

- 逐次探索の比較回数
  - 最小　１回　　　最大 n 回
  - 平均 $(n+1)/2$ 回

●正確に表現すると，要素の比較回数

✓while の終了判定はカウントしていない

```
xを入力する；
i = 0;
do {
    if (x == s[i]) i を返して終了;
    else i=i+1;
} while (i < n);
-1 を返して終了;
```


番兵：有名な高速化技法（テキスト外）

- 配列を $s[0..n]$ とする（n 個のデータは $s[0..n-1]$ に格納）
  - $s[n]$ に探索データ x を格納（番兵）
  - x の探索は必ず成功（x を $s[i]$ で見つけたとする）
    - $0 \le i \le n-1$ → 探索データ発見（成功探索）
    - $i = n$ → 探索データ発見できず（失敗探索）

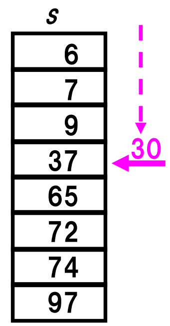
```
xを入力する；
s[n] = x;
i = 0;
while (x != s[i])
    i = i+1;
if (i < n) i を返して終了;
else -1 を返して終了;
```


第２章　探索問題

2.3 順序関係を利用した探索

- $s[0..n-1]$：集合 S（n 個のデータ）
  - 昇順（小さい順）にソート済の場合
- x：探索するデータ
- ソート列上での逐次探索法

```
xを入力する；
i = 0;
do {
    if (s[i] ≧ x) ループから出る;
    else i=i+1;
} while (i < n);
if (s[i] == x) i を返して終了;
else -1 を返して終了;
```



第２章　探索問題

2.4 m-ブロック法

- $s[0..n-1]$：集合 S（n 個のデータ）
  - 昇順（小さい順）にソート済の場合
- x：探索するデータ
- m-ブロック法
  - s を m 個のブロック $B_0, \ldots, B_{m-1}$ に分割
    - B_j：$s[jk\,..\,(j+1)k-1]$　（$k = n/m$）
  - $B_0, \ldots, B_{m-2}$ の最大値 $s[(j+1)k-1]$ と x を順に比較（逐次探索）→ x が存在するブロックを決定
  - $x \le s[(j+1)k-1]$ なる最初のブロック B_j（このような j がなければ B_{m-1}）を逐次探索

単純な逐次探索を２回繰返すだけで計算時間は大きく短縮

2.4 m-ブロック法

- $s[0..n-1]$：集合 S（n 個のデータ）
  - 昇順（小さい順）にソート済の場合

38 の探索（$n=13$）

6	
7	
9	
17	
22	①
26	③
31	④
38	⑤
46	
55	②
74	
81	
97	

- $m=3$ 個のブロックに分割
- ブロックの最後の要素（ブロックの最大要素）と順に比較
- 探すべきブロックがわかれば そのブロックを逐次探索

m-ブロック法　アルゴリズム2.3

- アルゴリズム2.3：m-ブロック法

```
//ステップ１：xを含むブロックの逐次探索
  j = 0;
  while (j ≦ m-2)
    if (x ≦ s[(j+1)k-1]) ループから出る;
    else j = j+1;
//ステップ２：ブロック内での逐次探索
  i = jk; t = min {(j+1)k-1, n-1};
  while (i < t)
    if (x ≦ s[i]) ループから出る;
    else i = i+1;
  if (x == s[i]) i を返して終了;
  else -1を返して終了;
```

m-ブロック法の比較回数

- 最大比較回数

  $m-1+\left\lceil \frac{n}{m} \right\rceil < m+\frac{n}{m}$ 回

  $\lceil x \rceil$：x の小数点以下の切上げ（x 以上の最小の整数）

  （参考）$\lfloor x \rfloor$：x の小数点以下の切捨て

- 最適なブロックサイズ
  - $m=\sqrt{n}$ のとき 最大比較回数は $2\sqrt{n}$

    厳密には，n が平方数の場合の評価

    n が平方数でない場合も同様

第２章　探索問題

2.5 ２分探索法

- $s[0..n-1]$：集合 S（n 個のデータ）
  - **昇順（小さい順）にソート済**の場合
- x：探索するデータ
- ２分探索法の基本アイデア（$s[left..right]$ に x が存在）
  - 初期化：$left=0, right=n-1$

    探索範囲 $s[left..right]$
  - $s[left] \le x \le s[right]$ なら $mid=(left+right)/2$
    - x と $s[mid]$ を比較
      - $x < s[mid]$ なら $s[left..mid-1]$ を探索
      - $x = s[mid]$ なら 探索終了
      - $x > s[mid]$ なら $s[mid+1..right]$ を探索

２分探索法　アルゴリズム2.7 (1)

- アルゴリズム2.7：２分探索法 (4)
  - 最終版：$left$, $right$ の扱いがトリッキーな方法

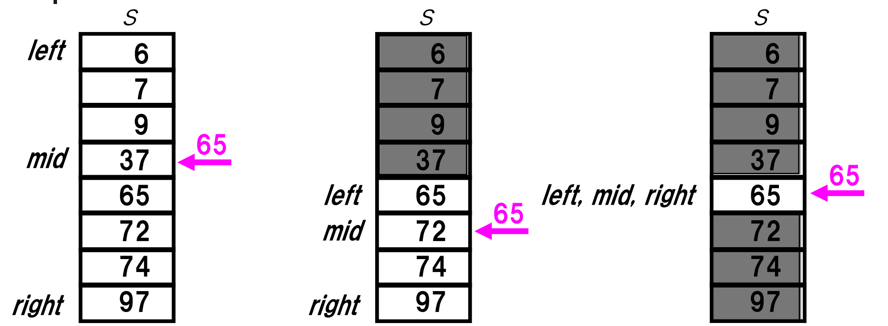
```
x を入力する;
if (x ＜ s[0] または x ＞ s[n-1]) -1を返して終了;
left = 0; right = n-1; //探索区間の設定
do {
    mid = (left + right)/2; //探索区間の中央
    if (x ＜ s[mid]) right = mid-1;
    else left = mid+1;
    } while (left ≦ right);
if (x == s[right]) then right を返して終了;
else -1 を返して終了;
```

２分探索法　アルゴリズム2.7 (2)

```
do {
    mid = (left + right) / 2; //探索区間の中央
    if (x < s[mid]) right = mid-1;
    else left = mid+1;
    } while (left ≦ right);
```

- **ループ不変式**：**ループ内で常に成立する性質**
  - x **が** S **に存在するなら，** $s[left-1] \le x \le s[right]$
- **ループ終了時**：$right = left - 1$
  - x **が** S **に存在するなら，** $x = s[right]$

２分探索法　アルゴリズム2.7　実行例




```
do {
    mid = (left + right) / 2;
    if (x < s[mid]) right = mid-1;
    else left = mid+1;
    } while (left ≦ right);
```

$x = s[mid]$ **なので，**
$left = mid + 1$ **を実行**
$left > right$ **となって**
while 文から脱出
$x = s[right]$ **が成立**

２分探索法　比較回数

- **探索範囲**
  - $s[left\,..right]$ **：サイズ** $right - left + 1$
  - **実行開始時**
    - $s[0\,..n-1]$ **：サイズ** n
- **比較するごとに探索範囲のサイズは半減**
  探索範囲のサイズが１になるまで
  - **比較回数：最大** $\log_2 n + 1 = O(\log n)$ **回**

mーブロック法と２分探索

- $m = 2$ **の場合の** m**-ブロック法（**$k = n/m = n/2$**）**

```
//ステップ１：xを含むブロックの探索
  j = 0;
  if (x ≦ s[k-1]) ループから出る;
     else j = j+1;
//ステップ２：ブロック内での探索
  i = jk; t = min {(j+1)k-1, n-1};
  while (i < t)
     if (x ≦ s[i]) ループから出る;
     else i = i+1;
  if (x == s[i]) i を返して終了;
  else -1を返して終了;
```

ブロック B_0 **か** B_1 **を選択**

ブロック内の選択

逐次探索
⇩
再帰的に ステップ１を実行

⇒ **２分探索**

第２章　探索問題

2.6 ハッシュ法

- 逐次探索，m-ブロック法，２分探索
  - 大小比較を基礎とする方法（比較法）
  - x に近い値を求めることにも拡張可能
  - ２分探索：比較 $O(\log n)$ 回　（最適）
- ハッシュ法
  - 大小比較以外の方法を利用
  - x に近い値を求めることは一般に不可能
  - 比較：平均 $O(1)$ 回　（$O(1)=$ 定数）

ここでは説明しない

ハッシュ法

ハッシュ表とよぶ

- ハッシュ法：基本アイデア
  - サイズ m（n の1.5～2倍）の配列 htb を利用
  - ハッシュ関数 h：
    データ定義域 → $\{0, 1, \ldots, m-1\}$
  - データ x は $htb[h(x)]$ に格納
  - データ x の探索は $htb[h(x)]$ を参照

ハッシュ法：データの衝突

- ハッシュ関数 h：データ定義域 → $\{0, 1, \ldots, m-1\}$
- データ x は $htb[h(x)]$ に格納
- データ定義域のサイズ $> m$ なら（よくあること）
  - $h(x) = h(y)$ なる $x \neq y$ が存在
    - ハッシュ関数値の衝突
    - x，y を同時に $htb[h(x)](= htb[h(y)])$ に格納できない
      - データの衝突

ハッシュ法：データ衝突時の処理

- データ x **格納時**
  - $htb[h(x)]$ に他データが存在（**データの衝突**）
    - $\Rightarrow$ $htb[h(x)+1], htb[h(x)+2], \ldots$ を順に調べ，
    - **最初の空の場所**に格納
    - $htb[m-1]$ の次は $htb[0]$ に戻る
- データ x **探索時**
  - $htb[h(x)]$ に他データが存在（**データの衝突**）
    - $\Rightarrow$ $htb[h(x)+1], s[h(x)+2], \ldots$ を
    - x **か空の場所**を見つけるまで探索
    - $htb[m-1]$ の次は $htb[0]$ に戻る

ハッシュ法：データ格納 アルゴリズム2.8

- ハッシュ法でデータをハッシュ表に蓄える手続き
  - アルゴリズム2.8

仮定：格納データは 0 でない
0 はその場所が空ということ

```
ハッシュ表 htb[0] 〜 htb[m-1] の内容を 0 に初期化;
for (i=0 to n-1) do {
   i 番目のデータを x とする;
   j = hash(x); //ハッシュ値を計算
   while (htb[j] != 0) //ハッシュ表で空いている場所を探す
      j = (j+1) % m; //次の場所へ移動
   htb[j] = x; //最初の空き場所に x を格納
}
```


ハッシュ法：データ格納の例

$h(x) = x \bmod 11$

25 の挿入

0	
1	
2	
3	3
4	15
5	5
6	25
7	7
8	
9	20
10	

挿入： 5 7 15
20 3 25

ハッシュ法：データ探索 アルゴリズム2.9

- 与えられたデータを探索する手続き
  - アルゴリズム2.9

```
探索すべきデータ x を入力する;
j = hash(x);
while (htb[j] != 0 かつ htb[j] != x)
   j = (j+1) % m; //次の場所へ移動
if (htb[j] == x) j を返して終了;
else -1 を返して終了;
```

ハッシュ法：データ探索の例

$h(x) = x \bmod 11$

② ２５の探索

③ ６の探索

① ２０の探索

0	
1	
2	
3	３
4	１５
5	５
6	２５
7	７
8	
9	２０
10	

ハッシュ法の比較回数

- ハッシュ法の比較回数
  - 占有率 $\alpha = n/m$ に依存
    - ハッシュ表のデータが格納されている割合
  - 平均成功探索回数

    $(1 + 1/(1-\alpha))/2$
    - $\alpha = 1/2$ なら $3/2$ 回
  - 平均失敗探索回数

    $(1 + 1/(1-\alpha)^2)/2$
    - $\alpha = 1/2$ なら $5/2$ 回

今日のまとめ　第２章　探索問題

2.1　探索問題とは

2.2　逐次探索の効率

2.3　順序関係を利用した探索

2.4　m-ブロック法

2.5　２分探索法

2.6　ハッシュ法

今日の学習目標（振返り）

- 探索問題とは何か、応用例を用いて説明できる
- 探索アルゴリズムを説明できる
  - 逐次探索、m-ブロック法、２分探索法、ハッシュ法
- 探索アルゴリズムの計算時間を説明できる
- 探索アルゴリズムを用いたプログラムを書ける