

プログラミング応用

<http://bit.ly/ouyou3d>

アルゴリズム (2)

前期 第14週

2018/7/24

本日は・・・

- ▶ 探索アルゴリズムの高速化
- ▶ 線形探索 + 番兵法
- ▶ 二分探索

第13週より

線形探索

「最初に34度になるのはいつ？」

① この値は34？

② 配列の末尾？ → 次に進む

③ この値は34？

④ 配列の末尾？ → 次に進む

7/17	7/18	7/19	7/20	7/21	7/22	7/23
36	35	34	34	35	34	34

⑤ この値は34？ → 探索終了

(もし見つからずにここまで到達したら…)

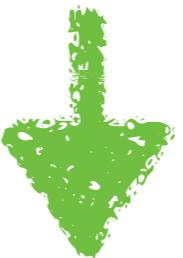
配列の末尾？ → 探索終了

本日は・・・

- ▶ 探索アルゴリズムの高速化
- ▶ 線形探索 + 番兵法
- ▶ 二分探索

線形探索の高速化①

比較回数を少しでも減らして
高速化できないか？



線形探索 + 番兵法による高速化

番兵法とは

配列末尾に条件を満たすデータを
追加することで高化する手法

7/17	7/18	7/19	7/20	7/21	7/22	7/23	
36	35	34	34	35	34	34	34

「最初に34度になる日」を見つける場合は、配列の末尾に
34度のデータを追加する（日付は不要）

線形探索 + 番兵法

値が見つかった場合

① この値は34？

② 配列の末尾？ → 次に進む 番兵法ではこの比較が不要になる

7/17	7/18	7/19	7/20	7/21	7/22	7/23	
36	35	34	34	35	34	34	34

⑤ この値は34？

⑥ 配列の末尾？ → 末尾でないので「見つかった」

③ この値は34？

④ 配列の末尾？ → 次に進む

番兵法では見つかった時に末尾の判定をする

番兵法ではこの比較が不要になる

線形探索 + 番兵法

値が見つからなかった場合

① この値は34？

② 配列の末尾？ → 次に進む 番兵法ではこの比較が不要になる

7/17	7/18	7/19	7/20	7/21	7/22	7/23	
36	35	34	34	35	34	34	34

この値は34？
配列の末尾？ → 末尾なので「見つからなかった」

③ この値は34？

④ 配列の末尾？ → 次に進む 番兵法ではこの比較が不要になる

線形探索の高速化①

比較回数を少しでも減らして
高速化できないか？



線形探索 + 番兵法による高速化



条件を満たすデータを見つけたときのみ
末尾であるか調べるので速い

線形探索 (13_search.c)

```
//引数nと一致する値を線形探索で見つける
void search(char *label[], int value[], int n, int size)
{
    int i = 0;
    while(1) {
        if(value[i]==n) {
            printf("label: %s, ", label[i]);
            printf("value: %d\n", value[i]);
            break;
        }
        if(i==size-1) {
            printf("見つかりませんでした\n");
            break;
        }
        i++;
    }
}
```

i番目の値はnと一致する？

配列の末尾？

breakで探索を終了する

```
graph TD; A[if(value[i]==n)] --> B["i番目の値はnと一致する？"]; C[if(i==size-1)] --> D["配列の末尾？"]; E[break] --> F["breakで探索を終了する"]
```

線形探索 + 番兵法 (14_banpei.c)

//引数nと一致する値を線形探索+番兵法で見つける

```
void banpei(char *label[], int value[], int n, int size)
{
    int i = 0;
    while(1) {
        if(value[i]==n) {
            if(i != size) {
                printf("label: %s, ", label[i]);
                printf("value: %d\n", value[i]);
            } else {
                printf("見つかりませんでした\n");
            }
            break;
        }
        i++;
    }
}
```

i番目の値はnと一致する？

見つかったのは配列の末尾ではない？

このif文は一致する値が見つかった時のみ処理される

本日は・・・

- ▶ 探索アルゴリズムの高速化
- ▶ 線形探索 + 番兵法
- ▶ 二分探索

線形探索（+番兵法）の利点と欠点

▶ 利点

▶ もっとも単純な探索アルゴリズム→実装が簡単

▶ 欠点

▶ 先頭から調べていくので、条件を満たすデータが
末尾の方にあると探索に時間がかかる

7/17	7/18	7/19	7/20	7/21	7/22	7/23
36	35	34	34	35	34	34

先頭で見つかった場合は比較回数が1回

末尾で見つかった場合は比較回数が7回

→ 配列のサイズが10000の場合は10000回!?

線形探索の高速化②

末尾にある値を見つける比較回数を減らして
高速化できないか？



二分探索による高速化

二分探索とは

データが既に昇順または降順に並べ替えられて
いるときに、探索を高化できる手法

昇順

7/19	7/20	7/22	7/23	7/18	7/21	7/17
34	34	34	34	35	35	36

降順

7/17	7/18	7/21	7/19	7/20	7/22	7/23
36	35	35	34	34	34	34

二分探索

次のような配列から 16 を探索する

データが区別できるようにラベルを付けてある

配列の添字

	0	1	2	3	4	5	6
label	num0	num1	num2	num3	num4	num5	num6
value	5	9	13	14	16	17	20

あらかじめ 昇順 に並んでいる

- ▶ 真ん中の要素の値と、見つけたい値を比較
- ▶ 「真ん中の要素の値 > 見つけたい値」 → 真ん中より 左側 にある
- ▶ 「真ん中の要素の値 < 見つけたい値」 → 真ん中より 右側 にある
- ▶ 上記の探索する範囲を半分に絞り込んでいく処理を繰り返す

二分探索

次のような配列から 16 を探索する

16 を 0~6 番目の中から探す (探索する範囲を赤字で示す)

0 1 2 3 4 5 6

label	num0	num1	num2	num3	num4	num5	num6
value	5	9	13	14	16	17	20

- ① 真ん中の値に着目 (3番目の値)
- ② 「14 < 16」 → 右側 (4~6番目) にあるはず

二分探索

次のような配列から 16 を探索する

16を4～6番目の中から探す

	0	1	2	3	4	5	6
label	num0	num1	num2	num3	num4	num5	num6
value	5	9	13	14	16	17	20

③ 真ん中の値に着目 (5番目の値)

④ 「17 > 16」 → 左側 (4番目) にあるはず

二分探索

次のような配列から 16 を探索する

16を4番目の中から探す

	0	1	2	3	4	5	6
label	num0	num1	num2	num3	num4	num5	num6
value	5	9	13	14	16	17	20

⑤ 真ん中の値に着目 (4番目の値)

⑥ 「16 == 16」 → 4番目に見つかったので探索終了

二分探索をCで実装する

探索する範囲を指定するために、

以下の変数を用意する

- ▶ **left** … 探索範囲の左端の添字
- ▶ **right** … 探索範囲の右端の添字
- ▶ **center** … 探索範囲の中央の添字

線形探索と同様に以下の変数も使う

- ▶ **label** … ラベルの配列
- ▶ **value** … データの配列
- ▶ **n** … 見つけたい値

二分探索をCで実装する

先程のアルゴリズムを変数を使って表すと…

- ▶ 真ん中の要素の値と、見つけたい値を比較
- ▶ 「真ん中の要素の値 > 見つけたい値」 → 真ん中より左側にある
- ▶ 「真ん中の要素の値 < 見つけたい値」 → 真ん中より右側にある
- ▶ 上記の探索する範囲を半分に絞り込んでいく処理を繰り返す



- ▶ 真ん中の要素の添字を求める
 $center = (left + right)/2$
- ▶ `value[center]`と、`n`を比較
 - ▶ 「`value[center] > n`」 → `left ~ center-1` の範囲にある
 - ▶ 「`value[center] < n`」 → `center+1 ~ right` の範囲にある
- ▶ `left, right`を更新して範囲を絞みながら処理を繰り返す

二分探索をCで実装する

16を0~6番目の中から探す (探索する範囲を赤字で示す)

▶ left = 0
▶ right = 6
▶ center = (left + right)/2 = 3

0 1 2 3 4 5 6

label	num0	num1	num2	num3	num4	num5	num6
value	5	9	13	14	16	17	20

- ① 真ん中の値に着目 (3番目の値)
② 「14 < 16」 → 右側 (4~6番目) にあるはず

▶ left = center+1 = 4
▶ right = 6

二分探索をCで実装する

▶ left = 4
▶ right = 6
▶ center = (left + right)/2 = 5

16を4~6番目の中から探す

0 1 2 3 4 5 6

label	num0	num1	num2	num3	num4	num5	num6
value	5	9	13	14	16	17	20

- ③ 真ん中の値に着目 (5番目の値)
④ 「17 > 16」 → 左側 (4番目) にあるはず

▶ left = 4
▶ right = center-1 = 4

二分探索をCで実装する

▶ left = 4
▶ right = 4
▶ center = (left + right)/2 = 4

16を4番目の中から探す

0 1 2 3 4 5 6

label	num0	num1	num2	num3	num4	num5	num6
value	5	9	13	14	16	17	20

⑤ 真ん中の値に着目 (4番目の値)

⑥ 「16 == 16」 → 4番目に見つかったので探索終了

二分探索をCで実装する

▶ 「見つからなかった」の判定は？

leftとrightを更新し続けて、「**left > right**」となっ
た時点で「**探索する範囲がもうない**」ため、
「見つからなかった」として終了する

▶ 「left+right」が奇数の時の真ん中は？

真ん中の要素はないため、**前の方の要素**に着目する
【例】 leftが3、rightが6の場合、centerは4
(int型の計算で、**(left+right)/2=4**となる)

二分探索 (14_binary.c)

//引数nと一致する値を二分探索で見つける

```
void binary(char *label[], int value[], int n, int left, int right)
{
    int center;
    while(1) {
        if(left>right) {
            printf("見つかりませんでした\n");
            return;
        }
        center = (left+right)/2;
        if(value[center]==n) {
            printf("label: %s, ", label[center]);
            printf("value: %d\n", value[center]);
            return;
        }
        if(value[center]>n) {
            right = center-1;
        }
        if(value[center]<n) {
            left = center+1;
        }
    }
}
```

見つからなかった場合

中央の添字を計算

見つかった場合

中央よりも左側にある場合

中央よりも右側にある場合

【練習14-1】

「14_banpei.c」 のサンプルプログラムをコンパイルして、**線形構造+番兵法の出力**の実行結果を確認しましょう。ただし、この例は「見つからなかった」場合の処理は正常に動作しません。（以下参考）

[実行結果]

label: 7/19, value: 34

(← 見つかった場合)

Segmentation fault: 11

(← 見つからなかった場合)

【練習14-2】

「14_binary.c」のサンプルプログラムをコンパイルして、**二分探索**の実行結果を確認しましょう。

【課題14-1】

練習14-1の「14_banpei.c」のサンプルプログラムで、「見つからなかった」時も動作するように、mainに代入処理を追加してください。（以下参考）

[mainの処理]

```
banpei(dates1, temperature1, 34, N);
```

//ここで配列の末尾に29を代入する

```
banpei(dates1, temperature1, 29, N);
```

[実行結果]

label: 7/19, value: 34

(← 見つかった場合)

見つかりませんでした

(← 見つからなかった場合)

【課題14-2】

「14_binary.c」の関数binary()を参考に、**再帰処理**で二分探索する関数binary2()を作成してください。
(変更点は以下の通り)

- ▶ while文による繰り返し処理を削除する
- ▶ value[center]>n の場合、次のrightが「center-1」になるように再帰呼び出しをする
- ▶ value[center]<n の場合、次のleftが「center+1」になるように再帰呼び出しをする

【課題14-2】

[mainでの処理]

```
char *l[N] = {"num0", "num1", "num2", "num3",
              "num4", "num5", "num6"};
int v[N] = {5, 9, 13, 14, 16, 17, 20};

binary2(l, v, 5, 0, N-1);
binary2(l, v, 20, 0, N-1);
binary2(l, v, 10, 0, N-1);
```

[実行結果]

```
label: num0, value: 5
label: num6, value: 20
見つかりませんでした
```

【課題14-3】

「14_binary.c」の関数binary()を参考に、**降順**で
あらかじめ並んでいる配列を二分探索する関数
binary3()を作成してください。（以下参考）

- ▶ value[center]>n の場合、真ん中よりも**右側**を探索するよう
に変更する
- ▶ value[center]<n の場合、真ん中よりも**左側**を探索するよう
に変更する

【課題14-3】

[mainでの処理]

```
char *l[N] = {"num0", "num1", "num2", "num3",
              "num4", "num5", "num6"};
int v2[N] = {20, 17, 16, 14, 13, 9, 5};

binary3(l, v2, 5, 0, N-1);
binary3(l, v2, 20, 0, N-1);
binary3(l, v2, 10, 0, N-1);
```

[実行結果]

```
label: num6, value: 5
label: num0, value: 20
見つかりませんでした
```

【課題14-4】

線形探索と二分探索で、「配列の最後で見つかった」場合のプログラムを作成し、実行時間を比較してください。（作成手順は以下を参照）

- ① 講義資料のページから「myarray.h」をダウンロードする
(10000個の要素が昇順に格納された配列v3とそのラベルl3が入っている)

- ② 14_binary.cに以下のincludeを追加

```
#include <stdio.h>
#include "myarray.h" //上記の配列v3とl3の宣言が読み込まれる
```

(以下省略)

【課題14-4】

③ 14_binary.cのmainを以下のようにする

```
int main(void)
{
    int i;
    //配列の最後で見つかる二分探索を100回繰り返す

    for(i=0; i<100; i++) {
        printf("%6d: ", i);
        binary(13, v3, 30131, 0, 9999);
    }
    return 0;
}
```

④ プログラムをコンパイルして、timeコマンドで実行する

```
$ cc 14_binary.c
$ time ./a.out
(二分探索を100回繰り返した結果と時間が表示される)
```

【課題14-4】

⑤ 13_search.cに対しても、②～④と同様の作業をして、時間を計測する

```
#include <stdio.h>
#include "myarray.h" //上記の配列v3と13の宣言が読み込まれる
(途中省略)

int main(void)
{
    int i;
    //配列の最後で見つかる線形探索を100回繰り返す

    for(i=0; i<100; i++) {
        printf("%6d: ", i);
        search(13, v3, 30131, 10000);
    }
    return 0;
}
```

```
$ cc 13_search.c
$ time ./a.out
(線形探索を100回繰り返した結果と時間が表示される)
```

【課題14-4】

mainで繰り返している探索の数を、**100回から増やしてみて**、それぞれどのように探索時間に差が出てくるのか試してください。

- ▶ 1000回では？
- ▶ 10000回では？
- ▶ 100000回では？
- ▶ それ以上は？

【課題14-5】

線形探索と二分探索で、「配列の先頭で見つかった」場合のプログラムを作成し、実行時間を比較してください。

(課題14-4で作ったmainで、v3の0番目の値を探索するように変更してみる)

※ 課題14-4に比べ、線形探索は速くなるが、二分探索は変わらない

まだ余裕のある人は…

【課題14-6】

「14_banpei.c」についても、課題14-4と課題14-5のように探索時間を計測して、線形探索と二分探索と比較してください。