

## アルゴリズムとデータ構造 2024 第3回 演習課題 「2分探索法と計算量」

1. 要素が小さい順に並んでいる配列 `data` に対して、次の手順で値 `key` を探索し、その位置（添字）を返すメソッドを作成せよ。さらに、適切なデータで動作を確認せよ。このアルゴリズムを **2分探索** という。
  - (1) 配列（探索範囲）の中央の値と `key` を比較し、もし等しければ見つかったのでその位置を返す。
  - (2) そうでなければ、大小関係から `key` が配列（探索範囲）の前半と後半のどちらに含まれるか判定し、新しい探索範囲を前半または後半に設定する。
  - (3) (1)~(2)の手順を繰り返し、探索範囲を次々に半分に絞り込んでいき、要素がなくなるまで続ける。

```
public static int binarySearch(int key, int[] data) {  
  
    int left = 0; // 探索範囲の左端  
    int right = data.length - 1; // 探索範囲の右端  
  
    while (left <= right) { // 探索範囲が1個以上である間繰り返す  
  
        int mid =  
  
        if (key == data[mid]) {  
  
        } else if (key < data[mid]) {  
  
        } else {  
  
        }  
    }  
    return -1; // 発見できなかった場合は-1を返す  
}
```

2. 下記は、1.と同様の処理を**再帰**を用いて記述したものである。ただし 1.とは引数が異なり、配列の中の `data[first]~data[last]`の範囲から値 `key` を探索する。プログラムを完成させ、再帰の仕組みを理解せよ。

```
public static int binarySearchR(int key, int[] data, int first, int last) {  
  
    if ( ) { // 探索範囲がなければ終了する  
  
        return -1;  
    }  
  
    int mid =  
  
    if (key == data[mid]) {  
  
        return  
    }  
  
    if (key < data[mid])  
        return binarySearchR(key, data, first, );  
    /* else */  
        return binarySearchR(key, data, , last);  
}
```

3. アルファベット順に並んだ文字列に、1.と同様のアルゴリズムを適用するプログラムを完成させよ。なお、Java ではクラス型に關係演算子(不等号)が適用できないが、compareTo メソッドがその代わりになる。

```
public static int binarySearch(String key, String[] data) {  
  
    int left = 0;  
    int right = data.length - 1;  
  
    while (left <= right) {  
  
        int mid =  
  
        int comp = key.compareTo(        );  
  
        if (comp ==        ) {  
  
        } else if (comp <        ) {  
  
        } else {  
  
        }  
    }  
    return -1;  
}
```

4. 2分探索の最大計算量(最悪計算量)を考えてみる。配列の要素数を  $n$  とする。
- (1) 最初の探索範囲には  $n$  個の要素がある。ループの処理を 1 回行うごとに探索範囲はほぼ  $\alpha$  分の 1 に狭まる。この  $\alpha$  の値を示し、 $i$  回目のループ後の探索範囲のおよその要素数を  $n$  と  $i$  の式で表せ。
  - (2) 最悪の場合のループ回数を  $k$  回とすると、 $k-1$  回目に探索範囲が最後の 1 個になって  $k$  回目に最後の比較をするので、およそ  $n \cdot \left(\frac{1}{\alpha}\right)^{k-1} = 1$  という式が成り立つ。この式から最大計算量  $k$  を求め、 $n$  を横軸としたグラフを描いて線形探索の最大計算量のグラフと比較せよ。
5. 2分探索の平均計算量を考えてみる。ここでは、扱いやすい  $n = 2^k - 1$  の場合について考える。
- (1) もし  $key$  が配列の中央にあれば 1 回目で見つけることができる。よって、1 回目で見つけることができる要素は 1 個である。以下、2 回目で見つけることができる要素は 2 個、3 回目は 4 個…と続く。では、最後の  $k$  回目までかかる要素は何個か示せ。

- (2) 平均計算量はループ回数の期待値に比例し、それは以下の式で求められる理由を考えよ。

$$C = \frac{1}{n} [1 \cdot 1 + 2 \cdot 2 + 3 \cdot 4 + \dots + (k-1) \cdot 2^{k-2} + k \cdot 2^{k-1}]$$

- (3) この値は、 $2C - C$  を計算することによって求めることができる。 $C$  を  $n$  だけの式で表せ。

$$C = 2C - C = \frac{1}{n} [k \cdot 2^k - (1 + 2 + 2^2 + \dots + 2^{k-1})] = \dots$$

6. 【発展】下記のプログラムは、2分探索を改良した内挿探索（補間探索）と呼ばれるものである。これは、配列内で添字が `left` から `right` まで増えるにつれて、値が `data[left]` から `data[right]` までほぼ一定の増加率で増えると仮定すると、探索値 `key` は次の比例式の `mid` 付近にあるだろうという推測に基づく。

$$mid - left : right - left = key - data[left] : data[right] - data[left]$$

$$\frac{mid - left}{right - left} = \frac{key - data[left]}{data[right] - data[left]}$$

$$mid = \frac{key - data[left]}{data[right] - data[left]} (right - left) + left$$

プログラムを完成させて実行し、動作を確認せよ。内挿探索は、通常は非常に高速だが、データの値の分布によっては遅くなってしまふ。それはどのような場合か考察せよ。

```
public static int interpolationSearch(int key, int[] data) {  
  
    int left = 0; // 探索範囲の左端  
    int right = data.length - 1; // 探索範囲の右端  
  
    while (left <= right) {  
        int ldata = data[left];  
        int rdata = data[right];  
  
        // mid の計算式で分母が 0 にならないようにする（重複値対策）  
        if (ldata == rdata) {  
            if (key == ldata) return left;  
            /* else */ break;  
        }  
  
        int mid =  
  
        if (key == data[mid]) {  
            return mid;  
        } else if (key < data[mid]) {  
            right = mid - 1;  
        } else {  
            left = mid + 1;  
        }  
    }  
    return -1;  
}
```