探索

探索とはデータの集合から目的とする値をもった要素を探し出すことである．探索には多数のアルゴリズムがあり，配列から探索する場合，以下の探索法がある．

・線形探索：ランダムなデータの並びからの探索
・２分探索：ソートされたデータの並びからの高速な探索
・ハッシュ法：探索だけでなく，追加・削除のいずれも高速
　□チェイン法：同一ハッシュ値のデータを線形リストでつなぐ．
　□オープンアドレス法：衝突時に再ハッシュを行う．

探索には多くの手法があるため，目的・実行速度・対象となるデータ構造によって，アルゴリズムを選択しなくてはならない．探索に加えてデータの追加や削除を頻繁に行う場合には，そのコストを考慮して総合的な評価が必要となってくる．今回は線形探索と２分探索について学び，アルゴリズムを評価するための指標として，計算量とその算出方法についても学ぶ．

探索について

探索において，探し出す値の項目をキーと呼ぶ．例えば，ある住所録が会員番号，氏名，住所，年齢，国籍といったメンバから構成される構造体の配列であるとする．国籍で探索する場合には国籍が，年齢で探索する場合には年齢がキーとなる．キーを用いて探索を行う場合には，以下のような例がある．

■国籍が日本である人を探す　→　キー値と一致することを指定する．
■年齢が 21 歳以上 27 歳未満の人を探す　→　キー値の区間で指定する．
■ある語句と最も発音が似ている名前の人を探す　→　キー値の近接として指定する．

これらは複合的に論理積や論理和で指定する場合も可能であり，例えば「国籍が日本で年齢が 21 歳以上 27 歳未満の人を探す」といった場合もあり得る．

このようにキーはデータの一部であるが，単純な int 型の配列であれば，要素の値がそのままキー値となる（図１参照）．今回はこの単純な int 型の配列によるデータの集合を元に説明を行っていく．

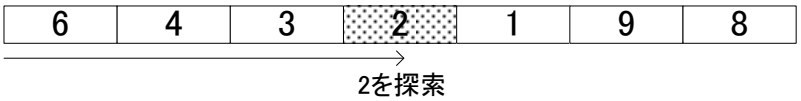

図１　配列からの探索

線形探索－単純な方法

配列からの探索は，目的とするキー値をもつ要素に出会うまで先頭から順になぞっていくことで実現される．これを線形探索あるいは逐次探索と呼ぶ．以下にあるデータが格納された配列からキー値を 2 として行った線形探索の過程を図２に示す．

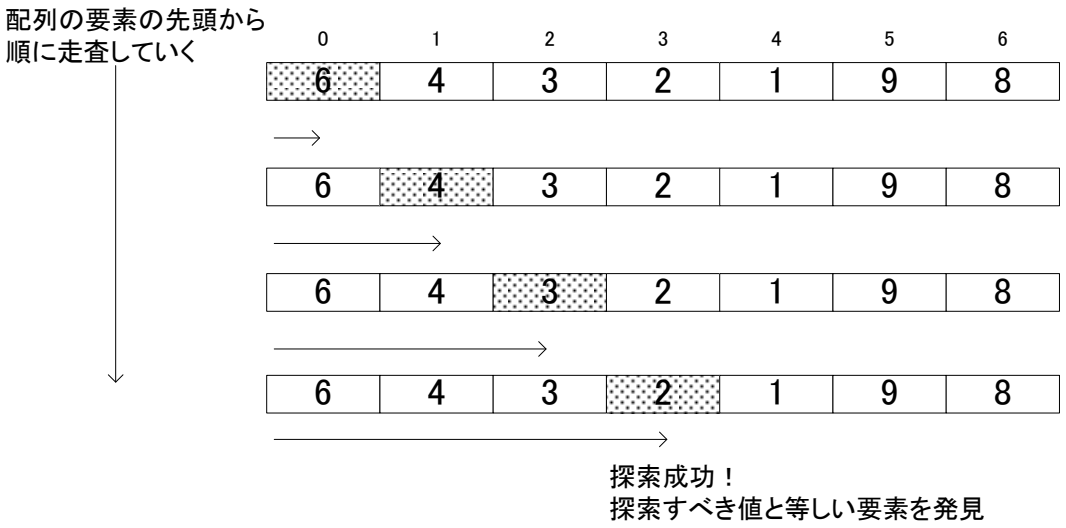

図２　線形探索の一例

このように先頭要素から始めて，目的とする値と等しい要素に出会うまで，一つずつ順に走査していく．この例では 4 番目の要素に達した時点で探索が成功し，処理は終了する．目的とする値と等しい要素が複数ある場合には，配列の先頭側にある要素を見つけた時点で終了することになる．目的とする値が配列中に存在しない場合，探索は失敗したことになる．その際には配列の末端を通り越してしまうまで走査が行われる．

以上のことから，要素を一つずつなぞっていく配列の走査は，以下の 2 つの条件判定の内どちらか一方が成立するまで行えばよいことになる．

(1) 探索すべき値が見つからず末端を通り越した．
(2) 探索すべき値と等しい要素を見つけた．

なお，要素数が n であるとき，(2) の判断のための比較回数は，平均で n / 2 回となる（1 回で終わる場合もあれば，n 回が必要な場合もある）．ただし，配列中に目的とする値が存在しない場合には n + 1 回となる．

さて，線形探索の終了のための条件判定は 2 つであることを確認した．次に示す番兵法では，線形探索の条件判定を 1 つに減らすよう工夫したアルゴリズムである．

線形探索－番兵法

単純な線形探索では，繰り返しのたびに 2 つの終了条件を判定している．手軽な手法とはいえ，探索するデータ数が増えた場合での，計算コストは決して小さいものではない．そこで番兵法では，配列の要素数が実際のデータ数より多く，末尾側に余裕がある場合には，図３に示すように要素の並びの直後に探索すべき値を格納した上で線形探索を行っていく．この代入したデータを番兵という．

すなわち，2 を探索するのであれば，要素の並びの直後に 2 を，5 を探索するのであれば，要素の並びの直後に 5 を代入する．そうすると番兵まで走査が行われれば，探索の成功・失敗にかか

わらず，終了条件（2）が必ず成立する．よって，条件（1）の判定が不要となる．

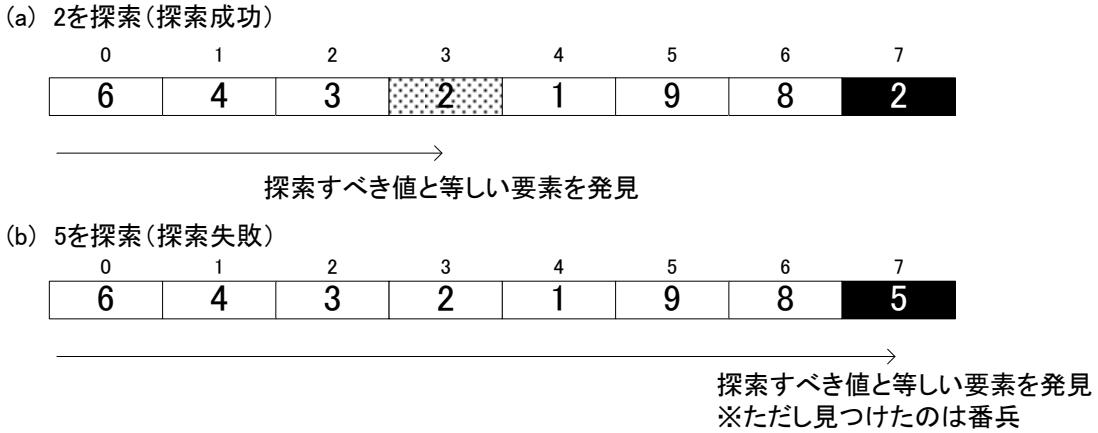

図３　線形探索（番兵法）

番兵を導入することで線形探索アルゴリズムの繰返し終了のための判定回数は半分になる．ただし，繰返しが終了した地点で，本来の配列内のデータを見つけたのか，それとも番兵を見つけたのかの判定が必要である．

２分探索

２分探索は要素があらかじめキーの昇順あるいは降順にソートされている配列から効率よく探索するアルゴリズムである．次に示す 11 個のデータの並びから 39 を探索することを考える．まず，配列の中央の値に着目する．

5	7	15	28	29	31	39	58	68	70	95

目的とする値はこの要素より大きいため，31 より後に存在するはずである．そこで，探索の対象を後側の 5 個に絞り込む．次に，その中央値の 68 に着目する．

5	7	15	28	29	31	39	58	68	70	95

目的とする値はこの要素より小さいため，68 より前に存在するはずである．そこで，探索の対象を前側の 2 個に絞り込む．この 2 個の要素に対して中央値は先頭側の 39 と末尾側の 58 のどちらでも構わない．ここでは 39 とする（整数同士の除算では小数点以下が切り捨てられるため (6+7)/2 が 6 になるので）．

5	7	15	28	29	31	39	58	68	70	95

この値は目的とするキー値と一致するため，探索成功となる．

２分探索のアルゴリズムの一般的な表現について以下のように述べることができる．

◆設定

key ：探索するキーの値
a[n]：要素数 n で昇順にソートされている配列
pl ：探索範囲の先頭の添字　初期値は 0
pr ：探索範囲の末尾の添字　初期値は n-1
pc ：探索範囲の中央の添字　初期値は(n-1) / 2

◆探索方法

・a[pc]を比較して等しければ探索成功

・そうでなければ以下のように探索範囲を縮小する．

■ key ＞ a[pc] のとき
探索範囲を a[pc+1] ～a[pr]に絞り込めるので，pl の値を pc+1 に更新する
■ key ＜ a[pc] のとき
探索範囲を a[pl] ～a[pc-1]に絞り込めるので，pr の値を pc-1 に更新する

・探索範囲の変更に伴い，pc の値を更新する．

◆終了条件

(c1) a[pc]と key が一致した．

(c2) 探索範囲が無くなった．

探索成功時の２分探索の流れを図４に示す．探索対象範囲が半分ずつに減っていく様子が見てとれる．

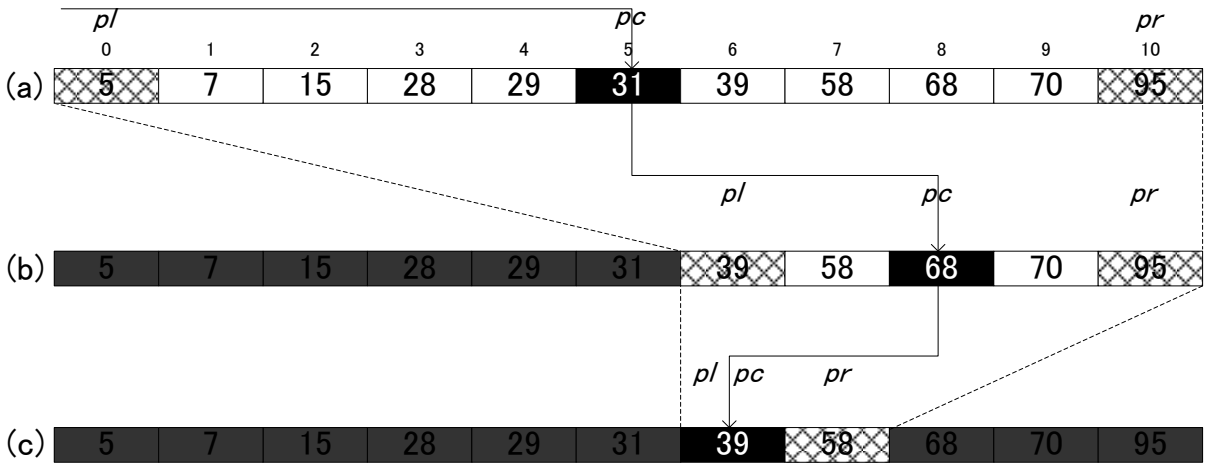

図４　２分探索（探索成功）

次に（c2）の条件が成立して探索に失敗する具体例を考えてみる．先ほどの配列から 6 を探索する様子を図５に示す．

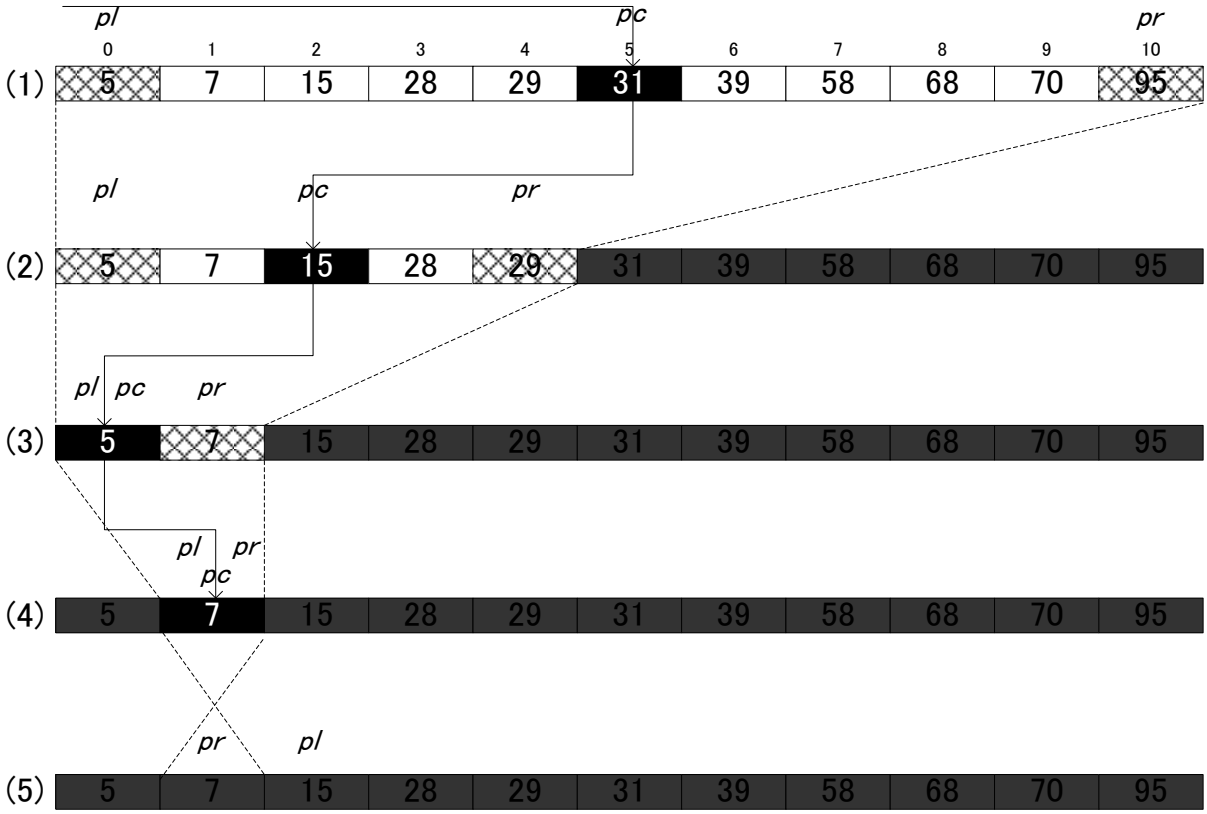

図５　２分探索（探索失敗）

定められた規則に従い，探索対象の範囲を狭められていき，最終的には（5）に示すように pl が pr よりも大きくなり探索失敗となる．

比較を繰り返すたびに探索範囲は半分になるため,必要となる比較回数の平均は log n となる．探索に失敗した場合には log(n + 1)回，探索に成功した場合は，約 log n - 1 回となる．

計算量

同じプログラムでもそれを動作させるためのハードウェアやコンパイラによって実行に要する速度は異なる．そのため，アルゴリズムの性能を客観的に評価するための尺度として計算量というものがある．計算量には以下の２種類がある．

・時間計算量：実行に要する時間を評価する
・領域計算量：必要とされる記憶域やファイル域を評価する

アルゴリズム選択の際はこれらのバランスを考慮する必要がある．以下に線形探索と２分探索の時間計算量について考えてみる．

線形探索の時間計算量

以下は要素数 n の配列 a から key と一致する要素を線形探索するプログラムと各ステップの実行回数と計算量の表を示す.

```
int search(const int a[], int n, int key)
{
        int      i= 0;                  // (1)
        while (i < n) {                 // (2)
                if (a[i] == key)        // (3)
                        return i;       // (4)
                i++;                    // (5)
        }
        return -1;                      // (6)
}
```

表１　線形探索における各ステップの実行回数と計算量

ステップ	実行回数	計算量
(1)	1	O(1)
(2)	n / 2	O(n)
(3)	n / 2	O(n)
(4)	1	O(1)
(5)	n / 2	O(n)
(6)	1	O(1)

変数 i に 0 を代入する(1)，関数から値を返すための(4)，(6)はデータの大きさ n に無関係であり，１回だけ実行される．このような計算量をO(1)と表す．配列の末尾に到達したかを判断する(2)，着目要素と探索すべき値が等しいかどうかを判断する(3)などは平均すると n/2 回となり，n に比例した回数だけ実行される．このような計算量をO(n)と表す．ここで利用しているOは order の略であり，O(n)は，“n のオーダー”あるいは”オーダーの n”などと呼ばれる．

さて，n が変われば，O(n)に要する計算時間は n に比例して長くなるが，O(1)に要する時間は変化することが無い．このことが示すようにO(f(n))とO(g(n))の操作を連続した場合の変化量は次のようになる．

$$O(f(n))+O(g(n)) = O(\max(O(f(n)), O(g(n))))$$

すなわち，より大きい方の計算量に支配される．それでは，線形探索のアルゴリズムの計算量を求めてみよう．以下に示すようにO(n)となる．

$$\begin{aligned} &O(1)+O(n)+O(n)+O(1)+O(n)+O(1) \\ &\quad = O(\max(1, n, n, 1, n, 1)) \\ &\quad = O(n) \end{aligned}$$

２分探索の時間計算量

２分探索では，着目する要素の範囲が半分ずつに減っていくため，各ステップの実行回数は表２のようになる．２分探索で必要となる比較回数の平均は log n となったことを思い出して欲しい．

```
int bin_search(const int a[], int n, int key)
{
        int pl = 0;                          // (1)
        int pr = n - 1;                      // (2)
        do {
                int pc = (pl + pr) / 2; // (3)
                if (a[pc] == key)            // (4)
                        return pc;           // (5)
                else if (a[pc] < key)        // (6)
                        pl = pc + 1;         // (7)
                else
                        pr = pc - 1;         // (8)
        } while (pl <= pr);                  // (9)

        return -1;                           // (10)
}
```

表２　線形探索における各ステップの実行回数と計算量

ステップ	実行回数	計算量
(1)	1	O(1)
(2)	1	O(1)
(3)	log n	O(log n)
(4)	log n	O(log n)
(5)	1	O(1)
(6)	log n	O(log n)
(7)	log n	O(log n)
(8)	log n	O(log n)
(9)	log n	O(log n)
(10)	1	O(1)

したがって，２分探索アルゴリズムの計算量を求めると，次のようにO(log n)となる．

$$O(1)+ O(1)+ O(\log n)+ O(\log n)+ O(1)+ O(\log n)+ \cdots +O(1)$$
$$=O(\log n)$$

計算量に関する大小関係を図６に示しておく．

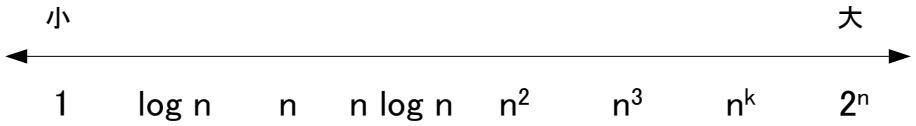

図６　計算量と増加率

課題１の続き

（２）List3-1,3-3,3-4 の探索用関数の動作確認．その後，プログラムを一つに統合する．

（３）線形探索の単純な方法と番兵法，および２分探索について説明する．