アルゴリズム
第10回授業
“基本アルゴリズム探索”
(教科書 Page 57-70)

山口雅樹　（CISSP）

https://github.com/masakage/algorithm

本日の進め方

・前回の復習（最大値最小値を求める）
・基本アルゴリズム（探索）
・基本アルゴリズム（2分探索）
・まとめ

2-2 基本アルゴリズム　線形探索

線形探索（せんけいたんさく、英: linear search, sequential search）は、検索のアルゴリズムの一つである。
リストや配列に入ったデータに対する検索を行うにあたって、
先頭から順に比較を行い、それが見つかれば終了する。

(WIKIPEDIAより)

58ページサンプルプログラム

○プログラム名：線形探索 /* 教科書 58ページサンプル */
○整数型：Ret

●Ret ← LinerSearch1(6)　**関数の呼び出し**
●表示処理(Ret)

/* 以下関数となる */
○整数型：LinerSearch1(整数型：X)
○整数型：T[5]
○整数型：Idx
●T[0] ← 5
●T[1] ← 4
●T[2] ← 2
●T[3] ← 1
●T[4] ← 6

■Idx：0,Idx<5,1
｜▲T[Idx]=X
｜｜●return(Idx)　**発見された場合**
｜▼
□
●return(-1)　**発見されない場合**

ここで6を発見

T[0]	T[1]	T[2]	T[3]	T[4]
5	4	2	1	6

Training 2-2

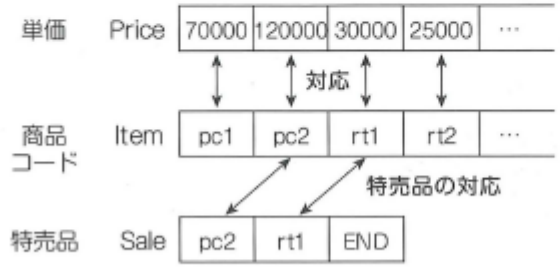
・文字型：Code　は、 pc1 pc2 rt1 rt2　とかが入る
・関数　PriceSearchは、Codeが見つからなければ、 -1 を返す

Training 2−2

商品コードから，当該商品の販売価格を求めるプログラムです。

商品コードと単価は，それぞれN要素の配列Item，Priceに格納されています。配列ItemとPriceの要素は，添字で関連付けられています。たとえば，商品コードがItem[Idx]の商品の単価は，Price[Idx]に格納されています。

特売品の商品コードが，配列Saleに格納されています。配列Saleの末尾要素には，文字列“END”が格納されています。



プログラムは，商品コードを受け取り，販売価格を表示します。受け取った商品コードが特売品のものであれば，単価の10％引きを販売価格とします。特売品でなければ，単価をそのまま表示します。

```
○大域：文字型：Item[N]
○大域：整数型：Price[N]
○大域：文字型：Sale[]

○SellingPrice(文字型：Code)        /* 実行するプログラム */
○整数型：SPrice
・SPrice ← PriceSearch(Code)      /* PriceSearchの呼出し*/
  a
  SaleSearch(Code) = true         /* SaleSearch(Code)の返却値がtrue */
  ・SPrice ← SPrice×0.9           /* 10%割引 */
  ・SPriceを表示
  ―――――――――――――
  ・「コードエラー」を表示
```

```
○整数型：PriceSearch(文字型：Code)    /* Itemを探索するプログラム */
○整数型：Idx
■ Idx：0，  b  ，1
  ▲ Item[Idx]=Code                  /* Codeが配列Itemに存在すれば */
  ・return Price[Idx]               /* 価格を返却 */
■
・return −1                         /* 探索失敗なら−1を返却 */

○論理型：SaleSearch(文字型：Code)     /* Saleを探索するプログラム */
○整数型：Idx
■ Idx：0，  c  ，1
  ▲ Sale[Idx] = Code                /* Codeが配列Saleに存在すれば */
  ・return  d
■
・return false                      /* 探索失敗 */
```

a〜cに関する解答群

ア　SPrice = −1　　イ　SPrice ≠ −1　　ウ　Idx < N

エ　Idx ≧ N　　オ　Price[Idx] ≠ −1　　カ　Sale[Idx]≠ “END”

dに関する解答群

ア　0　　イ　−1　　ウ　true　　エ　false

```
○プログラム名：2分探索 /* 教科書 66ページサンプル */
○整数型：Ret

●Ret ← BinarySearch(22)
●表示処理(Ret)

/* 以下関数となる */
○整数型：BinarySearch(整数型：X)
○整数型：T[6]
○整数型：L
○整数型：H
○整数型：M

●L←0
●H←5
●M←(L＋H)÷2

●T[0] ← 1
●T[1] ← 4
●T[2] ← 8
●T[3] ← 13
●T[4] ← 16
●T[5] ← 22

■L≦H
｜▲T[M]=X
｜｜　●return(M)
｜｜　▲T[M]>X
｜｜　｜●H ← M － 1
｜｜　＋－－－－－－－－－－－
｜｜　｜●L ← M ＋ 1
｜｜　▼
｜▼
｜●M←(L＋H)÷2
□
●return(-1)
```

前半を探しにいく場合

後半を探しにいく場合

22を探す場合

T[0]	T[1]	T[2]	T[3]	T[4]	T[5]
1	4	8	13	16	22

後半を探しにいく

	L	H	M
0回目	0	5	2
1回目	3	5	4
2回目	5	5	5

2回目終了後 T[5]=22となり、
探索が完了する。

○プログラム名：2分探索 /* 教科書 66ページサンプル */
○整数型：Ret

●Ret ← BinarySearch(45)
●表示処理(Ret)

/* 以下関数となる */
○整数型：BinarySearch(整数型：X)
○整数型：T[6]
○整数型：L
○整数型：H
○整数型：M

●L←0
●H←5
●M←(L＋H)÷2

●T[0] ← 1
●T[1] ← 4
●T[2] ← 8
●T[3] ← 13
●T[4] ← 16
●T[5] ← 22

■L≦H
｜▲T[M]=X
｜｜　●return(M)
｜｜　▲T[M]>X
｜｜　｜●H←M－1　**前半を探しにいく場合**
｜｜　＋－－－－－－－－－－－
｜｜　｜●L←M＋1　**後半を探しにいく場合**
｜｜　▼
｜▼
｜●M←(L＋H)÷2
□
●return(-1)

45を探す場合(存在しない)

T[0]	T[1]	T[2]	T[3]	T[4]	T[5]
1	4	8	13	16	22

後半を探しにいく

	L	H	M
0回目	0	5	2
1回目	3	5	4
2回目	5	5	5
3回目	6	5	5

3回目終了後 L≦Hを満たさなくなり
探索が終了する。return (-1)

```
namespace ConsoleApp1
{
    class BinarySearchTest
    {
        static void Main(string[] args)
        {
            Console.WriteLine(BinarySearch(13));
            Console.ReadKey();

            //以下関数
            int BinarySearch(int x)
            {
                var T = new int[] { 1, 4, 8, 13, 16, 22 };
                int Low = 0;
                int High = 5;
                int Mid = 0;

                while (Low <= High)
                {
                    if (T[Mid] == x)
                    {
                        return (Mid);
                    }
                    else if (T[Mid]>x) {
                        High = Mid - 1;
                    }

                    else
                    {
                        Low = Mid + 1;
                    }

                    Mid = (Low + High) / 2;
                }
                return -1;
            }
        }
    }
}
```

C#でのサンプルプログラム
13を探す場合
実行結果は　3

T[0]	T[1]	T[2]	T[3]	T[4]	T[5]
1	4	8	13	16	22

計算量について（教科書 68ページ）

C=log2(N)+1

例えば
配列の大きさが　4なら 2+1で3回の計算
配列の大きさが　16なら 4+1で5回の計算
配列の大きさが　32なら 5+1で6回の計算となる。

配列の大きさが　10000でも、13+1で14回の計算ですむ。
（つまり、線形探索と比べても　計算量が少なくて済む）

Training　2-3

2分探索を理解できてるかどうかがポイントとなる。

構造体
配列に複数要素がセットになっている。

■ Training 2−3

会員番号をもとに会員を探索し，会員の氏名，年齢を表示するプログラムです。

会員情報は，

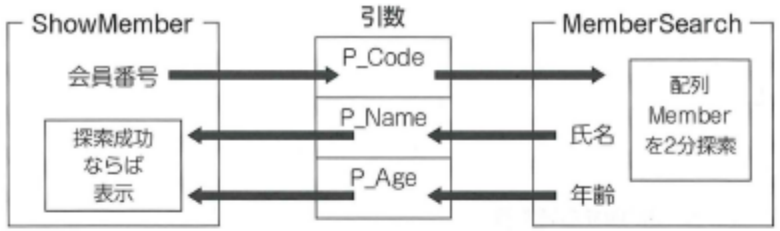
Code：会員番号　Name：氏名　Age：年齢

からなる構造型データで，配列Memberに「会員番号の昇順」で格納されています。会員Member[Idx]の情報は，次の指定で参照できます。

会員番号：Member[Idx].Code
氏名　　：Member[Idx].Name
年齢　　：Member[Idx].Age

プログラムは，会員番号を引数で受け取り，当該会員の氏名と年齢を表示します。該当する会員が存在しないときは「該当者なし」を表示します。

会員の探索は副プログラムMemberSearchを呼び出すことで行います。このMemberSearchは，会員番号を引数で受け取り，配列Memberを2分探索します。探索に成功した場合はtrueを返却し，氏名，年齢を出力用の引数に設定します。失敗した場合にはfalseを返却します。




```
○大域：構造型：Member[N]                    /* 会員配列，要素数はN */

○ShowMember(文字型：P_Code)                 /* P_Code：会員番号 */
○文字型：P_Name
○整数型：P_Age
○論理型：Ret
・Ret ← MemberSearch(P_Code，P_Name，P_Age)  /* 会員番号の2分探索 */
▲Ret ＝ [ a ]
  ・P_Name，P_Ageを表示
――――――――――――――――
  ・「該当者なし」を表示
▼
```

```
○論理型：MemberSearch(文字型：P_Code，文字型：P_Name，整数型：P_Age)
○整数型：L，H，M
・L ← 0                          /* 探索領域の初期化 */
・H ← N−1
・M ← (L+H)÷2
■ [ b ]
▲Member[M].Code ＝ P_Code        /* 探索成功か？ */
  ・P_Name ← Member[M].Name
  ・P_Age ← Member[M].Age
  ・return true
――――――――――――――――
▲Member[M].Code ＞ P_Code        /* 探索領域の再設定 */
  ・[ c ]
――――――――――――――――
  ・L ← M+1
▼
・[ d ]
■
・return false                   /* 探索失敗 */
```

a，c，dに関する解答群

ア　true	イ　false
ウ　H ← M−1	エ　H ← M+1
オ　M ← M−1	カ　M ← M+1
キ　M ← (L+H)÷2	

bに関する解答群

ア　L≦H　　イ　L<H　　ウ　L≧H　　エ　L>H