アルゴリズム論 4,5,6

探索

- 線形探索
- ２分探索
- ハッシュ探索

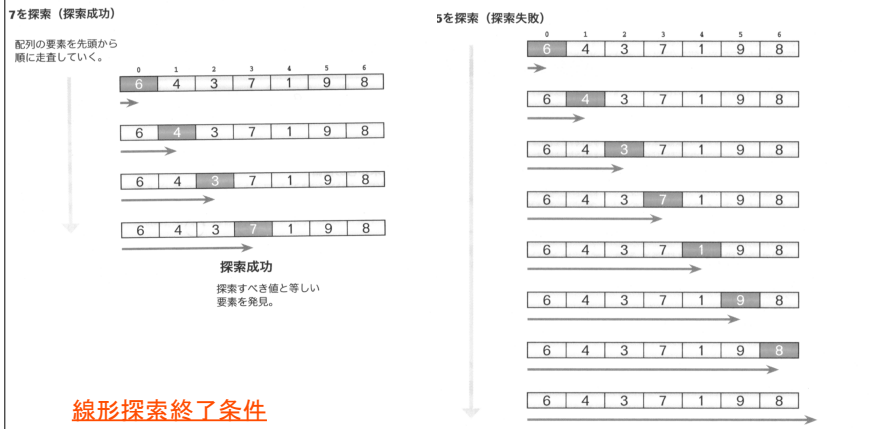
探索（searching）

- 探索(searching)：与えられたデータの中から目的のデータを探し出す処理
- 探索：与えられたn個のデータからa[0]…a[n-1]からx=a[i]となるi(0≦i ≦n-1)を見つける。
- 探索対象のデータ：構造体で定義される複数種別のメンバで登録されている。
- キー：探索を実施する項目
- 探索条件：一致、区間指定、近接指定　等

探索の応用分野

- 応用分野
  - データベースの検索
  - Web検索エンジンの構築
  - ワードプロセッサ検索機能(文字列探索)　等
- データ量が多いほど高速処理アルゴリズムが必要となる
- アルゴリズム選定に考慮する項目は速度のみではない
- 探索アルゴリズム
  - 線形探索（Linear search）
    - 探索の基本、ソート済みでないデータに適用可能
  - 2分探索（Binary search）
    - 高速、ソート済みデータに適用
  - ハッシュ探索（Hash search）
    - 探索するデータの格納方法に工夫した高速な探索法

線形探索（逐次探索） Linear search



線形探索終了条件

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

線形探索プログラム（メイン）

```
#include <stdio.h>
int lin_search(int a[], int n, int key);     /* 関数プロトタイプ */
#define NUM 7

int main(void)
{
        int     i,ky,idx;
        int     x[NUM+1];    /* 番兵法対策 */

        printf(" Input integer number %d times ¥n",NUM); /* データ入力 */
        for (i=0;i<NUM;i++)       {
                printf("x[%d]:",i);
                scanf("%d",&x[i]);
        }

        printf("Number to search:");    /* 探索数値入力 */
        scanf("%d",&ky);

        idx=lin_search(x,NUM,ky);         /* 線形探索 */
        if (idx==-1)
                printf("Searching was failed!¥n");
        else
                printf("%d is located at %d ¥n",ky,idx);

        return(0);
}
```


線形探索プログラム（関数）

```
int lin_search(int a[], int n, int key)
{
        int     i=0;

        while(1) {
                if (i==n)
                        return(-1);
                if (a[i]==key)
                        return (i);
                i++;
                }
}
```

→無限ループ

→条件1成立

→条件2成立

実行結果

```
Input integer number 7 times
x[0]:22
x[1]:5
x[2]:11
x[3]:32
x[4]:120
x[5]:68
x[6]:70
Number to search:120
120 is located at 4
```


番兵法を使用した線形探索関数

```
int lin_search_s(int a[], int n, int key)
{
        int     i=0;

        a[n]=key; /* 番兵 */

        while(1) {
                if (a[i]==key)
                        break;
                i++;
        }
        if (i==n) return(-1);
        else return (i);
}
```

→無限ループ

→条件2成立

→番兵による
条件1成立

データ最後尾にサーチ対象を入れることによって必ず探索が成功する

条件1の比較回数が減少する

演習問題4-1(講義時間内で実施)

- 線形探索を行うプログラムのソースコードを入力し実行する
  - メイン
  - 線形探索(単純)
  - 線形探索(番兵法)
- データを入力(p.7参照)し、実行結果を確認する

線形探索の計算量

- 番兵法を用いない場合と用いる場合で計算量(比較の回数)を比較する
- 各関数中でif文を何回使用したかを調べる
- 線形探索に使用する計算量(最大および平均)を検討する
- 番兵法の計算量における優位性を確認する

線形探索(単純)

```
int count1=0,count2=0; /* count1: 条件1, count2:条件2 */

int lin_search(int a[], int n, int key)
{









}
```


線形探索(番兵法)

```
int count1=0,count2=0; /* count1: 条件1, count2:条件2 */

int lin_search_s(int a[], int n, int key)
{









}
```

線形探索における比較回数

条件２の比較回数

- 最小：一番始めに見つかる
- 最大：一番最後に見つかる場合
- 平均：全ての確率が同じ場合

- データ数nの場合
  最小：１回
  最大：ｎ回
  平均(期待値):1/n x (1+2+・・・+n)=(n+1)/2

$$\because \quad 1+2+\cdots+n=\sum_{i=1}^{n} i=\frac{n(n+1)}{2}$$

- オーダ：O(n)

演習問題4-2(講義時間内で実施)

- 線形探索を行うプログラムのソースコードを以下のように改修する。
  - テキストファイルからデータを入力する(各ファイルにはそれぞれ1000個のデータが入っている)
    - test1.txt,test2.txt,test3.txt,test4.txt
  - 探索にかかった比較の回数を算出する
- データをファイルから入力し、実行結果を確認する

線形探索のまとめ

- 線形探索のアルゴリズム
- 線形探索の終了条件
- 番兵法の優位性
- 計算量:比較の回数
- 最大時間計算量のオーダ　:O(n)
  - データの個数が倍になったら計算量も倍になる