

アルゴリズム論 9

整列処理(ソート)

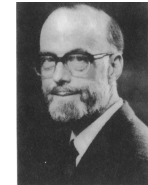
- バブルソート
- 単純選択ソート
- 挿入法
- クイックソート
- ヒープソート

高度な整列処理1(クイックソート)

- ・ より効果的な整列処理のため！！！！

クイックソート

- ・ 考案者 Charles A. R. Hoare 1960年提案
- ・ 最も高速なソートアルゴリズムの一つ
- ・ 分割統治法を応用：平均的には最も速いソート
- ・ 再帰処理を使用することによって効率的で短いソースコードを実現可能



37

クイックソート(原理)

以下のテストの点数を昇順に並べなさい



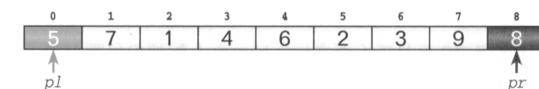
↓
分割統治

38

分割のアルゴリズム

まずは、配列を二つのグループに分割する手順を考えます。

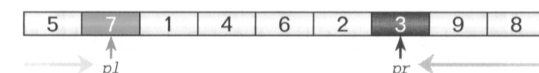
ここでは、下の図に示している配列 a から枢軸として6を選んで分割を行っていきましょう。なお、枢軸を x とし、配列両端の要素の添字である pl を左カーソル、 pr を右カーソルと呼ぶことにします。



枢軸以上の要素は配列の右側に、枢軸以下の要素は配列の左側に移動させなければなりません。そこで、次のことを行います。

- $a[pl] \geq x$ が成立する要素が見つかるまで右方向へ走査する。
- $a[pr] \leq x$ が成立する要素が見つかるまで左方向へ走査する。

そうすると、 pl と pr は下図のように位置します。



39

分割のアルゴリズム(つづき)

ここで、左右のカーソルが指す要素 $a[pl]$ と $a[pr]$ の値を交換します。

5	3	1	4	6	2	7	9	8
---	---	---	---	---	---	---	---	---

再び走査を続けると、左右のカーソルは、下図の位置でストップします。

5	3	1	4	6	2	7	9	8
---	---	---	---	---	---	---	---	---

$\xrightarrow{\quad\quad\quad}$
 \uparrow \uparrow

 pl pr

ここで、これら二つの要素 $a[pl]$ と $a[pr]$ の値を交換します。

5	3	1	4	2	6	7	9	8
---	---	---	---	---	---	---	---	---

再び走査を続けようとしませんが、下図のようにカーソルが交差します。

5	3	1	4	2	6	7	9	8
---	---	---	---	---	---	---	---	---

\uparrow \uparrow

 pr pl

40

分割のアルゴリズム(つづき)

このとき、配列は次のように分割されています。

枢軸以下のグループ	$a[0], \dots, a[pl - 1]$
枢軸以上のグループ	$a[pr + 1], \dots, a[n - 1]$

なお、 $pl > pr + 1$ のときに限りますが、次のようになります。

枢軸と等しいグループ	$a[pr + 1], \dots, a[pl - 1]$
------------	-------------------------------

41

分割プログラム1(メイン)

```
#include <stdio.h>
#define swap(type,x,y) do {type t=x; x=y; y=t;} while(0)
#define NUM 9
void partition(int a[],int n);
int main(void)
{
    int i;
    int x[NUM];

    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]);
    }

    partition(x,NUM); /* データ列分割 */

    printf("Partition is finished \n");
    return(0);
}
```

42

分割プログラム1(関数)

```
void partition(int a[],int n)
{
    int i;
    int pl=0;
    int pr=n-1;
    int x=a[n/2]; /* ピボット */

    do {
        while (a[pl]<x) pl++; /* 左カーソル移動 */
        while (a[pr]>x) pr--; /* 右カーソル移動 */
        if (pl<=pr) {
            swap(int, a[pl],a[pr]); /* 交換 */
            pl++;
            pr--;
        }
    } while (pl<=pr); /* 左カーソル≦右カーソル */
}
```

43

分割プログラム2(関数)

```

/* 分割データ表示 */

printf("Group under pivot %n");
for (i=0;i<=pl-1;i++)
    printf("%d ",a[i]);
printf("%n");

printf("Group over pivot %n");
for (i=pr+1;i<=pr+1;i++)
    printf("%d ",a[i]);
printf("%n");

if (pl>pr+1) {
    printf("Group equivalent pivot %n");
    for (i=pr+1;i<=pl-1;i++)
        printf("%d ",a[i]);
    printf("%n");
}
}

```

44

分割プログラム実行結果

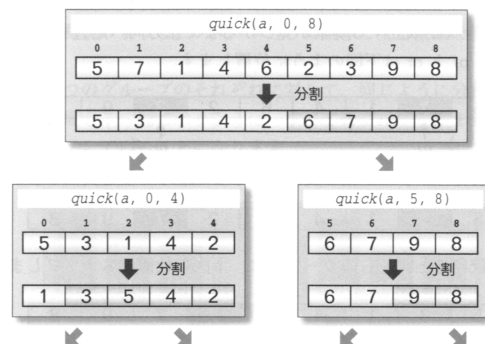
```

Input integer number 9 times
x[0]:5
x[1]:7
x[2]:1
x[3]:4
x[4]:6
x[5]:2
x[6]:3
x[7]:9
x[8]:8
Value at pivot=6
Group under pivot
5 3 1 4 2
Group over pivot
6 7 9 8
Partition is finished

```

45

分割からソートへ



(以下省略)

右カーソル *pr* が先頭要素の添字より大きければ、左グループを再分割。
 左カーソル *pl* が末尾要素の添字より小さければ、右グループを再分割。

46

クイックソートプログラム1(メイン)

```

#include <stdio.h>
#define swap(type,x,y) do {type t=x; x=y; y=t;} while(0)
#define NUM 5
void quick(int a[],int left, int right); /* 関数プロトタイプ */
int count0=0,count1=0; /* count0:比較回数, count1:交換回数 */

int main(void)
{
    int i;
    int x[NUM];

    printf("Input integer number %d times %n",NUM);
    for (i=0;i<NUM;i++) {
        printf("x[%d]:",i);
        scanf("%d",&x[i]);
    }
    quick(x,0,NUM-1);
    printf("Sorting is finished %n");
    for (i=0;i<NUM;i++)
        printf("x[%d] =%d\n",i,x[i]);

    printf("Number of comparison=%d\n",count0);
    printf("Number of swap=%d\n",count1);

    return(0);
}

```

47

クイックソートプログラム2(関数)

```
void quick(int a[],int left, int right)
{
    int    pl=left;
    int    pr=right;
    int    x=a[(pl+pr)/2];      /* ピボット */

    do {
        while (a[pl]<x) { pl++; count0++; } /*左カーソル移動*/
        count0++;
        while (a[pr]>x) { pr--; count0++; } /*右カーソル移動*/
        count0++;
        if (pl<=pr) {
            swap(int, a[pl],a[pr]); /* 交換 */
            pl++;
            pr--;
            count1++;
        }
    } while (pl<=pr); /* 左カーソル<=右カーソル */
    if (left<pr) quick(a,left,pr); /* 再帰呼び出し */
    if (pl<right) quick(a,pl,right); /* 再帰呼び出し */
}
}
```

48

ソートプログラム実行結果

```
Input integer number 5 times
x[0]:60
x[1]:75
x[2]:70
x[3]:56
x[4]:52
Sorting is finished
x[0] =52
x[1] =56
x[2] =60
x[3] =70
x[4] =75
Number of comparison=13
Number of swap=5
```

49

枢軸(Pivot)の選択方法

- 理想的な枢軸:ソート後にメジアンになる値
 - 中央値を求める操作が必要になる:余分な計算
- 中央値に近い値になる可能性の高い値を使用
 - 例1:データ列の中央の値(ソースコードの例)
 - 例2:データ列の先頭、中央、末尾の値の中央値

50

枢軸による効率の違い

- 入力データ:9 8 7 6 5 4 3 2 1

枢軸	中央	先頭	末尾
比較	26	56	56
交換	9	8	8

クイックソートでは枢軸の取り方によって比較の回数が大きく異なる

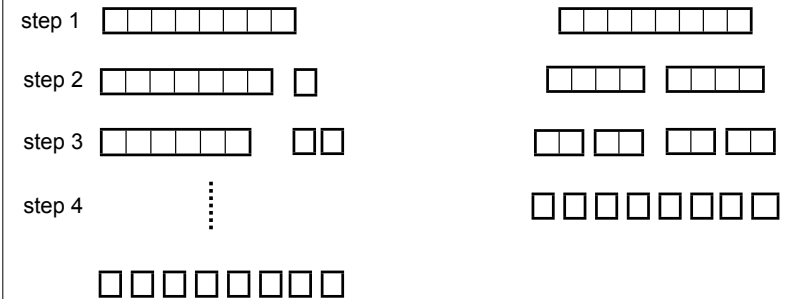
51

演習問題8-2(講義時間内で実施)

- ☑ クイックソート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
 - ☑ クイックソート関数
- ☑ データを入力し、実行結果を確認する
- ☑ 枢軸の値を決定する方法を変え、対応する計算量を検討する

52

クイックソート の計算量1



(a)効率が悪い場合

(b)効率が良い場合

53

クイックソート の計算量2

データn個のソート比較回数

- 最悪の場合
 - 分割した場合一方にn-1個の要素が残る場合
 - 枢軸として最小または最大の値をとる場合
 - 比較の回数

$$(n-1)+(n-2)+\dots+2+1 \text{ 回}$$

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \text{ 回} \rightarrow \text{オーダー } O(n^2)$$

54

クイックソート の計算量3

データn個のソート比較回数

- 効率が良い場合
 - 分割した場合に枢軸の上下に(n-1)/2個の要素が残る場合
 - 枢軸として中央値をとる場合
 - 比較の回数
 - (1)枢軸より大きい小さいかを分ける
 - (2)大きい部分のソート、小さい部分のソート

55

クイックソートの計算量5

データn個のソート比較回数

(1)の計算量:g(n)

$$g(n)=r+(n-r)+1=n+1$$

(2)の計算量:f(n)

nが小さい場合の比較回数

$$f(1)=0$$

$$f(3)=4 \quad \dots$$

•(1)と(2)の合計

$$f(3)=g(3)+f(1)+f(1)=4+0+0=4$$

$$f(7)=g(7)+f(3)+f(3)=8+4+4=16$$

$$f(15)=g(15)+f(7)+f(7)=16+16+16=48$$

$$f(31)=g(31)+f(15)+f(15)=32+48+48=128$$

$$f(63)=g(63)+f(31)+f(31)=64+128+128=320$$

$$f(2^k-1)=g(2^k-1)+2f(2^{k-1}-1) \rightarrow f(2^k-1)/2^k=g(2^{k-1}-1)/2^k+2f(2^{k-1}-1)/2^k$$

$$f(2^{k-1}-1)/2^k=1+2f(2^{k-1}-1)/2^k$$

$$f(2^{k-1}-1)/2^k=1+f(2^{k-1}-1)/2^{k-1}$$

$$\bullet F(k)=f(2^k-1)/2^k \rightarrow F(k)=1+F(k-1)$$

$$F(1)=0 \quad \dots =f(2^1-1)/2^1=f(1)/2=0$$

$$F(2)=1$$

$$F(3)=2$$

$$F(k)=k-1$$

$$\bullet f(2^k-1)=2^k(k-1) \rightarrow \text{近似} \rightarrow f(2^k)=2^k(k-1)$$

$$n=2^k \text{ とする } k=\log_2(n)$$

$$\bullet f(n)=n(\log_2(n)-1)=n \log_2(n)-n$$

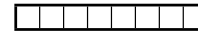


$O(n \log_2 n)$

56

クイックソートの計算量4

簡単に考えると...



データ数がn個の場合、分割のステップ数は



$k=\log_2 n$ 回必要



各ステップにおいて必要な比較の回数はn+1回



従って比較の回数の合計は

$$(n+1) \times k = (n+1) \times \log_2 n = n \log_2 n + \log_2 n$$

57

クイックソートの計算量6

データn個のソート比較回数

• まとめ

– 最悪の場合

$$O(n^2)$$

– 最良および平均的な場合

$$O(n \log_2 n)$$

58