情報工学実験II
ソーティングプログラム

イマム カイリ ルビス

2023年6月11日

目 次

1 概要

1.1 ソートアルゴリズムとは

ソートアルゴリズムとは，データの要素をある順序に並べるアルゴリズムである．最も頻繁に使用される順序は，数値順と辞書順で、昇順または降順のどちらかである．効率的なソートは，入力データがソートされたデータであることを必要とする他のアルゴリズム（検索やマージアルゴリズムなど）の効率を最適化するために重要である．また，人間が読みやすい出力を作成したりする際にもよく使われる．

1.2 計算量

アルゴリズムを実行するのにかかるコンピュタの時間を**時間計算量** (*time complexity*) と呼ばれる．時間計算量は，一般的に *big O notation*(オーダー) という書き方で記す．例えば，$O(n)$, $O(n \log n)$，$O(2^n)$，など．

1.3 効率的なアルゴリズム

効率的なアルゴリズムとは，与えられた課題を最も早く解決するアルゴリズムであると考えることができる．そこで，この実験では，データを数値順に並べるいくつかのソートアルゴリズムを比較し，ある種の数値データに対して、どのアルゴリズムが最も効率的かを調べる．本実験では各プログラムの実行時間は，ソート関数が呼び出される前後のシステム時間差を clock() 関数でカウントされる．

1.4 実行環境

本実験で使用される実行環境：

- プロセッサ：AMD Ryzen 5 5600X
- メモリー：16.0 GB
- OS：Windows 11 Pro
- コンパイラ：gcc

1.5 対象データ

この実験では，すべてのアルゴリズムが C 言語で記述されます．ショーティング対象データは、表 (1) に示すように，異なる特性を持っている．

表 1: ソーティング対象データ

ダーた	特徴
ダーた 1-3	乱数
ダーた 4	昇順
ダーた 5	降順
ダーた 6	バイトニック
ダーた 7	ジグザグ
でーた 8	ランダムマイナス

2　バケットソート

バケットソートは、ソートされていない配列要素をバケットと呼ばれるいくつかのグループに分割するソートアルゴリズムです．各バケットは，適切なソートアルゴリズムを使用するか，同じバケットアルゴリズムを再帰的に適用することによってソートされます．

今回は，入力されたデータの値を配列の添字として，他の配列に格納することにします．そのため，データの個数を数える必要はなく，データの最小値と最大値を知ることが必要．

2.1　プログラム

以下はバケットソートのプログラムである．

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

void setFileName(char *dst, int index) {
    sprintf(dst, "data%d.dat", index);
}

void setOutFileName(char *dst, int index){
    sprintf(dst, "bucket%d.dat", index);
}

void processTime(clock_t t) {
    double time = ((double)t)/CLOCKS_PER_SEC;
    printf("%.3lf ms\n", time*1000);
}

void bucketSort(FILE *in, int *p, int *sort, int offset, int n) {
    int x;

    rewind(in);

    while (fscanf(in, "%d", &x) != EOF) {
        p[x-offset]++;
    }

    int s = 0;

    for (int i = 0; i <= n; i++) {
        if (p[i] > 0)
            for (int j = 0; j < p[i]; j++) {
```

```
                sort[s] = i + offset;
                s++;
            }
    }
}

int  countLines(FILE *in) {
    char c;
    int count = 0;

    do {
        c = fgetc(in);
        if(c == '\n') count++;
    } while (c != EOF);

    rewind(in);

    return count;
}

void startSorting(int n) {
    FILE *in;
    FILE *out;


    for (int i = 1; i <= n; i++) {
        int x;
        char *filename = malloc(10);
        char *outname = malloc(10);

        int max = 0;
        int min = 0;

        printf("data %d,", i);

        setFileName(filename, i);

        if((in = fopen(filename, "r"))== NULL)
            printf("input file name wrong\n");

        while(fscanf(in, "%d", &x) != EOF) {
            max = (max < x) ? x : max;
            min = (min > x) ? x : min;
        }

        rewind(in);
        int lines = countLines(in);

        int *p = calloc((max-min+1), sizeof(int));
        int *s = malloc((lines) * sizeof(int));

        clock_t t;

        t = clock();
        bucketSort(in, p, s, min, (max-min));
        t = clock() - t;
        processTime(t);

        setOutFileName(outname, i);


        out = fopen(outname, "w");

        for (int i = 0; i < lines; i++) {
            fprintf(out, "%d\n", s[i]);
        }

```

```
100         free(p);
101         free(s);
102         free(filename);
103         free(outname);
104     }
105
106     fclose(in);
107     fclose(out);
108 }
109
110 int main() {
111     startSorting(8);
112     return 0;
113 }
```

2.2 動作

このアルゴリズムを動けるためには，$(max - min + 1)$ の大きさの配列を用意する必要がある，この理由は後述する．

バケットソートのプログラムの主な流れは，以下の通りである：

1. バケット配列に $(max - min + 1)$ の大きさを設定する．
2. 出力配列に入力データの大きさを設定する．
3. 各入力データ x はバケット配列の添字 $(x - min)$ 値を $+1$ 増やす．
4. バケット配列から出力配列に元の値変更し、格納する．
5. ソート完了．

2.3 時間計算量

バケットソートの時間的複雑さは $O(n + k)$ であり，ここで n は要素数，k はバケット配列の大きさである．

各データに対してバケットソートの実行時間は表（2）のように表す．ここで表す結果は 10 回の実行結果の平均値である．

表 2: 計算時間 (バケットソート)

データ	data1	data2	data3	data4
計算時間 (ms)	7.2712	7.2169	7.1648	6.3128

データ	data5	data6	data7	data8
計算時間 (ms)	6.1445	5.9061	7.0154	8.6275

表 3: 改善結果

問題	元のアルゴリズム	改善したアルゴリズム
マイナスの対応	×	○
同じ値データ	○	○
バケット配列の大きさ	大きい	小さい

2.4 問題点

入力データをバケット配列の添字に変換するだけでは，マイナスの値に対応できなくなる．マイナスの配列添字はそもそもないからだ．そこで，バケット配列の大きさが $(max - min + 1)$ に設定しいないといけない．入力データがすべて同じ数である場合，$x - x = 0$ となり，大きさ 0 の配列はデータを格納できないため、このような場合に $+1$ が不可欠．

3 挿入ソート

挿入ソートは，トランプを並べ替えるのと同じような仕組みです．配列は事実上，ソートされた部分とソートされていない部分に分けられます．ソートされていない部分から値が選ばれ，ソートされた部分の正しい位置に配置される．

今回は，全部の入力データをソートするでなく，入力データを 1 個ずつ取ってからソートする．

3.1 プログラム

以下は挿入ソートのプログラムである．

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int  countLines(FILE *in) {
    char c;
    int count = 0;

    do {
        c = fgetc(in);
        if(c == '\n') count++;
    } while (c != EOF);

    rewind(in);

    return count;
}

void store_array(FILE *in, int *p) {
    int x;
    int i = 0;

    while (fscanf(in, "%d", &x) != EOF) {
        p[i] = x;
        i++;
    }
}
```

```
28
29  void processTime(clock_t t) {
30      double time = ((double)t)/CLOCKS_PER_SEC;
31      // printf("%.3lf ms\n", time*1000);
32      printf("%.3lf\n", time*1000);
33  }
34
35  void insertionSort(FILE *in, int lines, int *p) {
36      int key;
37      int i = 0;
38
39      while (fscanf(in, "%d", &key) != EOF) {
40          p[i] = key;
41          int j = i - 1;
42          while(j >= 0 && p[j] > key) {
43              p[j+1] = p[j];
44              j--;
45          }
46          p[j+1] = key;
47          i ++;
48      }
49  }
50
51  void setFileName(char *dst, int index) {
52      sprintf(dst, "data%d.dat", index);
53  }
54
55  void setOutFileName(char *dst, int index){
56      sprintf(dst, "insert%d.dat", index);
57  }
58
59  void startSorting(int n) {
60      FILE *in;
61      FILE *out;
62
63      for (int i = 1; i <= n; i++) {
64          char *filename = malloc(10);
65          char *outname = malloc(10);
66
67          // printf("%d, ", i);
68
69          setFileName(filename, i);
70
71          if((in = fopen(filename, "r")) == NULL)
72              printf("input file name error\n");
73
74          int lines = countLines(in);
75          int *p = malloc(lines * sizeof(int));
76
77          clock_t t;
78          t = clock();
79          insertionSort(in, lines, p);
80          t = clock() - t;
81          processTime(t);
82
83          setOutFileName(outname, i);
84
85          out = fopen(outname, "w");
86
87          for (int i = 0; i < lines; i++) {
88              fprintf(out, "%d\n", p[i]);
89          }
90
91          free(p);
92          free(filename);
93          free(outname);
94      }
```

```
95
96      fclose(in);
97      fclose(out);
98  }
99
100 int main(void) {
101     startSorting(8);
102     return 0;
103 }
```

3.2 動作

挿入ソートは比較によりソーティングを行うアルゴリズムである．配列に格納されるデータは，１個前の要素から最初の要素までに比較し、適切の位置に置くこと．しかし，適切の位置に置くという関数がないため，以下の通りである：

1. 入力データ x が n 番目の要素に入力する．
2. $(n-1)$ 番目から 0 番目の要素までに比較する．
3. 左隣の要素が着目している要素より大きければ，着目している要素の位置に代入する．
4. 全部ソートされるまで繰り返す．
5. ソート完了．

3.3 時間計算量

挿入ソートの計算量は場合によって分れる．

- 最悪の計算量：$O(N^2)$
- 平均的な計算量：$O(N^2)$
- 最良の計算量：$O(N)$

各データに対してバケットソートの実行時間は表（4）のように表す．ここで表す結果は 10 回の実行結果の平均値である．

表 4: 計算時間 (バケットソート)

データ	data1	data2	data3	data4
計算時間 (ms)	3568.585	3566.8257	3569.5971	5.7314

データ	data5	data6	data7	data8
計算時間 (ms)	7068.0738	3536.4216	3554.4499	3535.3692

4　バブルソート

バブルソートは隣接している要素を比較し，意図する順番になるまで入れ替える．水の気泡が水面に上がっていくように，配列の各要素は反復するごとに末尾に移動していきます．そのため，バブルソートと呼ばれています．

しかし，今回は要素は末尾に移動するでなく，先頭の方向に移動することにする．そうすると，ソートされた配列は配列の先頭から形成されます．