数据结构课程设计



班级： 1618403

学号： 161840227

姓名： 韦 鑫

指导教师：孙 涵

目录

1.采用的数据结构 ………………………………………… 3

2.算法设计思想 …………………………………………… 3

3.关键代码 ………………………………………………… 4

4.测试数据和结果 ………………………………………… 16

5.算法的时间复杂度及其改进方法 ……………………… 18

6.结束语 …………………………………………………… 19

一、采用的数据结构

用数组存储需要排序的值

二、算法设计思想

（以升序为例）

1. 直接插入排序：从第二个元素开始，并用零号位作为哨兵，将其与前面的值比较，若前面的值大，则后移，直到前面的值小于该值，则在前面值的后一个位置插入该值。
2. 希尔排序：先定义增量数组，以增量数组中的值作为步长，对等步长的不相邻（当步长为1时响铃）的值进行直接插入排序。
3. 冒泡排序：两两比较，若前者大于后者则交换，使得后部分的值为较大值且有序的。
4. 快速排序：先找到枢轴，将小于枢轴的放在其左侧，大于的放在其右侧，返回枢轴，再以枢轴为界，对左右区间分别进行递归操作。
5. 选择排序：保证前部分较小且有序，每次从后部分找到最小值与当前值交换。
6. 堆排序：从下至上进行建堆操作，若当前值小于孩子值，则交换，以此往复得到大顶堆。再从数组(数组模拟树存储)末尾与第一个元素交换，向下调整，可得数组后部分较大且有序。
7. 归并排序：将需排序的数分隔至不能再分隔，既只有一个元素（一个元素是有序的），再进行有序归并。

⑧基数排序：本实验与教材所述的基数排序有所差异。先是找到数组中最大的数以确定数最多有多少为。先对个位进行排序，由辅助数组count[]记录该值所在位置（与矩阵的快速转置中cpot[]类似）。反复此操作至最大位。

三、关键代码

//运行环境:Dev-C++ 5.11

#include<cstdio>

#include<ctime>

#include<cstring>

#include<fstream>

#include<queue>

#include<iostream>

#pragma warning(disable:4996)

using namespace std;

const int maxn = 20000;

const int tot = 10;

int a[maxn + 5];

//用来存储排序名以及所耗的时间，定义排序方式(耗时少的优先级高)

struct node

{

char name[20];

double second;

friend bool operator < (node a, node b)

{

return a.second > b.second;

}

};

//插入排序

void insert\_sort(int a[], int n) //n为数组最后下标

{

for (int i = 2; i <= n; i++)

{

if (a[i] < a[i - 1])

{

a[0] = a[i];

a[i] = a[i - 1];

int j;

for (j = i - 2; a[0] < a[j]; j--)

{

a[j + 1] = a[j];

}

a[j + 1] = a[0];

}

}

}

//希尔排序

//增量数组

int delta[maxn];

//增量数组初始化(可自定义)

int delta\_ini(int delta[], int n)

{

int t = 0;

while (n / 2)

{

delta[t++] = n / 2;

n /= 2;

}

return t;

}

void shell\_insert(int a[], int dk, int n)

{

for (int i = dk + 1; i <= n; i++)

{

if (a[i] < a[i - dk])

{

a[0] = a[i];

int j;

for (j = i - dk; j > 0 && a[0] < a[j]; j -= dk)

{

a[j + dk] = a[j];

}

a[j + dk] = a[0];

}

}

}

void shell\_sort(int a[], int n,int delta[]) //n为数组最后下标

{

int t = delta\_ini(delta, n);

for (int k = 0; k < t; k++)

{

shell\_insert(a, delta[k], n);

}

}

//冒泡排序

void bubble\_sort(int a[], int n)

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n - i - 1; j++)

{

if (a[j] > a[j + 1])

{

int temp = a[j];

a[j] = a[j + 1];

a[j + 1] = temp;

}

}

}

}

//快速排序

int partition(int a[], int left, int right)

{

int temp = a[left];

while (left < right)

{

while (left < right && a[right] > temp)right--;

a[left] = a[right];

while (left < right && a[left] <= temp)left++;

a[right] = a[left];

}

a[left] = temp;

return left;

}

void quick\_sort(int a[], int left, int right)

{

if (left < right)

{

int pos = partition(a, left, right);

quick\_sort(a, left, pos - 1);

quick\_sort(a, pos + 1, right);

}

}

//选择排序

void select\_sort(int a[], int n)

{

for (int i = 0; i < n; i++)

{

int k = i;

for (int j = i + 1; j < n; j++)

{

if (a[j] < a[k])k = j;

}

if (k != i)

{

int temp = a[i];

a[i] = a[k];

a[k] = temp;

}

}

}

//堆排序

//自顶向下调整(大顶堆)

void downAdjust(int heap[], int pa, int child)

{

int i = pa, j = i \* 2;

while (j <= child)

{

if (j + 1 <= child && heap[j + 1] > heap[j])j++;

if (heap[j] > heap[i])

{

swap(heap[j], heap[i]);

//向下

i = j;

j = i \* 2;

}

else break;

}

}

//建堆

void createHeap(int heap[], int n)

{

for (int i = n / 2; i >= 1; i--)

{

downAdjust(heap, i, n);

}

}

//堆排序

void heap\_sort(int heap[], int n)

{

createHeap(heap, n);

for (int i = n; i > 1; i--)

{

swap(heap[i], heap[1]);

downAdjust(heap, 1, i - 1);

}

}

//归并排序

void merge(int a[], int bl, int br, int cl, int cr)

{

int temp[maxn];

int i = bl, j = cl, t = 0;

while (i <= br && j <= cr)

{

if (a[i] <= a[j])temp[t++] = a[i++];

else temp[t++] = a[j++];

}

while (i <= br)temp[t++] = a[i++];

while (j <= cr)temp[t++] = a[j++];

for (int i = 0; i < t; i++)

{

a[bl + i] = temp[i];

}

}

void merge\_sort(int a[], int left, int right)

{

if (left < right)

{

int mid = (left + right) / 2;

merge\_sort(a, left, mid);

merge\_sort(a, mid + 1, right);

merge(a, left, mid, mid + 1, right);

}

}

//基数排序

//选出待排序数组中最大值以确定有多少位

int getMax(int a[], int n)

{

int mx = a[0];

for (int i = 1; i < n; i++)

{

if (a[i] > mx)mx = a[i];

}

return mx;

}

int output[maxn + 5];

void countSort(int a[], const int n, int exp)

{

int i, count[10] = { 0 };

//统计同一位上各个数(0-9)出现的次数

for (i = 0; i < n; i++)count[(a[i] / exp) % 10]++;

//确定位置

for (i = 1; i < 10; i++)count[i] += count[i - 1];

for (i = n - 1; i >= 0; i--)//（不倒着应该也可以吧）

{

output[count[(a[i] / exp) % 10] - 1] = a[i];

count[(a[i] / exp) % 10]--;

}

for (i = 0; i < n; i++)a[i] = output[i];

}

void radix\_sort(int a[],int n)

{

int m = getMax(a, n);

for (int exp = 1; m / exp > 0; exp \*= 10)countSort(a, n, exp);

}

//1、原始数据存在文件中，用相同样本对不同算法进行测试

void task1()

{

char filename[10] = "rand0.txt";

fstream File,outFile;

clock\_t startTime, endTime;

outFile.open("task1.txt", ios::out);

for (int i = 0; i < tot; i++)

{

//使用优先级队列，按照耗时递增的顺序存储排序

priority\_queue<node>q;

node newnode;

filename[4] = i + '0';

outFile << "样例" << i << ":" << endl;

/\*---------------------------------------------------------------------------------------------\*/

File.open(filename, ios::in);

for (int j = 0; j < maxn; j++)File >> a[j];

File.close();

startTime = clock();//计时开始

heap\_sort(a, maxn);

endTime = clock();//计时结束

strcpy(newnode.name, "堆排序 \0");

newnode.second = (double)(endTime - startTime) / CLOCKS\_PER\_SEC;

q.push(newnode);

/\*---------------------------------------------------------------------------------------------\*/

File.open(filename, ios::in);

for (int j = 0; j < maxn; j++)File >> a[j];

File.close();

startTime = clock();

bubble\_sort(a, maxn);

endTime = clock();

strcpy(newnode.name, "冒泡排序\0");

newnode.second = (double)(endTime - startTime) / CLOCKS\_PER\_SEC;

q.push(newnode);

/\*---------------------------------------------------------------------------------------------\*/

File.open(filename, ios::in);

for (int j = 0; j < maxn; j++)File >> a[j];

File.close();

startTime = clock();

select\_sort(a, maxn);

endTime = clock();

strcpy(newnode.name, "选择排序\0");

newnode.second = (double)(endTime - startTime) / CLOCKS\_PER\_SEC;

q.push(newnode);

/\*---------------------------------------------------------------------------------------------\*/

File.open(filename, ios::in);

for (int j = 0; j < maxn; j++)File >> a[j];

File.close();

startTime = clock();

merge\_sort(a, 0, maxn - 1);

endTime = clock();

strcpy(newnode.name, "归并排序\0");

newnode.second = (double)(endTime - startTime) / CLOCKS\_PER\_SEC;

q.push(newnode);

/\*---------------------------------------------------------------------------------------------\*/

File.open(filename, ios::in);

for (int j = 0; j < maxn; j++)File >> a[j];

File.close();

startTime = clock();

quick\_sort(a, 0, maxn - 1);

endTime = clock();

strcpy(newnode.name, "快速排序\0");

newnode.second = (double)(endTime - startTime) / CLOCKS\_PER\_SEC;

q.push(newnode);

/\*---------------------------------------------------------------------------------------------\*/

File.open(filename, ios::in);

for (int j = 1; j <= maxn; j++)File >> a[j];//留0号下标作为“哨兵”

File.close();

startTime = clock();

insert\_sort(a, maxn);

endTime = clock();

strcpy(newnode.name, "插入排序\0");

newnode.second = (double)(endTime - startTime) / CLOCKS\_PER\_SEC;

q.push(newnode);

/\*---------------------------------------------------------------------------------------------\*/

File.open(filename, ios::in);

for (int j = 1; j <= maxn; j++)File >> a[j];

File.close();

startTime = clock();

shell\_sort(a, maxn,delta);

endTime = clock();

strcpy(newnode.name, "希尔排序\0");

newnode.second = (double)(endTime - startTime) / CLOCKS\_PER\_SEC;

q.push(newnode);

/\*---------------------------------------------------------------------------------------------\*/

File.open(filename, ios::in);

for (int j = 0; j < maxn; j++)File >> a[j];

File.close();

startTime = clock();

radix\_sort(a, maxn);

endTime = clock();

strcpy(newnode.name, "基数排序\0");

newnode.second = (double)(endTime - startTime) / CLOCKS\_PER\_SEC;

q.push(newnode);

while (!q.empty())

{

node top = q.top();

q.pop();

outFile << top.name << ": " << top.second << "s" << endl;

}

outFile << "---------------------------------" << endl;

}

outFile.close();

}

//2、屏幕显示每种排序算法对不同样本所花的时间

void task2()

{

char filename[10] = "rand0.txt";

fstream File,outFile;

clock\_t startTime, endTime;

outFile.open("task2.txt", ios::out);

/\*---------------------------------------------------------------------------------------------\*/

outFile << "堆排序:" << endl;

for (int i = 0; i < tot; i++)

{

filename[4] = i + '0';

File.open(filename, ios::in);

if (!File)cout << "fail" << endl;

for (int j = 0; j < maxn; j++)File >> a[j];

File.close();

startTime = clock();

heap\_sort(a, maxn);

endTime = clock();

outFile << "样例" << i << ": " << (double)(endTime - startTime) / CLOCKS\_PER\_SEC << "s" << endl;

}

outFile << "---------------------------------" << endl;

/\*---------------------------------------------------------------------------------------------\*/

outFile << "冒泡排序:" << endl;

for (int i = 0; i < tot; i++)

{

filename[4] = i + '0';

File.open(filename, ios::in);

if (!File)cout << "fail" << endl;

for (int j = 0; j < maxn; j++)File >> a[j];

File.close();

startTime = clock();

bubble\_sort(a, maxn);

endTime = clock();

outFile << "样例" << i << ": " << (double)(endTime - startTime) / CLOCKS\_PER\_SEC << "s" << endl;

}

outFile << "---------------------------------" << endl;

/\*---------------------------------------------------------------------------------------------\*/

outFile << "选择排序:" << endl;

for (int i = 0; i < tot; i++)

{

filename[4] = i + '0';

File.open(filename, ios::in);

if (!File)cout << "fail" << endl;

for (int j = 0; j < maxn; j++)File >> a[j];

File.close();

startTime = clock();

select\_sort(a, maxn);

endTime = clock();

outFile << "样例" << i << ": " << (double)(endTime - startTime) / CLOCKS\_PER\_SEC << "s" << endl;

}

outFile << "---------------------------------" << endl;

/\*---------------------------------------------------------------------------------------------\*/

outFile << "归并排序:" << endl;

for (int i = 0; i < tot; i++)

{

filename[4] = i + '0';

File.open(filename, ios::in);

if (!File)cout << "fail" << endl;

for (int j = 0; j < maxn; j++)File >> a[j];

File.close();

startTime = clock();

merge\_sort(a,0, maxn - 1);

endTime = clock();

outFile << "样例" << i << ": " << (double)(endTime - startTime) / CLOCKS\_PER\_SEC << "s" << endl;

}

outFile << "---------------------------------" << endl;

/\*---------------------------------------------------------------------------------------------\*/

outFile << "快速排序:" << endl;

for (int i = 0; i < tot; i++)

{

filename[4] = i + '0';

File.open(filename, ios::in);

if (!File)cout << "fail" << endl;

for (int j = 0; j < maxn; j++)File >> a[j];

File.close();

startTime = clock();

quick\_sort(a,0, maxn - 1);

endTime = clock();

outFile << "样例" << i << ": " << (double)(endTime - startTime) / CLOCKS\_PER\_SEC << "s" << endl;

}

outFile << "---------------------------------" << endl;

/\*---------------------------------------------------------------------------------------------\*/

outFile << "插入排序:" << endl;

for (int i = 0; i < tot; i++)

{

filename[4] = i + '0';

File.open(filename, ios::in);

if (!File)cout << "fail" << endl;

for (int j = 1; j <= maxn; j++)File >> a[j];

File.close();

startTime = clock();

insert\_sort(a, maxn);

endTime = clock();

outFile << "样例" << i << ": " << (double)(endTime - startTime) / CLOCKS\_PER\_SEC << "s" << endl;

}

outFile << "---------------------------------" << endl;

/\*---------------------------------------------------------------------------------------------\*/

outFile << "希尔排序:" << endl;

for (int i = 0; i < tot; i++)

{

filename[4] = i + '0';

File.open(filename, ios::in);

if (!File)cout << "fail" << endl;

for (int j = 1; j <= maxn; j++)File >> a[j];

File.close();

startTime = clock();

shell\_sort(a, maxn,delta);

endTime = clock();

outFile << "样例" << i << ": " << (double)(endTime - startTime) / CLOCKS\_PER\_SEC << "s" << endl;

}

outFile << "---------------------------------" << endl;

/\*---------------------------------------------------------------------------------------------\*/

outFile << "基数排序:" << endl;

for (int i = 0; i < tot; i++)

{

filename[4] = i + '0';

File.open(filename, ios::in);

if (!File)cout << "fail" << endl;

for (int j = 0; j < maxn; j++)File >> a[j];

File.close();

startTime = clock();

radix\_sort(a, maxn);

endTime = clock();

outFile << "样例" << i << ":" << (double)(endTime - startTime) / CLOCKS\_PER\_SEC << "s" << endl;

}

outFile << "---------------------------------" << endl;

outFile.close();

}

int main()

{

cout << "数据处理中 ... " << endl;

task1();

task2();

cout << "数据处理完毕！" << endl;

system("pause");

return 0;

}

四、测试数据和结果

rand0.txt从小到大有序

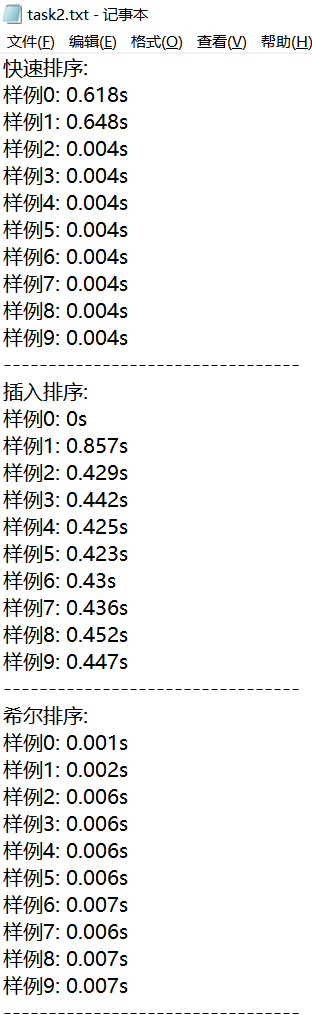
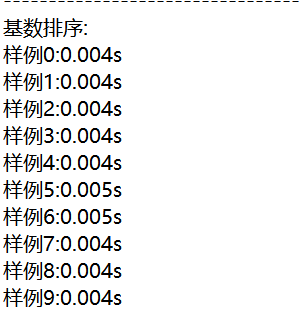
rand1.txt从大到小有序

randi.txt（i = 2,3,…,9）随机数

以上文本均包含两万个数

五、算法的时间复杂度即改进方法

平均时间复杂度

直接插入排序：O(n\*n)

希尔排序：O(n3/2)

冒泡排序：O(n\*n)

快速排序：O(n\*logn)

选择排序：O(n\*n)

堆排序：O(n\*logn)

归并排序：O(n\*logn)

基数排序：O(d(r+n))

六、结束语

代码共512行

本实验通过对八种排序的分析和比较，让我对各个排序的时间以及空间复杂度有了更直观的认识，例如在有序的待排数值有序的情况下快速排序会退化成冒泡排序，而直接选择、冒泡排序则可以降至O(n)的时间复杂度。

另外，使用Visual Studio 2017运行快速排序时，出现了栈溢出的现象，根据所查资料可以判断此情况是因为递归的层数过多引起的栈溢出，所以我换成使用Dev-c++运行该程序，且正常运行。