哈尔滨工业大学

人工智能专业

实验报告

课程名称：计算机算法基础

实验项目：排序方法的实验比较

实验题目：归并排序与快速排序算法的实现与实验比较

实验日期：2020.10.14

班级：1180400510

学号：1180400510

姓名：石瑞河

同组人：张皓为

|  |  |  |
| --- | --- | --- |
| 设计成绩 | 报告成绩 | 指导老师 |
|  |  | 张岩 |

**一、实验目的**

排序是数据处理的最基本也是最重要的操作，其目的是将一组“无序”的记录序列调整为“有序”的记录序列。计算机头两个字就是计算，诞生的目的最初就是为了帮助人类简化计算工作，如繁重的公式计算、各种数据计算和存储等等。基础操作的效率的提升势必也意味着最终产品功能的大跃进。本实验通过讨论不同算法处理不同数据时速度的差异来讨论算法的特性。

**二、实验要求及实验环境**

**实验要求**

1.实现归并排序的递归和非递归算法

2.实现快速排序算法。划分方法要求：首元素划分法、三数取中划分法、随机划分法、“不同取大”划分法。

**实验环境**

Windows 10 Intel(R) Core i7-8550U CPU @ 1.80GHz

**三、设计的基本思想和算法的原理**

归并排序: 归并排序是建立在归并操作上的一种有效的排序算法。该算法是分治法的一个典型应用，所谓归并就是将若干个有序结点序列的归并来实现排序，即不断将列表拆分为一半，如果列表为空或有一个项，则按定义进行排序。如果列表有多个项，我们分割列表，并递归调用两个半部分的合并排序。一旦对两半排序完成，获取两个较小的排序列表并将它们组合成单个排序的新列表的过程。

归并排序一般都是用递归实现，但往往会一直调用栈，会消耗大量内存，所以我们采取循环的手段来以非递归的方式同样进行归并排序。

快速排序：快速排序于1962年由C.A.R.Hoare提出，基本思想是通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

快速排序有四种类型，分别是首元素划分法、三数取中划分法、随机划分法和“不同取大”划分法。

三数取中法：在快排的过程中每一次要取一个元素作为主元，以这个数字来将序列划分为两部分。三数取中就是取左端、中间和右端三个数，然后进行排序，将中间数作为主元，而后根据主元进行分割，双向扫描，从左边找大于主元的数，从右边找小于主元的数，然后交换。再分别对左序列和右序列做三数取中。

首元素划分法：取首元素为主元，从起始和末尾同时进行扫描，最后两端指针位置重合。

随机划分法：随机一个元素作为主元，再进行划分。

“不同取大”划分法：选取连续两个元素，取其中最大的为主元，然后进行划分。

**四、实验结果与分析**

**实验结果数据汇总表**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验  环境 | CPU | Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99GHz | | | | | **此表中的数据规模仅供参考，可根据实际情况进行合理调整。** | | |
| 内存 | 8GB | | | | |
| 操作系统 | Windows 10 | | | | |
| 编译软件 | g++ | | | | |
| 数据规模 |  | 10^4 | 2\*10^4 | 10^5 | 2\*10^5 | 10^6 | 2\*10^6 | 10^7 | 2\*10^8 |
| 归并排序(递归) | 随机序列 | 0.002s | 0.003s | 0.019s | 0.037s | 0.215s | 0.416s | 2.218s | 122.107s |
| 正序序列 | 0.001s | 0.003s | 0.014s | 0.031s | 0.275s | 0.614s | 4.071s | 74,821s |
| 反序序列 | 0.005s | 0.011s | 0.025s | 0.054s | 0.27s | 0.417s | 3.652s | 31.609s  (1\*10^8) |
| 归并排序(非递归) | 随机序列 | 0.002s | 0.003s | 0.017s | 0.038s | 0.205s | 0.417s | 2.245s | 122.042s |
| 正序序列 | 0.001s | 0.003s | 0.015s | 0.032s | 0.269s | 0.632s | 4.264s | 91.077s |
| 反序序列 | 0.002s | 0.003s | 0.017s | 0.032s | 0.194s | 0.402s | 2.119s | 22.922  (1\*10^8) |
| 快速排序(随机序列) | 首元素划分法 | 0.001s | 0.003s | 0.013s | 0.026s | 0.133s | 0.238s | 1.319s | 39.585s |
| 三数取中划分法 | 0.001s | 0.005s | 0.03s | 0.06s | 0.381s | 0.531s | 3.311s | 65.172s |
| 随机划分法 | 0.002s | 0.006s | 0.033s | 0.07s | 0.25s | 0.577s | 2.91s | 68.849s |
| “不同取大”划分法 | 0.002s | 0.005s | 0.049s | 0.056s | 0.228s | 0.532s | 2.495s | 47.43s |
| 快速排序(正序序列) | 首元素划分法 | 0.139s | 0.653s | boom | boom | boom | boom | boom | boom |
| 三数取中划分法 | 0s | 0.001s | 0.007s | 0.014s | 0.103s | 0.182s | 1.204s | boom |
| 随机划分法 | 0.002s | 0.003s | 0.012s | 0.039s | 0.289s | 0.838s | 14.655s | boom |
| “不同取大”划分法 | 0.067s | 0.279s | boom | boom | boom | boom | boom | boom |
| 快速排序(反序序列) | 首元素划分法 | 0.05s | 0.095s | 2.007s | 2.825s | 23.852s | 48.55s | 234.099s | boom |
| 三数取中划分法 | 0.001s | 0.005s | 0.009s | 0.01s | 0.111s | 0.183s | 0.852s | 5.756s  (1\*10^8) |
| 随机划分法 | 0s | 0.002s | 0.006s | 0.013s | 0.07s | 0.172s | 1.464s | 54.729s  (1\*10^8) |
| “不同取大”划分法 | 0.082s | 0.28s | 2.944s | 5.991s | 29.402s | 61.212s | 324.664s | boom |

归并排序的递归和非递归实现处理速度较为接近，没有显著差别。对于正序、反序的序列，归并排序的耗时约为同规模随机数据的50%。

快速排序，不同的划分方法对快排的效率影响较大，对能处理的数据规模也有影响。几种划分方法中，三数取中法表现最好，在随机序列中，几种划分方法的结果没有显著差异；在正序和反序序列上，首元素划分法和不同取大划分法很快就由于递归层数过大而栈溢出，因此不能处理较大规模数据，而随机划分法在正序和反序序列上的效率大幅下降，耗时为三数取中的数倍至10倍，三数划分法耗时仅为随机序列上的约30-50%。

比较快排和归并两种排序方式，可以发现，采用三数取中法的快排比归并算法在随机和正反序列上都显著的优秀，可以认为归并排序的时间复杂度常数较大。

**五、经验体会与不足**

在处理较大规模数据时，快速排序如果划分方式不合理，在某些情况下会遇到栈溢出的问题；归并排序由于需要额外的辅助数组，会遇到内存不足的问题。

**六、附录：源代码（带注释）**

MergeSort

void Merge(int array[], int L, int Mid, int R)  // 有序数组合并操作 [L,Mid], [Mid+1,R]

{

    static const int MAXN = 1e7;

    static int Temp[MAXN];

    int i = L, j = Mid+1, k = L;

    while(i <= Mid && j <= R)

    {   //比较，取小

        if(array[i] <= array[j])Temp[k++] = array[i++];

        else Temp[k++] = array[j++];

    }

    while(i <= Mid)Temp[k++] = array[i++];

    while(j <= R) Temp[k++] = array[j++];   //把剩余的放入辅助数组

    for(i = L; i <= R; i++)array[i] = Temp[i];  //把合并后的段拷贝回原数组

}

void MergeSort(int array[], int L, int R)  //递归实现的归并排序

{

    if(L >= R)return;

    int Mid = (L+R) >> 1;

    MergeSort(array, L, Mid);

    MergeSort(array, Mid+1, R);

    Merge(array, L, Mid, R);

}

void MergeSort\_NoRec(int array[], int L, int R) //非递归实现的归并排序

{

    int len = R-L+1;

    int i, st, Mid, ed;

    for(i = 0; (1<<i) <= len; i++)    //从小段开始归并，直到将整个数组归并

    {

        for(st = L; st <= R; st += 1<<(i+1) )

        {

            Mid = st-1 + (1<<i);

            ed = st-1 + (1<<(i+1));

            if(Mid > R) Mid = R;

            if(ed > R) ed = R;

            Merge(array, st, Mid, ed);

        }

    }

}

QuickSort

#define MAX(a, b)       (((a) > (b) ) ? (a) : (b))

#define MIN(a, b)       (((a) < (b) ) ? (a) : (b))

#define MID(a,b,c)      (MAX(a,b)>c? MAX(MIN(a,b),c) : MIN(MAX(a,b),c))

void Quick\_sort(int array[], int L, int R, int Method)

{

    int Mid=array[(L+R)/2],temp;

    int a,b,c;

    int i,j;

    switch (Method)  // 1:首元素  2：三数取中  3：随机划分  4：不同取大

    {

    case 1:

        Mid=array[L];

        break;

    case 2:

        a = array[L];

        b = array[R];

        c = array[(L+R)>>1];

        Mid = MID(a,b,c);

        break;

    case 3:

        a = L + rand()%(R-L+1);

        Mid = array[a];

        break;

    case 4:

        Mid = array[L];

        for(i = L+1; i <= R; i++)

            if(array[i] != Mid) {Mid = MAX(array[i],Mid); break;}

        if(i > R) return;

        break;

    default:

        Mid = array[(L+R)>>1];

        break;

    }

    int x=L,y=R;

    while(x<=y)

    {

        while(array[x]<Mid) x++;

        while(array[y]>Mid) y--;        // 与划分值比较

        if(x<=y)

        {

            temp=array[x]; array[x]=array[y]; array[y]=temp; //交换变量值

            x++; y--;

        }

    }

    if(L<y) Quick\_sort(array, L, y, Method);

    if(x<R) Quick\_sort(array, x, R, Method);

}

测试代码，以归并非递归算法为例，随机产生10000组数据测试代码的正确性。

int a[50000],b[50000],c[50000];

int test()

{

    int cnt = 0, n, i, j;

    while(cnt++ < 10000)

    {

        n=rand()%10000;

        for(i=1;i<=n;i++)a[i] = b[i] = c[i] = rand();

        MergeSort\_NoRec(a,1,n);

        sort(b+1,b+1+n);

        cout<<n<<endl;

        //getchar();

        for(i=1;i<=n;i++)

        {

            //printf("%d\n",a[i]-b[i]);

            if(a[i] != b[i])

            {

                printf("Something Wrong!\n");

                freopen("wrong7.txt","w",stdout);

                printf("%d  %d\n", n);

                for(j = 1; j <= n; j++)printf("%d ",c[j]);

                putchar('\n');

                for(j = 1; j <= n; j++)printf("%d ",a[j]);

                putchar('\n');

                for(j = 1; j <= n; j++)printf("%d ",b[j]);

                return 0;

            }

        }

        printf("OK!\n");

    }

    return 1;

}

数据生成 main.cpp

#include <bits/stdc++.h>

#include "Merge.hpp"

#include "Quick.hpp"

using **namespace** std;

**const** **int** N1 = 1e4;

**const** **int** N2 = 1e5;

**const** **int** N3 = 1e6;

**const** **int** N4 = 1e7;

**const** **int** N5 = 1e8;

**const** **int** N = N4\*2;

**int** a[N+100];

**int** b[N+100];

**int** c[N+100];

**int** d[N+100];

**int** e[N+100];

**int** f[N+100];

**void** output(**int** n, **int** flag = 1)

{

**int** i, t1, t2, t3, interval;

*//归并、递归*

    printf("归并，递归:  \t\t\t");

    t1 = clock();

    MergeSort(a,1,n);

    t2 = clock();

    interval = t2 - t1;

    cout << interval << endl;

*//归并、非递归*

    printf("归并，非递归:\t\t\t");

    t1 = clock();

    MergeSort\_NoRec(b,1,n);

    t2 = clock();

    interval = t2 - t1;

    cout << interval << endl;

*//快排,1*

    printf("快排，首元素:\t\t\t");

    t1 = clock();

    if(n\*flag < N2 && n <= N4) Quick\_sort(c,1,n,1);

    t2 = clock();

    interval = t2 - t1;

    cout << interval << endl;

*//快排,2*

    printf("快排，三数取中:\t\t\t");

    t1 = clock();

    Quick\_sort(d,1,n,2);

    t2 = clock();

    interval = t2 - t1;

    cout << interval << endl;

*//快排,3*

    printf("快排，随机划分:\t\t\t");

    t1 = clock();

    Quick\_sort(e,1,n,3);

    t2 = clock();

    interval = t2 - t1;

    cout << interval << endl;

*//快排,4*

    printf("快排，不同取大:\t\t\t");

    t1 = clock();

    if(n\*flag < N2 && n <= N4) Quick\_sort(f,1,n,4);

    t2 = clock();

    interval = t2 - t1;

    cout << interval << endl;

}

**void** test(**int** n)

{

    printf("\n数据量 = %d \n", n);

**int** i, t1, t2, t3, interval;

    cout<<"随机序列"<<endl;

    for(i = 1;i <= n; i++) a[i] = b[i] = c[i] = d[i] = e[i] = f[i] = rand()\*rand();

    output(n,0);

    putchar('\n');

    cout<<"正序序列"<<endl;

    output(n);

    putchar('\n');

    cout<<"反序序列"<<endl;

    for(i = 1;i <= n; i++) a[i] = b[i] = c[i] = d[i] = e[i] = f[i] = n+1-i;

    output(n);

    putchar('\n');

    cout<<endl;

}

**int** main()

{

*//srand(time(NULL));*

    cout<<"running"<<endl;

*//freopen("output1.txt","w",stdout);*

**int** i,n;

**int** t1, t2, interval;

    n = 10;

**int** p[15];

    for(i = 1;i <= n; i++) p[i] = rand();

    MergeSort(p,1,n);

    test(N1);

    test(N1\*2);

    test(N2);

    test(N2\*2);

    test(N3);

    test(N3\*2);

    test(N4);

    test(N4\*2);

    printf("\n\ntotal time = %d\n", clock());

}