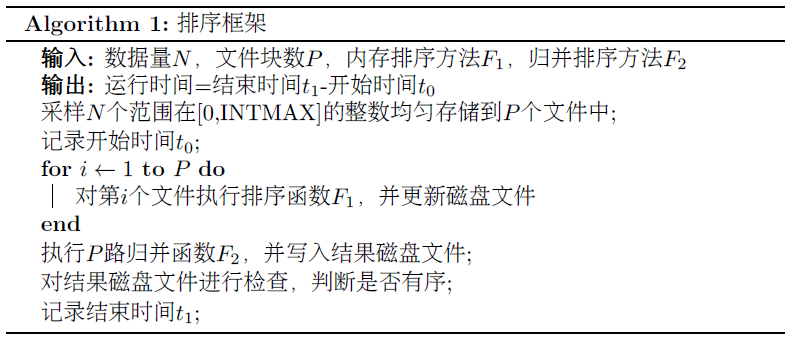
# 海量整数排序

## 应承轩 201785071 电计1701

## 算法流程图

我的算法排序框架定义如下



## 实验结果

在我的实验中，我的数据为[0~INTMAX]的均匀分布，默认排序函数为STDSORT，它是一种混合排序方法，在N较大的时候是快速排序，当N较小的时候使用插入排序，STDSORT是一个经过了实际检验，被广泛应用，并且被纳入CPP标准的排序函数。

1. **运行时间和数据量的关系：**

|  |  |  |  |
| --- | --- | --- | --- |
| 数据量N | 分块排序（MS） | 归并排序（MS） | 总时间（MS） |
| 1000000000 | 113,044 | 82,218 | 195,262 |
| 100000000 | 9,050 | 8,784 | 17,834 |
| 10000000 | 1,063 | 879 | 1,942 |
| 1000000 | 109 | 111 | 220 |
| 100000 | 52 | 10 | 62 |
| 10000 | 24 | 1 | 25 |

图表 1 数据量-运行时间表

在这个实验中，我设置分块数量=10，并改变数据量，获得数据量-运行时间曲线（ms）如下：

图表 2 数据量-运行时间折线图

可以看出，随着数据量的增加，运行时间也增加，成近似线性的关系。

1. **运行时间和切块个数的关系：**

|  |  |  |  |
| --- | --- | --- | --- |
| 切片个数 | 分块排序（MS） | 归并排序（MS） | 总时间（MS） |
| 10 | 113,044 | 82,218 | 195,262 |
| 50 | 97,930 | 97,541 | 195,471 |
| 100 | 97,924 | 97,278 | 195,202 |
| 500 | 88,797 | 102,368 | 191,165 |

图表 3 运行时间-切块个数表

在这个实验中，我设置数据量=10亿，并改变切块个数，获得切块个数-运行时间曲线（ms）如下：

图表 4 运行时间-切块个数折线图

可以看出，随着数据量的增加，总运行时间几乎不变。但是，分块排序时间随着切片个数的增加，先减少后增加，成“凹“曲线，一个直观的解释是：随着切片个数的增加，数据的局部性也增加，提高了CPU的性能，使得算法运行时间缩短，当切片个数过多的时候，磁盘IO反而成为算法的瓶颈，进而导致了时间的增加。

1. **运行时间和分块排序算法F1的关系：**

在这个实验中，我设置数据量=1亿，切块个数=100，并尝试不同的排序方法，实验结果如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 排序方法 | 分块排序（MS） | 归并排序（MS） | 总时间（MS） |
| 堆排序 | 20,548 | 9,586 | 30,134 |
| 快速排序 | 15,153 | 9,527 | 24,680 |
| 归并排序 | 12,927 | 9,487 | 22,414 |
| 标准排序 | 9,642 | 9,813 | 19,455 |
| 稳定标准排序 | 9,022 | 9,434 | 18,456 |

图表 5 运行时间-分块排序算法F1表

从表中可以看出，稳定标准排序速度较快，其次是标准排序，然后是归并排序和快速排序，最慢的是堆排序，堆排序最慢的一个直观的原因就是：堆排序需要比较根节点和子节点的大小，在内存中，往往这两者存储的距离较远，不满足缓存的局部性，因此比较的延迟比较大。

图表 6 运行时间-分块排序算法F1折线图

从柱状图中，我们可以看出，所有方法的归并排序时间基本相同，主要差异在分块排序算法F1的时间。

## 算法代码

#include <iostream>

#include <algorithm>

#include <random>

#include <fstream>

#include <queue>

using namespace std;

#include <stdio.h>

#include <time.h>

#define TOT (int)1e9

#define PARTS 10

struct node {

int f\_id;

int num;

bool operator < (const struct node &a) const {

return num > a.num;

}

} ;

void sort\_a\_file(int id) {

time\_t start = clock();

int\* mem = new int[TOT / PARTS];

char filename[100];

sprintf(filename, "data\_part%d", id);

FILE\* f = fopen(filename, "rb");

fread(mem, sizeof(int), TOT / PARTS, f);

fclose(f);

//cout << mem[0] << endl;

sort(mem, mem + TOT / PARTS);

f = fopen(filename, "wb+");

fwrite(mem, sizeof(int), TOT / PARTS, f);

fclose(f);

delete[]mem;

cout << filename << "排序时间：" << clock() - start << "ms" << endl;

}

int main(void)

{

time\_t start, end;

// 初始化随机数据 并写入文件

srand(1);

// 把数据写入文件

int\* mem = new int[TOT/PARTS];

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

{

for (size\_t i = 0; i < TOT / PARTS; i++)

{

mem[i] = rand() \* rand();

}

char filename[100];

sprintf(filename, "data\_part%d", i);

cout << "正在写入" << filename << endl;

FILE\* f = fopen(filename, "wb+");

fwrite(mem, sizeof(int), TOT / PARTS, f);

fclose(f);

}

delete[]mem;

cout << "完成初始化随机数据" << endl << endl << endl;

// 对每个文件排序，并记录总时间

start = clock();

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

{

sort\_a\_file(i);

}

cout << "局部排序时间：" << clock() - start << "ms" << endl;

// 进行归并排序、

FILE\* ff[PARTS], \* ans;

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

{

char filename[100];

sprintf(filename, "data\_part%d", i);

ff[i] = fopen(filename, "rb");

}

ans = fopen("ans", "wb+");

priority\_queue<node> q;

int head[PARTS];

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

{

int t;

fread(&t, sizeof(int), 1, ff[i]);

q.push({i,t});

}

start = clock();

int c = 0, t;

struct node hd;

while (!q.empty())

{

hd = q.top();

q.pop();

fwrite(&hd.num, sizeof(int), 1, ans);

//cout << ++c << ":" << hd.num << "-" << hd.f\_id << endl;

if (!feof(ff[hd.f\_id]) && fread(&t, sizeof(int), 1, ff[hd.f\_id]) == 1) {

q.push({ hd.f\_id,t });

}

}

cout << "全局归并排序时间：" << clock() - start << "ms" << endl;

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

{

fclose(ff[i]);

}

fclose(ans);

ans = fopen("ans", "rb");

int last = -1;

int cur;

int cnt = 0;

while (!feof(ans) && fread(&cur, sizeof(int), 1, ans) == 1)

{

if (cur < last)

{

//cout << cnt << ":" << last << "-" << cur << endl;

cout << "出错了" << endl;

return 1;

}

last = cur;

cnt++;

}

cout << "有序" << endl;

return 0;

}

当参数为：

#define TOT (int)1e9

#define PARTS 10

的运行结果：

正在写入data\_part1

正在写入data\_part2

正在写入data\_part3

正在写入data\_part4

正在写入data\_part5

正在写入data\_part6

正在写入data\_part7

正在写入data\_part8

正在写入data\_part9

完成初始化随机数据

data\_part0排序时间：10407ms

data\_part1排序时间：10387ms

data\_part2排序时间：10273ms

data\_part3排序时间：11426ms

data\_part4排序时间：10293ms

data\_part5排序时间：10564ms

data\_part6排序时间：10257ms

data\_part7排序时间：10747ms

data\_part8排序时间：10261ms

data\_part9排序时间：10396ms

局部排序时间：113044ms

全局归并排序时间：82218ms

有序