外部排序

09022313 李晶晶

问题描述/需求分析

外部磁盘有一个大于内存且含有无序数字的文件，需要对该文件内的数字进行排序。

系统结构/算法设计

采用归并排序的方法。具体操作分为2步：

1.生成已经排好序的初始顺串

2.归并顺串，采用2路归并的策略

生成已经排好序的文件

功能模块设计

const int BUFFER\_SIZE = 5;

const int RUN\_SIZE = 10;

1.生成N个随机数字，放入input.txt文件中

void generateInputFile(const string& filename, int N)

2.计算文件内含有多少数字

生成初始顺串并放入临时文件

int countData(const string& filename)

void sortAndWriteChunk(ifstream& input, int chunkIndex)

3.运用buffer归并两个顺串

只有一个顺串时直接复制

进行2路归并

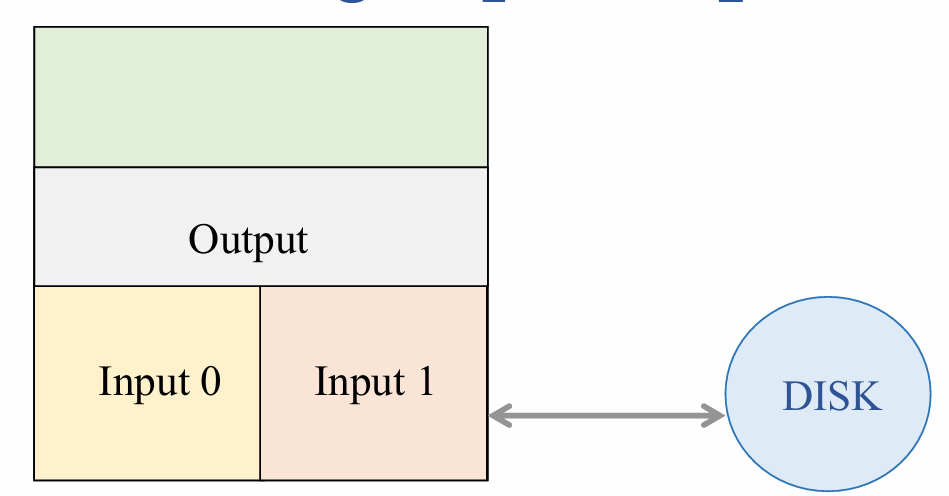
void mergeChunks(const string& file1, const string& file2, const string& outputFile)

void mergeChunks(const string& file1, const std::string& outputFile)

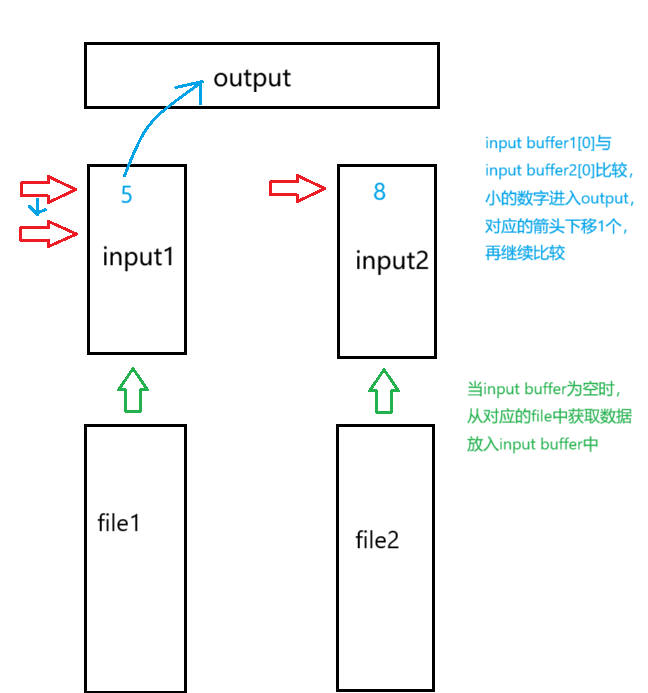
int twoWayMerge(int totalChunks)

这里重点介绍运用buffer归并两个顺串和进行2路归并的逻辑

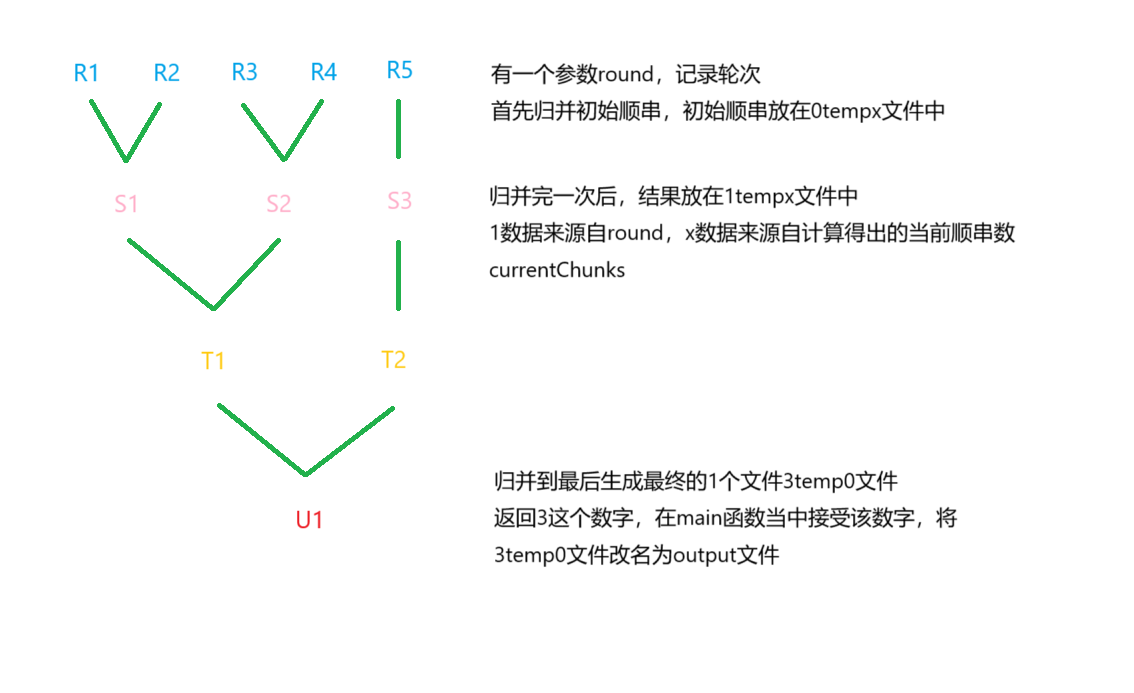
运用buffer归并两个顺串



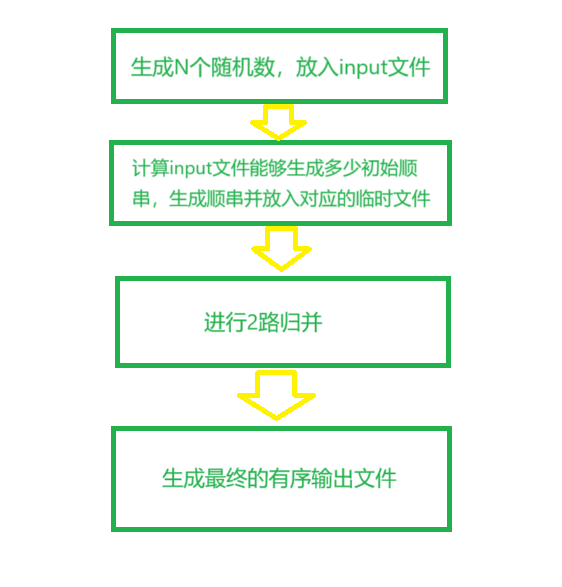
有2个input buffer分别对应顺串1顺串2，buffer长度设定为5



2路归并



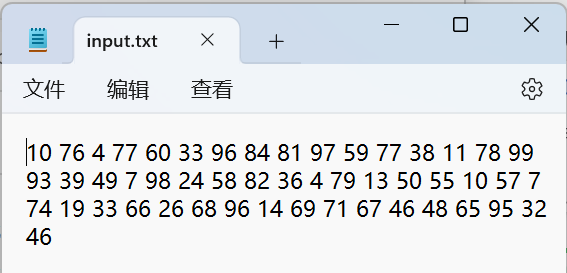
模块流程



结果测试与分析

为方便测试说明，这里选取N为50。

Input文件



过程性临时文件

文本

中度可信度描述已自动生成

结果输出文件

文本

描述已自动生成

测试结果与预期相符

接下来进行IO的性能分析

测试不同的N和RUN\_SIZE对IO性能的影响

图片包含 表格

描述已自动生成

横坐标为N，纵坐标为IO次数，设置RUN\_SIZE为10。

从图表中可以看出，N越大，IO次数越大，并且两者基本成正向关系。

图片包含 图形用户界面

描述已自动生成

横坐标为RUN\_SIZE，纵坐标为IO次数，设置N为8000。

从图表中可以看出，RUN\_SIZE越大，IO次数越小，两者成负相关。

实验总结

在本次实验过程中，遇到的问题就是进行2路归并时，归并轮次和一次归并完的结果应该如何表示？我采用临时文件的方式解决这个问题。临时文件存储两个顺串归并完的结果，同时临时文件的命名上体现了归并的轮次和顺序。如0temp0，代表着第0轮归并的第0号文件；3temp4，代表着第3轮归并的第4号文件，诸如此类。

还有问题是具体实施两个顺串归并，因为采用buffer，所以实现上会稍稍复杂一点。

本次外部排序的实验整体完成度较高，为后续的改进打下基础。

附录

#include <iostream>

#include <fstream>

#include <vector>

#include <queue>

#include <string>

#include <algorithm>

#include <filesystem>

#include <cstdlib>

#include <ctime>

using namespace std;

const int BUFFER\_SIZE = 5;

const int RUN\_SIZE = 350;

int ioReadCount = 0; // To count read operations

int ioWriteCount = 0; // To count write operations

// Function to generate an input file with N random numbers

void generateInputFile(const string& filename, int N)

{

ofstream output(filename);

if (!output.is\_open())

{

cerr << "Error creating input file." << endl;

return;

}

// Seed the random number generator

srand(static\_cast<unsigned int>(time(0)));

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

{

output << rand() % 100 << " ";

}

output.close();

}

// Function to count the number of data entries in the file

int countData(const string& filename)

{

ifstream input(filename);

int count = 0;

int num;

while (input >> num)

{

count++;

}

return count;

}

// Function to sort and write a chunk to a temporary file

void sortAndWriteChunk(ifstream& input, int chunkIndex)

{

vector<int> buffer;

int num;

// Read a chunk of data

while (buffer.size() < RUN\_SIZE && input >> num)

{

buffer.push\_back(num);

}

ioReadCount++;

// If no data was read, return

if (buffer.empty()) return;

// Sort the chunk

sort(buffer.begin(), buffer.end());

// Write sorted data to a temporary file

ofstream tempFile(to\_string(0) + "temp" + to\_string(chunkIndex) + ".txt");

for (int val : buffer)

{

tempFile << val << " ";

}

ioWriteCount++;

tempFile.close();

}

void mergeChunks(const string& file1, const std::string& outputFile)

{

filesystem::copy(file1, outputFile);

ioReadCount++;

ioWriteCount++;

return;

}

// Function to merge two chunks into an output file

void mergeChunks(const string& file1, const string& file2, const string& outputFile)

{

ifstream input1(file1);

ifstream input2(file2);

ofstream output(outputFile);

vector<int> buffer1(BUFFER\_SIZE);

vector<int> buffer2(BUFFER\_SIZE);

size\_t index1 = 0, index2 = 0;

while (index1 < BUFFER\_SIZE && input1 >> buffer1[index1])

{

index1++;

}

ioReadCount++;

while (index2 < BUFFER\_SIZE && input2 >> buffer2[index2])

{

index2++;

}

ioReadCount++;

while (index1 > 0 || index2 > 0)

{

if (index1 == 0)

{

output << buffer2[0] << " ";

copy(buffer2.begin() + 1, buffer2.begin() + index2, buffer2.begin());

index2--;

if (input2 >> buffer2[index2])

{

index2++;

}

}

else if (index2 == 0)

{

output << buffer1[0] << " ";

copy(buffer1.begin() + 1, buffer1.begin() + index1, buffer1.begin());

index1--;

if (input1 >> buffer1[index1])

{

index1++;

}

}

else if (buffer1[0] < buffer2[0])

{

output << buffer1[0] << " ";

copy(buffer1.begin() + 1, buffer1.begin() + index1, buffer1.begin());

index1--;

if (input1 >> buffer1[index1])

{

index1++;

}

}

else

{

output << buffer2[0] << " ";

copy(buffer2.begin() + 1, buffer2.begin() + index2, buffer2.begin());

index2--;

if (input2 >> buffer2[index2])

{

index2++;

}

}

}

input1.close();

input2.close();

output.close();

}

// Function to perform two-way merge of sorted chunks

int twoWayMerge(int totalChunks)

{

int currentChunks = totalChunks;

int round = 1;

while (currentChunks > 1)

{

int newChunks = (currentChunks + 1) / 2;

for (int i = 0; i < currentChunks; i += 2)

{

string file1 = to\_string(round - 1) + "temp" + to\_string(i) + ".txt";

if (i + 1 >= currentChunks)

{

string outputFile = to\_string(round) + "temp" + to\_string(i / 2) + ".txt";

mergeChunks(file1, outputFile);

}

else

{

string file2 = to\_string(round - 1) + "temp" + to\_string(i + 1) + ".txt";

string outputFile = to\_string(round) + "temp" + to\_string(i / 2) + ".txt";

mergeChunks(file1, file2, outputFile);

}

}

currentChunks = newChunks;

round++;

}

return round - 1;

}

int main()

{

const string inputFileName = "input.txt";

const int N = 8000;

generateInputFile(inputFileName, N);

ifstream input(inputFileName);

int totalData = countData(inputFileName);

int totalChunks = (totalData + RUN\_SIZE - 1) / RUN\_SIZE;

input.clear();

input.seekg(0);

// Sort and write each chunk

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

{

sortAndWriteChunk(input, i);

}

// Merge all chunks

int num = twoWayMerge(totalChunks);

// Build output file name

string tempFileName = to\_string(num) + "temp0.txt";

string outputFileName = "output.txt";

// Rename the temporary file to output file

filesystem::rename(tempFileName, outputFileName);

cout << "Sorting and merging completed. Output written to " << outputFileName << endl;

cout << "Total I/O Read Count: " << ioReadCount << endl;

cout << "Total I/O Write Count: " << ioWriteCount << endl;

cout << ioReadCount + ioWriteCount << endl;

}