Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет», ПНИПУ

ЛАБОРАТОРНАЯ РАБОТА

СЛОЖНЫЕ СОРТИРОВКИ

Выполнил: студент группы РИС-23-3б

Артем Владимирович Швецов

Проверила: доцент кафедры ИТАС

Ольга Андреевна Полякова

Пермь 2024

**Постановка задачи**

Реализовать методы сложных сортировок: естественного слияния, многофазного слияния.

**Анализ задачи**

1. Данные методы сортировок основываются на взаимодействии с файловой структурой, необходимы методы взаимодействия с файлами и директориями.
2. При запуске программы необходимо разметить файловое пространство, которое будет использовано в ходе выполнения сортировок.
3. Для естественной сортировки: файлы формируются на основании нахождения уже отсортированной серии в файле не отсортированных значений, после полученные файлы попарно «сливаются» до получения отсортированного файла.
4. Для многофазной сортировки: файлы формируются из серий конкретных длин, в файле значения сортируются простыми методами сортировки, определяется полученное количество с помощью определения ближайших чисел серии Фибоначчи, после полученные файлы из разных директорий попарно «сливаются» до получения отсортированного файла.

**Блок-схема**

См. в конце отчета

**Код**

FilePrepairing.h

#pragma once

#include <string>

void PrepairFolders();

void CreateUnsortedFile(int);

void ReadUnsortedFile();

void ReadSortedFile();

void DivideFile(std::string, std::string);

void ReadSortedFiles(std::string);

std::string defineFilePath(int, std::string);

FilePrepairing.cpp

#include "FilePrepairing.h"

#include <filesystem>

#include <fstream>

#include <string>

#include <random>

#include <iostream>

namespace fs = std::filesystem;

std::string defineFilePath(int fileIndex, std::string folder) {

std::string pathFolder;

if (folder == "n") {

pathFolder = "../files/natural/";

}

else if (folder == "m") {

pathFolder = "../files/multiphase/primary/";

}

else if (folder == "m1") {

pathFolder = "../files/multiphase/container1/";

}

else if (folder == "m2") {

pathFolder = "../files/multiphase/container2/";

}

else if (folder == "m3") {

pathFolder = "../files/multiphase/container3/";

}

std::string pathBase;

if (fileIndex != 0) {

pathBase = std::to\_string(fileIndex);

}

else if (fileIndex == 0) {

pathBase = "buffer";

}

std::string pathDat = ".bin";

std::string pathNameSort = pathFolder + pathBase + pathDat;

return pathNameSort;

}

void PrepairFolders() {

fs::create\_directory("../files/");

fs::create\_directory("../files/multiphase/");

fs::create\_directory("../files/multiphase/primary/");

fs::create\_directory("../files/multiphase/container1/");

fs::create\_directory("../files/multiphase/container2/");

fs::create\_directory("../files/multiphase/container3/");

fs::create\_directory("../files/natural/");

}

void CreateUnsortedFile(int count) {

std::string pathName = "../files/unsorted.bin";

FILE\* createdFile = fopen(pathName.c\_str(), "wb");

for (int i = 0; i < count; ++i) {

int writeValue = rand() % 100;

fwrite(&writeValue, sizeof(int), 1, createdFile);

}

fclose(createdFile);

}

void ReadUnsortedFile() {

std::string pathName = "../files/unsorted.bin";

FILE\* createdFile = fopen(pathName.c\_str(), "rb");

int readValue;

while (fread(&readValue, sizeof(int), 1, createdFile)) {

std::cout << readValue << '/';

}

std::cout << std::endl;

fclose(createdFile);

}

void ReadSortedFile() {

std::string pathName = "../files/sorted.bin";

FILE\* createdFile = fopen(pathName.c\_str(), "rb");

int readValue = 0;

int readValueCount = 0;

while (fread(&readValue, sizeof(int), 1, createdFile)) {

readValueCount++;

std::cout << readValue << '/';

}

std::cout << std::endl << readValueCount << std::endl;

fclose(createdFile);

}

void ReadSortedFiles(std::string folder) {

int fileIndex = 1;

std::string pathName = defineFilePath(fileIndex, folder);

FILE\* sortedFile;

while (sortedFile = fopen(pathName.c\_str(), "rb")) {

int readValue;

while (fread(&readValue, sizeof(int), 1, sortedFile)) {

std::cout << readValue << '/';

}

std::cout << std::endl;

fclose(sortedFile);

fileIndex++;

pathName = defineFilePath(fileIndex, folder);

}

}

int multMod = 100;

void DivideFile(std::string pathNameUnsort, std::string folder) {

FILE\* unsortedFile = fopen(pathNameUnsort.c\_str(), "rb");

int readValue, bufferValue = 0;

int fileIndex = 1;

if (folder == "n") bufferValue = -INT\_MIN;

else if (folder == "m") bufferValue = 1;

std::string pathNameSort = defineFilePath(fileIndex, folder);

FILE\* sortedFile = fopen(pathNameSort.c\_str(), "wb");

while (fread(&readValue, sizeof(int), 1, unsortedFile)) {

if ((folder == "n" && readValue >= bufferValue) ||

(folder == "m" && bufferValue <= multMod))

{

fwrite(&readValue, sizeof(int), 1, sortedFile);

if (folder == "n") bufferValue = readValue;

else if (folder == "m") bufferValue++;

}

else

{

fclose(sortedFile);

fileIndex++;

pathNameSort = defineFilePath(fileIndex, folder);

sortedFile = fopen(pathNameSort.c\_str(), "wb");

fwrite(&readValue, sizeof(int), 1, sortedFile);

if (folder == "n") bufferValue = readValue;

else if (folder == "m") bufferValue = 1;

}

}

fclose(sortedFile);

}

NaturalMerge.h

#pragma once

void SortNaturalMergePair(int, int);

void SortNaturalMerge();

NaturalMerge.cpp

#include "FilePrepairing.h"

#include <filesystem>

#include <fstream>

#include <string>

namespace fs = std::filesystem;

void SortNaturalMergePair(int inIndex, int outIndex) {

std::string path1 = defineFilePath(inIndex, "n");

std::string path2 = defineFilePath(inIndex + 1, "n");

std::string pathb = defineFilePath(0, "n");

FILE\* inFile1 = fopen(path1.c\_str(), "rb");

FILE\* inFile2 = fopen(path2.c\_str(), "rb");

FILE\* outFile = fopen(pathb.c\_str(), "wb");

int inputValue1;

int inputValue2;

bool isAddedValue1 = false;

bool isAddedValue2 = false;

fread(&inputValue1, sizeof(int), 1, inFile1);

fread(&inputValue2, sizeof(int), 1, inFile2);

while (!feof(inFile1) && !feof(inFile2)) {

if (inputValue1 < inputValue2) {

fwrite(&inputValue1, sizeof(int), 1, outFile);

isAddedValue1 = true;

isAddedValue2 = false;

}

else {

fwrite(&inputValue2, sizeof(int), 1, outFile);

isAddedValue2 = true;

isAddedValue1 = false;

}

if (isAddedValue1) {

fread(&inputValue1, sizeof(int), 1, inFile1);

}

if (isAddedValue2) {

fread(&inputValue2, sizeof(int), 1, inFile2);

}

}

if (!feof(inFile1)) {

fwrite(&inputValue1, sizeof(int), 1, outFile);

while (fread(&inputValue1, sizeof(int), 1, inFile1)) {

fwrite(&inputValue1, sizeof(int), 1, outFile);

}

}

else if (!feof(inFile2)) {

fwrite(&inputValue2, sizeof(int), 1, outFile);

while (fread(&inputValue2, sizeof(int), 1, inFile2)) {

fwrite(&inputValue2, sizeof(int), 1, outFile);

}

}

fclose(inFile1);

fclose(inFile2);

fclose(outFile);

remove(path1.c\_str());

remove(path2.c\_str());

inFile1 = fopen(pathb.c\_str(), "rb");

path1 = defineFilePath(outIndex, "n");

outFile = fopen(path1.c\_str(), "wb");

while (fread(&inputValue1, sizeof(int), 1, inFile1)){

fwrite(&inputValue1, sizeof(int), 1, outFile);

}

fclose(outFile);

fclose(inFile1);

remove(pathb.c\_str());

}

void SortNaturalMerge() {

int inIndex = 1;

int outIndex = 1;

std::string path1 = defineFilePath(inIndex, "n");

std::string path2 = defineFilePath(inIndex + 1, "n");

if (!fs::exists(path1)) {

return;

}

while (!(inIndex == 1 && !fs::exists(path2)))

{

while (fs::exists(path1))

{

if (!fs::exists(path2)) {

FILE\* inFile2 = fopen(path2.c\_str(), "wb");

fclose(inFile2);

}

SortNaturalMergePair(inIndex, outIndex);

inIndex += 2;

outIndex++;

path1 = defineFilePath(inIndex, "n");

path2 = defineFilePath(inIndex + 1, "n");

}

inIndex = 1;

outIndex = 1;

path1 = defineFilePath(inIndex, "n");

path2 = defineFilePath(inIndex + 1, "n");

}

std::string path = defineFilePath(1, "n");

FILE\* inFile = fopen(path.c\_str(), "rb");

FILE\* outFile = fopen("../files/sorted.bin", "wb");

int outvalue;

while (fread(&outvalue, sizeof(int), 1, inFile)) {

fwrite(&outvalue, sizeof(int), 1, outFile);

}

fclose(inFile);

fclose(outFile);

remove(path.c\_str());

}

Multiphase.h

#pragma once

void SortMultiphase();

std::pair<int, int> SortMultiphasePrepairGroups();

void SortMultiphaseSingleIteration(int, int, int, int);

Multiphase.cpp

#include "FilePrepairing.h"

#include <filesystem>

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

namespace fs = std::filesystem;

std::pair<int, int> GetFibonnachiNumbers(int numberToCompareTo, std::pair<int, int> PreviousTwoNumbers)

{

int currentNumber = PreviousTwoNumbers.first + PreviousTwoNumbers.second;

std::pair <int, int> PairToReturn;

if (currentNumber < numberToCompareTo)

PairToReturn = GetFibonnachiNumbers(numberToCompareTo, { PreviousTwoNumbers.second, currentNumber });

else

PairToReturn = PreviousTwoNumbers;

return PairToReturn;

}

std::pair<int, int> GetFibonnachiNumbers(int numberToCompareTo)

{

return GetFibonnachiNumbers(numberToCompareTo, { 0, 1 });

}

std::pair<int, int> SortMultiphasePrepairGroups() {

int inIndex = 1;

std::string path = defineFilePath(inIndex, "m");

while (fs::exists(path)) {

std::vector<int> vect;

FILE\* inFile = fopen(path.c\_str(), "rb");

int outvalue;

while (fread(&outvalue, sizeof(int), 1, inFile)) {

vect.push\_back(outvalue);

}

fclose(inFile);

fs::remove(path);

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

FILE\* outFile = fopen(path.c\_str(), "wb");

while (!vect.empty()) {

outvalue = vect[0];

fwrite(&outvalue, sizeof(int), 1, outFile);

vect.erase(vect.begin());

}

fclose(outFile);

inIndex++;

path = defineFilePath(inIndex, "m");

}

std::pair<int, int> groupCount = GetFibonnachiNumbers(inIndex-1);

while (inIndex <= groupCount.first + groupCount.second) {

std::string path = defineFilePath(inIndex, "m");

FILE\* inFile1 = fopen(path.c\_str(), "wb");

fclose(inFile1);

inIndex++;

}

inIndex = 1;

int newIndex = 1;

for (int i = 0; i < groupCount.first; i++) {

std::string path1 = defineFilePath(inIndex, "m");

std::string path2 = defineFilePath(newIndex, "m1");

FILE\* inFile = fopen(path1.c\_str(), "rb");

FILE\* outFile = fopen(path2.c\_str(), "wb");

int outvalue;

while (fread(&outvalue, sizeof(int), 1, inFile)) {

fwrite(&outvalue, sizeof(int), 1, outFile);

}

fclose(inFile);

fclose(outFile);

fs::remove(path1);

inIndex++;

newIndex++;

}

newIndex = 1;

for (int i = 0; i < groupCount.second; i++) {

std::string path1 = defineFilePath(inIndex, "m");

std::string path2 = defineFilePath(newIndex, "m2");

FILE\* inFile = fopen(path1.c\_str(), "rb");

FILE\* outFile = fopen(path2.c\_str(), "wb");

int outvalue;

while (fread(&outvalue, sizeof(int), 1, inFile)) {

fwrite(&outvalue, sizeof(int), 1, outFile);

}

fclose(inFile);

fclose(outFile);

fs::remove(path1);

inIndex++;

newIndex++;

}

return groupCount;

}

void SortMultiphaseSingleIteration(int fIndex, int from1, int from2, int to1) {

std::string path1;

std::string path2;

std::string path3;

switch (from1)

{

case 1:

path1 = defineFilePath(fIndex, "m1");

break;

case 2:

path1 = defineFilePath(fIndex, "m2");

break;

case 3:

path1 = defineFilePath(fIndex, "m3");

break;

}

switch (from2)

{

case 1:

path2 = defineFilePath(fIndex, "m1");

break;

case 2:

path2 = defineFilePath(fIndex, "m2");

break;

case 3:

path2 = defineFilePath(fIndex, "m3");

break;

}

switch (to1)

{

case 1:

path3 = defineFilePath(fIndex, "m1");

break;

case 2:

path3 = defineFilePath(fIndex, "m2");

break;

case 3:

path3 = defineFilePath(fIndex, "m3");

break;

}

FILE\* inFile1 = fopen(path1.c\_str(), "rb");

FILE\* inFile2 = fopen(path2.c\_str(), "rb");

FILE\* outFile = fopen(path3.c\_str(), "wb");

int inputValue1;

int inputValue2;

bool isAddedValue1 = false;

bool isAddedValue2 = false;

fread(&inputValue1, sizeof(int), 1, inFile1);

fread(&inputValue2, sizeof(int), 1, inFile2);

while (!feof(inFile1) && !feof(inFile2)) {

if (inputValue1 < inputValue2) {

fwrite(&inputValue1, sizeof(int), 1, outFile);

isAddedValue1 = true;

isAddedValue2 = false;

}

else {

fwrite(&inputValue2, sizeof(int), 1, outFile);

isAddedValue2 = true;

isAddedValue1 = false;

}

if (isAddedValue1) {

fread(&inputValue1, sizeof(int), 1, inFile1);

}

if (isAddedValue2) {

fread(&inputValue2, sizeof(int), 1, inFile2);

}

}

if (!feof(inFile1)) {

fwrite(&inputValue1, sizeof(int), 1, outFile);

while (fread(&inputValue1, sizeof(int), 1, inFile1)) {

fwrite(&inputValue1, sizeof(int), 1, outFile);

}

}

else if (!feof(inFile2)) {

fwrite(&inputValue2, sizeof(int), 1, outFile);

while (fread(&inputValue2, sizeof(int), 1, inFile2)) {

fwrite(&inputValue2, sizeof(int), 1, outFile);

}

}

fclose(inFile1);

fclose(inFile2);

fclose(outFile);

remove(path1.c\_str());

remove(path2.c\_str());

}

void SortMultiphase() {

int biggerPath, lesserPath, targetPath;

std::string path1;

std::string path2;

std::pair<int, int> currentCount = SortMultiphasePrepairGroups();

biggerPath = 2;

lesserPath = 1;

targetPath = 3;

while (currentCount.first + currentCount.second > 1) {

for (int i = 1; i <= currentCount.first; i++) {

SortMultiphaseSingleIteration(i, biggerPath, lesserPath, targetPath);

}

for (int i = currentCount.first + 1; i <= currentCount.second; i++) {

switch (biggerPath)

{

case 1:

path1 = defineFilePath(i, "m1");

path2 = defineFilePath(i - currentCount.first, "m1");

break;

case 2:

path1 = defineFilePath(i, "m2");

path2 = defineFilePath(i - currentCount.first, "m2");

break;

case 3:

path1 = defineFilePath(i, "m3");

path2 = defineFilePath(i - currentCount.first, "m3");

break;

}

rename(path1.c\_str(), path2.c\_str());

}

int temp = lesserPath;

lesserPath = biggerPath;

biggerPath = targetPath;

targetPath = temp;

currentCount.second -= currentCount.first;

std::swap(currentCount.first, currentCount.second);

}

switch (biggerPath)

{

case 1:

path1 = defineFilePath(1, "m1");

break;

case 2:

path1 = defineFilePath(1, "m2");

break;

case 3:

path1 = defineFilePath(1, "m3");

break;

}

FILE\* inFile = fopen(path1.c\_str(), "rb");

FILE\* outFile = fopen("../files/sorted.bin", "wb");

int outvalue;

while (fread(&outvalue, sizeof(int), 1, inFile)) {

fwrite(&outvalue, sizeof(int), 1, outFile);

}

fclose(inFile);

fclose(outFile);

remove(path1.c\_str());

}

Main.cpp

#include "FilePrepairing.h"

#include "NaturalMerge.h"

#include "Multiphase.h"

int main() {

PrepairFolders();

CreateUnsortedFile(10000);

ReadUnsortedFile();

DivideFile("../files/unsorted.bin", "m");

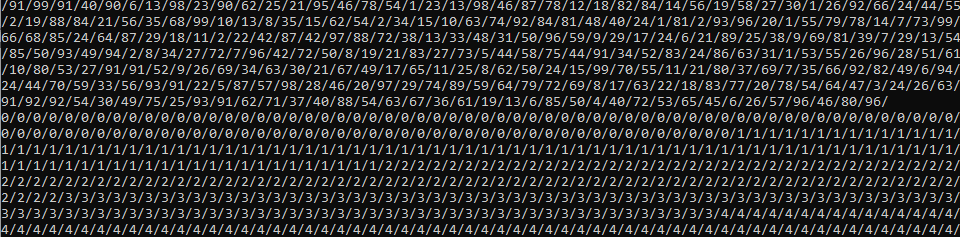
SortMultiphase(); //SortNaturalMerge();

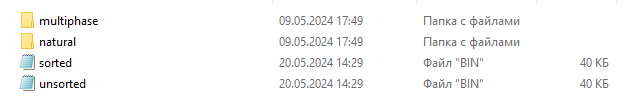
ReadSortedFile();

return 0;

}

**Решение**





**Выводы**

В ходе выполнения работы были изучены методы сложных сортировок, а также более углубленно изучены методы взаимодействия с файловой структурой.

**Github**

<https://github.com/Hitikov/Lab_Diff_Sortings>