**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 2 з дисципліни

«Проектування алгоритмів»

„ **Проектування структур даних**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-02 Василенко Павло Олександрович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Вєчєрковська А.С.*

Київ 2021

Зміст

[1 Мета лабораторної роботи 3](#_Toc81070524)

[2 Завдання 4](#_Toc81070525)

[3 Виконання 7](#_Toc81070526)

[3.1 Псевдокод алгоритмів 7](#_Toc81070527)

[3.2 Часова складність пошуку 7](#_Toc81070528)

[3.3 Програмна реалізація 7](#_Toc81070529)

[3.3.1 Вихідний код 7](#_Toc81070530)

[3.3.2 Приклади роботи 7](#_Toc81070531)

[3.4 Тестування алгоритму 8](#_Toc81070532)

[3.4.1 Часові характеристики оцінювання 8](#_Toc81070533)

[Висновок 9](#_Toc81070534)

[Критерії оцінювання 10](#_Toc81070535)

# Мета лабораторної роботи

Мета роботи – вивчити основні підходи проектування та обробки складних структур даних.

## Завдання

Відповідно до варіанту (таблиця 2.1), записати алгоритми пошуку, додавання, видалення і редагування запису в структурі даних за допомогою псевдокоду (чи іншого способу по вибору).

Записати часову складність пошуку в структурі в асимптотичних оцінках.

Виконати програмну реалізацію невеликої СУБД, з функціями пошуку (алгоритм пошуку у вузлі структури згідно варіанту таблиця 2.1, за необхідності), додавання, видалення та редагування записів (запис складається із ключа і даних, ключі унікальні і цілочисельні, даних може бути декілька полів для одного ключа, але достатньо одного рядка фіксованої довжини). Для зберігання даних використовувати структуру даних згідно варіанту (таблиця 2.1).

Заповнити базу випадковими значеннями до 10000 і зафіксувати середнє (із 10-15 пошуків) число порівнянь для знаходження запису по ключу.

Зробити висновок з лабораторної роботи.

Таблиця 2.1 – Варіанти алгоритмів

|  |  |
| --- | --- |
| **№** | **Структура даних** |
| 1 | Файли з щільним індексом з перебудовою індексної області, бінарний пошук |
| 2 | Файли з щільним індексом з областю переповнення, бінарний пошук |
| 3 | Файли з не щільним індексом з перебудовою індексної області, бінарний пошук |
| 4 | Файли з не щільним індексом з областю переповнення, бінарний пошук |
| 5 | АВЛ-дерево |
| 6 | Червоно-чорне дерево |
| 7 | B-дерево t=10, бінарний пошук |
| 8 | B-дерево t=25, бінарний пошук |
| 9 | B-дерево t=50, бінарний пошук |
| 10 | B-дерево t=100, бінарний пошук |
| 11 | Файли з щільним індексом з перебудовою індексної області, однорідний бінарний пошук |
| 12 | Файли з щільним індексом з областю переповнення, однорідний бінарний пошук |
| 13 | Файли з не щільним індексом з перебудовою індексної області, однорідний бінарний пошук |
| 14 | Файли з не щільним індексом з областю переповнення, однорідний бінарний пошук |
| 15 | АВЛ-дерево |
| 16 | Червоно-чорне дерево |
| 17 | B-дерево t=10, бінарний пошук |
| 18 | B-дерево t=25, бінарний пошук |
| 19 | B-дерево t=50, бінарний пошук |
| 20 | B-дерево t=100, бінарний пошук |
| 21 | Файли з щільним індексом з перебудовою індексної області, метод Шарра |
| 22 | Файли з щільним індексом з областю переповнення, метод Шарра |
| 23 | Файли з не щільним індексом з перебудовою індексної області, метод Шарра |
| 24 | Файли з не щільним індексом з областю переповнення, метод Шарра |
| 25 | АВЛ-дерево |
| 26 | Червоно-чорне дерево |
| 27 | B-дерево t=10, метод Шарра |
| 28 | B-дерево t=25, метод Шарра |
| 29 | B-дерево t=50, метод Шарра |
| 30 | B-дерево t=100, метод Шарра |

# Виконання

## Псевдокод алгоритмів

Insert: (string data)

Int key = generateRandomKey();

If(not indexSegmend.has(key)):

dataSegment.push\_back(key, data)

int indexDataPointer = dataSegment.size()

forEach(block of indexSegmant.blocks):

if(key>=block.min AND key<=block.max):

block->push\_back(key,indexDataPointer)

Select: (int key)

forEach(block of indexSegment.blocks):

if(key>=block.min AND key<=block.max):

return block.getByKey(key)

return overflowArea.getByKey(key)

getByKey: (int key)

if(records.size()==0) return null

if(records.size()==1) return records[0].key==key

result = binarySearch(key,0,records.size()-1)

return result

BinarySearch: (key, left, right)

While(left<=right):

Mid=left+(right-left)/2

If(records[mid].key==key) return records[mid]

If(records[mid].key<key) left=m+1;

Else: right=m-1;

Return null

Delete: (int key)

ForEach(record of dataSegment.records):

If(record.key==key) record.isDeleted=true

Rewrite file

ForEach(block of indexSegment.blocks):

if(key>=block.min AND key<=block.max):

forEach(record of block.records):

if(record.key==key) records.Delete(record)

Rewrite file

Update: (int key, string newData)

forEach(block of indexSegment.blocks):

if(key>=block.min AND key<=block.max):

dataPointer = block.getByKey(key)

dataSegment.records[dataPointer-1].dataValue = newData

dataPointer = overflowArea.getByKey(key)

dataSegment.records[dataPointer-1].dataValue = newData

getByKey: (int key)

if(records.size()==0) return null

if(records.size()==1) return records[0].key==key

result = binarySearch(key,0,records.size()-1)

return result

## Часова складність пошуку

В лабораторній роботі я мав 2 файли: в індексному файлі ключі відсортовано, а в основному маємо всі дані невідсортованими. Для індексного файлу було використано бінарний пошук. Так як двійковий пошук кожну ітерації відсіює половину варіантів, то його часова складність O(log n).

## Програмна реалізація

### Вихідний код

main.cpp

#include <iostream>

#include "DenseIndexFile.h"

#include <ctime>

#include <random>

using namespace std;

int main() {

srand(time(NULL));

auto denseIndexFile = new DenseIndexFile();

cout << "COMMANDS: \nSELECT data\_key \nINSERT data\_value \nDELETE data\_key \nUPDATE data\_key data\_value \nQUIT - for end of the program\nTRUNC DATABASE\n\n";

string command = "";

try {

while (command != "QUIT") {

cout << "Enter command: ";

getline(cin, command);

if (command == "TRUNC DATABASE") {

denseIndexFile->dataSegment->records.clear();

for (auto block : denseIndexFile->indexSegment->blocks) {

block->records.clear();

}

ofstream data("data\_seg.csv");

data.close();

ofstream index("index\_seg.csv");

index << "Overflow area";

}

else if (command == "FILL") {

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distribution(0, INT\_MAX);

for (int i = 0; i < 10000; i++)

{

string data = to\_string(i);

int key;

string found = "";

while (found != "Not found--system-reserved") {

key = distribution(gen);

found = denseIndexFile->getData(key);

}

denseIndexFile->insert(key, data);

}

}

else {

ifstream dataseg("data\_seg.csv");

if (!dataseg.is\_open()) {

cout << "CREATE DATABASE FIRST" << endl;

return -1;

}

dataseg.close();

ifstream indexseg("index\_seg.csv");

if (!indexseg.is\_open()) {

cout << "CREATE DATABASE FIRST" << endl;

return 1;

}

indexseg.close();

if (command.find("INSERT") == 0) {

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distribution(0, INT\_MAX);

string data = command.substr(command.find("INSERT") + 7);

int key;

string found = "";

while (found != "Not found--system-reserved") {

key = distribution(gen);

found = denseIndexFile->getData(key);

}

denseIndexFile->insert(key, data);

cout << "Added" << endl;

}

else if (command.find("SELECT") == 0) {

int key = stoi(command.substr(command.find("INSERT") + 7));

string result = denseIndexFile->getData(key);

cout << "Found: " << result << endl;

}

else if (command.find("DELETE") == 0) {

int key = stoi(command.substr(command.find("DELETE") + 7));

denseIndexFile->deleteByKey(key);

cout << "Deleted" << endl;

}

else if (command.find("UPDATE") == 0) {

//update 124658 78

string all = command.substr(command.find(' ') + 1);

int key = stoi(command.substr(7, command.find\_last\_of(' ') - 7));

string newData = all.substr(all.find(' ') + 1);

string res = denseIndexFile->updateByKey(key, newData);

cout << res << endl;

}

else {

if(command!="QUIT")

cout << "Unknown command" << endl;

}

}

}

}

catch (exception e) {

cout << e.what();

}

return 0;

}

DenseIndexFile.h

#pragma once

#include <iostream>

#include "IndexFile.h"

#include "DataSegmentFile.h"

using namespace std;

class DenseIndexFile

{

public:

IndexFile\* indexSegment;

DataSegmentFile\* dataSegment;

DenseIndexFile() {

indexSegment = new IndexFile();

dataSegment = new DataSegmentFile();

}

bool insert(int, string);

string getData(int);

void deleteByKey(int);

string updateByKey(int, string);

};

DenseIndexFile.cpp

#include "DenseIndexFile.h"

bool DenseIndexFile::insert(int key, string data)

{

if (indexSegment->search(key) != nullptr) {

return false;

}

auto dataPointer = dataSegment->insert(key, data);

indexSegment->insert(key, dataPointer);

return true;

}

string DenseIndexFile::getData(int key)

{

for (auto block : this->indexSegment->blocks) {

if (key >= block->MIN\_KEY\_VALUE && key <= block->MAX\_KEY\_VALUE) {

IndexRecord\* result = block->get(key);

if (result == nullptr) {

auto result1 = this->indexSegment->overflowArea->get(key);

if (result1 == nullptr) return "Not found--system-reserved";

string found1 = this->dataSegment->getByRowNumber(result1->dataPointer);

return found1;

}

string found = this->dataSegment->getByRowNumber(result->dataPointer);

return found;

}

}

}

void DenseIndexFile::deleteByKey(int key)

{

this->dataSegment->deleteByKey(key);

this->indexSegment->deleteByKey(key);

}

string DenseIndexFile::updateByKey(int key, string newData)

{

for (auto block : this->indexSegment->blocks) {

if (key >= block->MIN\_KEY\_VALUE && key <= block->MAX\_KEY\_VALUE) {

IndexRecord\* result = block->get(key);

if (result == nullptr) {

return "Not found";

}

string res = this->dataSegment->updateByRowNumber(result->dataPointer, newData);

return res;

}

}

}

IndexFile.h

#pragma once

#include <iostream>

#include <string>

#include <fstream>

#include <vector>

using namespace std;

struct IndexRecord {

int keyValue;

int dataPointer;

void parseLine(string line);

IndexRecord() {

keyValue = 0;

dataPointer = 0;

}

IndexRecord(int key, int data) {

keyValue = key;

dataPointer = data;

}

};

class IndexBlock {

public:

int MAX\_KEY\_VALUE, MIN\_KEY\_VALUE;

vector<IndexRecord\*> records;

IndexBlock(int,int);

void pushNewRecord(IndexRecord\*);

IndexRecord\* get(int);

IndexRecord\* binarySearch(int,int,int);

bool add(int, int);

void deleteByKey(int);

};

class IndexFile

{

static const unsigned int MAX\_KEY\_VALUE = INT\_MAX;

static const unsigned int NUMBER\_OF\_BLOCKS = 4;//index file contains 4 blocks

static const unsigned int MAX\_BLOCK\_SIZE = 1500;

public:

vector<IndexBlock\*> blocks;

IndexBlock\* overflowArea;

IndexFile() {

overflowArea = new IndexBlock(0, MAX\_KEY\_VALUE);

int BLOCK\_STEP = MAX\_KEY\_VALUE / NUMBER\_OF\_BLOCKS;

int prevMax = 0;

for (int i = 0; i < NUMBER\_OF\_BLOCKS; i++)

{

auto block = new IndexBlock(prevMax, prevMax + BLOCK\_STEP);

this->blocks.push\_back(block);

prevMax += BLOCK\_STEP + 1;

}

this->readFile();

}

void readFile();

IndexRecord\* search(int);

void insert(int, int);

void writeFile();

void deleteByKey(int);

};

IndexFile.cpp

#include "IndexFile.h"

void IndexFile::readFile()

{

ifstream file("index\_seg.csv");

if (!file.is\_open()) {

cout << "No index file found";

return;

}

string line;

getline(file, line);

while (line!="Overflow area") {

int currentBlock = 0;

auto currentIndexRecord = new IndexRecord();

if(line.empty()){

continue;

}

currentIndexRecord->parseLine(line);

while (!(currentIndexRecord->keyValue >= this->blocks[currentBlock]->MIN\_KEY\_VALUE

&& currentIndexRecord->keyValue <= this->blocks[currentBlock]->MAX\_KEY\_VALUE)) {

currentBlock++;

}

blocks[currentBlock]->pushNewRecord(currentIndexRecord);

getline(file, line);

}

getline(file, line);

while (!file.eof()) {

int currentBlock = 0;

auto currentIndexRecord = new IndexRecord();

if (line.empty()) {

continue;

}

currentIndexRecord->parseLine(line);

overflowArea->pushNewRecord(currentIndexRecord);

getline(file, line);

}

}

IndexRecord\* IndexFile::search(int keyValue)

{

for (auto block:blocks)

{

if (block->MAX\_KEY\_VALUE >= keyValue && block->MIN\_KEY\_VALUE <= keyValue) {

auto result = block->get(keyValue);

if (result == nullptr) {

result = overflowArea->get(keyValue);

}

return result;

}

}

}

void IndexFile::insert(int key, int dataPointer)

{

for (auto block : blocks) {

if (key <= block->MAX\_KEY\_VALUE && key >= block->MIN\_KEY\_VALUE) {

//auto record = new IndexRecord(key,dataPointer);

if (block->records.size() < this->MAX\_BLOCK\_SIZE) {

auto result = block->add(key, dataPointer);

if (result) {

this->writeFile();

}

return;

}

else {

auto result = overflowArea->add(key, dataPointer);

if (result) {

this->writeFile();

}

return;

}

}

}

}

void IndexFile::writeFile()

{

ofstream file("index\_seg.csv");

for (auto block : blocks) {

for (int i = 0; i < block->records.size(); i++)

{

file << block->records[i]->keyValue << ',' << block->records[i]->dataPointer << endl;

}

}

file << "Overflow area" << endl;

for (int i = 0; i < overflowArea->records.size(); i++)

{

file << overflowArea->records[i]->keyValue << ',' << overflowArea->records[i]->dataPointer << endl;

}

file.close();

}

void IndexFile::deleteByKey(int key)

{

for (auto block : blocks) {

if (key >= block->MIN\_KEY\_VALUE && key <= block->MAX\_KEY\_VALUE) {

block->deleteByKey(key);

this->writeFile();

}

}

}

IndexBlock::IndexBlock(int min=0, int max= INT\_MAX)

{

this->MAX\_KEY\_VALUE = max;

this->MIN\_KEY\_VALUE = min;

}

void IndexBlock::pushNewRecord(IndexRecord\* rec)

{

this->records.push\_back(rec);

}

IndexRecord\* IndexBlock::get(int key)

{

if (records.size() == 0) return nullptr;

if (records.size() == 1) return records[0]->keyValue == key ? records[0] : nullptr;

auto result = binarySearch(key,0,records.size()-1);

return result;

}

IndexRecord\* IndexBlock::binarySearch(int key, int left, int right)

{

int iterationCounter = 0;

while (left <= right) {

iterationCounter++;

int m = left + (right - left) / 2;

if (records[m]->keyValue == key) {

cout << "Iterations: " << iterationCounter << endl;

return records[m];

}

if (records[m]->keyValue < key)

left = m + 1;

else

right = m - 1;

}

cout << "Iterations: " << iterationCounter << endl;

return nullptr;

}

bool IndexBlock::add(int key, int dataPointer)

{

auto result = binarySearch(key, 0, records.size()-1);

if (result == nullptr) {

if (!records.empty()) {

vector<IndexRecord\*>::iterator iterator = records.begin();

for (iterator; iterator < records.end(); iterator++) {

if ((\*iterator)->keyValue >= key) break;

}

auto newRecord = new IndexRecord(key, dataPointer);

records.insert(iterator, newRecord);

}

else {

auto newRecord = new IndexRecord(key, dataPointer);

records.push\_back(newRecord);

}

return true;

}

return false;

}

void IndexBlock::deleteByKey(int key)

{

for (auto iterator = records.begin(); iterator < records.end(); ++iterator) {

if ((\*iterator)->keyValue == key) {

records.erase(iterator);

return;

}

}

}

void IndexRecord::parseLine(string line)

{

//121655,17

this->keyValue = stoi(line.substr(0, line.find(',')));

this->dataPointer = stoi(line.substr(line.find(',') + 1));

}

DataSegment.h

#pragma once

#include <string>

#include <iostream>

#include <vector>

#include <fstream>

using namespace std;

struct DataRecord {

int keyValue;

string dataValue;

bool isDeleted;

void parseLine(string);

DataRecord(){

this->keyValue = 0;

this->dataValue = "";

this->isDeleted = false;

}

DataRecord(int key, string data) {

this->dataValue = data;

this->keyValue = key;

this->isDeleted = false;

}

};

class DataSegmentFile

{

public:

vector<DataRecord\*> records;

DataSegmentFile();

void readFile();

int insert(int, string);

string getByRowNumber(int);

void deleteByKey(int);

void writeFile();

string updateByRowNumber(int, string);

};

DataSegment.cpp

#include "DataSegmentFile.h"

DataSegmentFile::DataSegmentFile()

{

this->readFile();

}

void DataSegmentFile::readFile()

{

ifstream file("data\_seg.csv");

if (!file.is\_open()) {

cout << "No data segment file found;" << endl;

}

string line;

getline(file, line);

while (!file.eof()) {

auto dataRecord = new DataRecord();

if (line.empty()) {

continue;

}

dataRecord->parseLine(line);

this->records.push\_back(dataRecord);

getline(file, line);

}

if (!line.empty()) {

auto dataRecord = new DataRecord();

dataRecord->parseLine(line);

this->records.push\_back(dataRecord);

}

}

int DataSegmentFile::insert(int key, string data)

{

auto newRecord = new DataRecord(key,data);

records.push\_back(newRecord);

ofstream file("data\_seg.csv", ios::app);

file << key << ',' << data << endl;

return records.size();

}

string DataSegmentFile::getByRowNumber(int number)

{

if(records.size() >= number)

return this->records[number-1]->dataValue;

return "Not found--system-reserved";

}

void DataSegmentFile::deleteByKey(int key)

{

for (auto record : records) {

if (record->keyValue == key) {

record->isDeleted = true;

}

}

this->writeFile();

}

void DataSegmentFile::writeFile()

{

ofstream file("data\_seg.csv");

for (auto record : records) {

if (record->isDeleted) {

file << record->keyValue<< ',' << record->dataValue<<"[DELETED]" << endl;

}

else {

file << record->keyValue << ',' << record->dataValue<< endl;

}

}

}

string DataSegmentFile::updateByRowNumber(int rowNumber, string newData)

{

if (records.size() < rowNumber - 1) return "Not found";

if (records[rowNumber - 1]->isDeleted) return "Not found(was deleted)";

records[rowNumber - 1]->dataValue = newData;

this->writeFile();

return "Updated";

}

void DataRecord::parseLine(string line)

{

//166468198,sdfsmdlfksdfl;kmsdfk

if (line.find("[DELETED]") != string::npos) {

this->keyValue = stoi(line.substr(0, line.find(',')));

this->dataValue = line.substr(line.find(',') + 1, line.find("[DELETED]") - line.find(',') -1);

this->isDeleted = true;

}

else {

this->keyValue = stoi(line.substr(0, line.find(',')));

this->dataValue = line.substr(line.find(',') + 1);

}

}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми для додавання і пошуку запису.

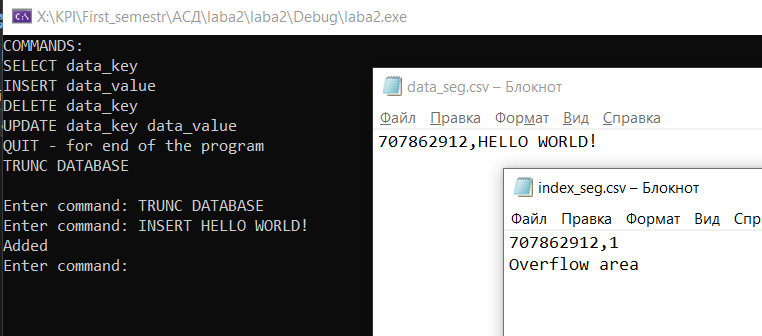


Рисунок 3.1 –Додавання запису

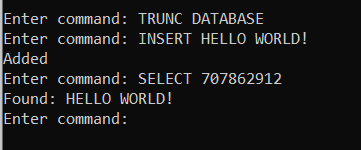


Рисунок 3.2 – Пошук запису

## Тестування алгоритму

### Часові характеристики оцінювання

В таблиці 3.1 наведено кількість порівнянь для 15 спроб пошуку запису по ключу.

Таблиця 3.1 – Число порівнянь при спробі пошуку запису по ключу

|  |  |
| --- | --- |
| Номер спроби пошуку | Число порівнянь |
| 1, пошук на 1008 рядку | 10 |
| 2, пошук на 1498 рядку | 9 |
| 3, пошук на 2019 рядку | 11 |
| 4, пошук на 2181 рядку | 11 |
| 5, пошук на 3400 рядку | 9 |
| 6, пошук на 3837 рядку | 8 |
| 7, пошук на 4000 рядку | 11 |
| 8, пошук на 4552 рядку | 10 |
| 9, пошук на 6000 рядку | 11 |
| 10, пошук на 6001 рядку | 10+11 |
| 11, пошук на 6437 рядку | 10+12 |
| 12, пошук на 7063 рядку | 11+12 |
| 13, пошук на 8402 рядку | 11+12 |
| 14, пошук на 8807 рядку | 10+12 |
| 15, пошук на 9273 рядку | 11+12 |
| Середнє значення: | 15 |

Висновок

В рамках лабораторної роботи Я вивчив роботу індексних файлів, та розробив програмну реалізацію індексного файлу зі щільним індексом з областю переповнення на прикладі власної СУБД. Основні операції над СУБД було описано за допомогою псевдокоду. Пошук по структурі даних здійснювався за допомогою двійкового пошуку.

Дана структура даних може бути корисною коли потрібно зберігати багато різних полів для одного ключа. В моїй реалізації було лише одне поле – data. Таким чином під час пошуку ми можемо не перебирати велику кількість інформації, а лише її ключ, після чого знаючи номер рядка в основній області просто перейти на нього. Бінарний пошук є ефективним, адже індексна область є впорядкованою

Критерії оцінювання

За умови здачі лабораторної роботи до 15.10.2021 включно максимальний бал дорівнює – 5. Після 15.10.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 20%;
* аналіз часової складності – 5%;
* програмна реалізація алгоритму – 60%;
* тестування алгоритму – 10%;
* висновок – 5%.

+1 додатковий бал можна отримати за реалізацію графічного інтерфейсу.

+1 додатковий бал можна отримати за реалізацію графічного зображення структури ключів.