Государственный комитет Российской Федерации по телекоммуникациям Сибирский государственный университет телекоммуникаций и информатики

Курсовая работа

По дисциплине «Структуры и алгоритмы обработки данных»

Вариант №7

Работу выполнил:

студент 2 курса

группы ЗП-21

Лощаков Юрий Сергеевич

Работу проверил:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Работа защищена

«\_\_\_» \_\_\_\_\_\_\_\_\_\_2014г.

**Новосибирск 2014**

Оглавление

[Задание для курсовой работы 2](#_Toc386520806)

[Краткое изложение основных идей и характеристик применяемых алгоритмов 3](#_Toc386520807)

[Распечатка текста программы 5](#_Toc386520808)

[Скриншоты 16](#_Toc386520809)

# Задание для курсовой работы

1. Файл базы данных загружается в динамическую память, сортируется **по дням рождения и ФИО** с использованием очередей (**метод прямого слияния**), для проведения поиска строится индексный массив. Провести поиск по **дню рождения** в упорядоченной базе, из записей с одинаковым ключом сформировать очередь. Вывести содержимое очереди. Из записей очереди построить АВЛ-дерево поиска по **дате рождения** и произвести поиск в дереве по запросу.
2. При выполнении задания главное внимание следует уделить эффективности применяемых алгоритмов, исключению всех лишних операций.
3. Операции, выражающие логически завершенные действия, рекомендуется оформлять в виде подпрограмм, грамотно выбирая между процедурами и функциями. Имена переменных и подпрограмм, параметры подпрограмм, используемые языковые конструкции должны способствовать удобочитаемости программы.
4. Для сравнения символьных строк не рекомендуется пользоваться встроенными языковыми средствами и библиотечными функциями.

# Краткое изложение основных идей и характеристик применяемых алгоритмов

Для работы программы описаны следующие структуры данных:

структура с данными

struct unitBase{

char fio[30];

UINT16 numUnit;

char job[22];

char dob[10];

}

элемент очереди

struct T\_Data{

unitBase \*data;

T\_Data \*next;

}

очередь

struct T\_Queue{

T\_Data \*head;

T\_Data \*tail;

}

структура для представления узлов дерева

struct node

{

unitBase \*key;

unsigned char height;

node\* left;

node\* right;

конструктор для создания узла дерева

node(unitBase \* k) { key = k; left = right = 0; height = 1; }

}

База данных сортируется методом прямого слияния без использования дополнительного индексного массива. Трудоёмкость методапрямого слияния определяется сложностью операции слияния серий. На каждой итерации происходит ровно n перемещений элементов списка и не более *n* сравнений. Как нетрудно видеть, количество итераций равно. n-количество элементов в списке. Тогда

C < *n*, M=*n*+*n.*

Дополнительные *n* перемещений происходят во время начального расщепления исходного списка. Асимптотические оценки для М и С имеют следующий вид

С=О (*n* log *n*), М=О (*n* log *n*) при *n → ∞*.

Метод обеспечивает устойчивую сортировку.При реализации для массивов, метод требует наличия второго вспомогательного массива, равного по размеру исходному массиву. При реализации со списками дополнительной памяти не требуется.

После сортировки базы данных строится индексный массив(index[31]) с указателями на элементы базы данных. Сортированная база данных позволяет использовать этот массив следующим образом – index[K] = указателю на крайний левый элемент базы с этим ключом поиска. Для вывода всех элементов базы с этим ключом, достаточно идти вправо по списку базы, до тех пор, пока ключ поиска совпадает со значением дня рождения в элементе списка.

Из записей очереди с одинаковым ключом строится АВЛ-дерево поиска по дате рождения(dd-mm-yy). АВЛ-дерево подразумевает, что ключи узлов уникальны. В алгоритме добавления ключа в АВЛ-дерево из методического пособия следует, что если такой ключ есть в дерево он должен игнорироваться. В базе данных присутствую совпадающие даты рождения, для корректного балансирования и добавления новых вершин в дерево была выбрана реализация, описанная в статье «АВЛ-деревья» автора Николая Ершова(<http://habrahabr.ru/post/150732/>). Данная реализация достаточно быстра и эффективна. Так как в дереве присутствуют одинаковые ключи, то для вывода всех элементов базы с заданным ключом нельзя прерывать поиск после первого нахождения ключа. После нахождения ключа требуется также проверить левое и правое поддерево узла с найденным ключом.

# Распечатка текста программы

Файлы:

AVLtree.h – описание структур и объявление функций

AVLtree.cpp – определение функций для работы АВЛ-дерева

Coursework.cpp – определение главной функции main и остальных функций сортировки и поиска

// AVLtree.h

//структура с данными

struct unitBase{

char fio[30];

UINT16 numUnit;

char job[22];

char dob[10];

};

//элемент очереди

struct T\_Data{

unitBase \*data;

T\_Data \*next;

}\*index[31];

//очередь

struct T\_Queue{

T\_Data \*head;

T\_Data \*tail;

};

// структура для представления узлов дерева

struct node

{

unitBase \*key;

unsigned char height;

node\* left;

node\* right;

node(unitBase \* k) { key = k; left = right = 0; height = 1; }

};

//контроль удаленных узлов дерева

long count = 0;

inline void SetLoc(UINT);

inline void PrintMenu();

void readFilename();

void OpenBase(T\_Queue&);

int Compare(char \*, char \*, int);

int CompareDate(char \*, char \*);

int LessDayAndFio(unitBase\*, unitBase\*);

void Search();

void addToQueue(T\_Queue&, unitBase\*);

void showQueue(T\_Queue&);

void freeQueue(T\_Queue&);

void sortMerge(T\_Queue&);

void split(T\_Queue&, T\_Queue&, T\_Queue&, int&);

void merge(T\_Queue&, int, T\_Queue&, int, T\_Queue&);

unsigned char height(node\* );

int bfactor(node\*);

void fixheight(node\*);

node\* rotateright(node\*);

node\* rotateleft(node\*);

node\* balance(node\*);

node\* insert(node\*, unitBase \*);

node\* findmin(node\*);

void find(node\*, char \*);

node\* removemin(node\*);

node\* remove(node\*, unitBase \*);

void freetree(node\*);

#include "AVLtree.cpp"

// AVLtree.cpp

// высота узла

unsigned char height(node\* p)

{

return p ? p->height : 0;

}

// для расчета фактора баланса

int bfactor(node\* p)

{

return height(p->right) - height(p->left);

}

// правка высоты узла

void fixheight(node\* p)

{

unsigned char hl = height(p->left);

unsigned char hr = height(p->right);

if (hl>hr)

{

p->height = hl + 1;

}

else

{

p->height = hr + 1;

}

//p->height = (hl>hr ? hl : hr) + 1;

}

// правый поворот вокруг p

node\* rotateright(node\* p)

{

node\* q = p->left;

p->left = q->right;

q->right = p;

fixheight(p);

fixheight(q);

return q;

}

// левый поворот вокруг q

node\* rotateleft(node\* q)

{

node\* p = q->right;

q->right = p->left;

p->left = q;

fixheight(q);

fixheight(p);

return p;

}

// балансировка узла p

node\* balance(node\* p)

{

fixheight(p);

if (bfactor(p) == 2)

{

if (bfactor(p->right) < 0)

p->right = rotateright(p->right);

return rotateleft(p);

}

if (bfactor(p) == -2)

{

if (bfactor(p->left) > 0)

p->left = rotateleft(p->left);

return rotateright(p);

}

return p; // балансировка не нужна

}

// вставка ключа k в дерево с корнем p

node\* insert(node\* p, unitBase \* k)

{

if (!p) return new node(k);

if (CompareDate(k->dob,p->key->dob)==1)

p->left = insert(p->left, k);

else

p->right = insert(p->right, k);

return balance(p);

}

// поиск узла с минимальным ключом в дереве p

node\* findmin(node\* p)

{

if (p->left)

{

return findmin(p->left);

}

return p;

//return p->left ? findmin(p->left) : p;

}

void find(node\* p, char\* k)

{

if (p)

{

if (CompareDate(k, p->key->dob) == 2)

{

puts("+");

printf\_s("%s ", p->key->fio);

printf\_s("%03d\t", p->key->numUnit);

printf\_s("%s\t", p->key->job);

printf\_s("%s\n", p->key->dob);

find(p->right, k);

find(p->left, k);

}

else

if (CompareDate(k, p->key->dob) == 0)

{

find(p->right, k);

}

else

find(p->left, k);

}

}

// удаление узла с минимальным ключом из дерева p

node\* removemin(node\* p)

{

if (p->left == 0)

return p->right;

p->left = removemin(p->left);

return balance(p);

}

// удаление ключа k из дерева p

node\* remove(node\* p, unitBase\*k)

{

if (!p) return 0;

if (CompareDate(k->dob, p->key->dob) == 1)

p->left = remove(p->left, k);

else if (CompareDate(k->dob, p->key->dob) == 0)

p->right = remove(p->right, k);

else // k == p->key

{

node\* q = p->left;

node\* r = p->right;

delete p;

if (!r) return q;

node\* min = findmin(r);

min->right = removemin(r);

min->left = q;

return balance(min);

}

return balance(p);

}

//освободить память из под дерева

void freetree(node\* tree)

{

if (tree)

{

freetree(tree->right);

freetree(tree->left);

delete tree;

count++;

tree = NULL;

}

}

//показать все элементы дерева

void showwtree(node\* p)

{

if (p)

{

puts(p->key->dob);

showwtree(p->left);

showwtree(p->right);

}

}

// Coursework.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include "rusmenu.h"

#include "AVLtree.h"

FILE \*fi;

//флаг сортировки

char checkIndex = 0;

int \_tmain(int argc, \_TCHAR\* argv[])

{

SetLoc(866);

T\_Queue queue;

queue.head =queue.tail = NULL;

while (true)

{

system("CLS");

PrintMenu();

switch (\_getch())

{

case '1': OpenBase(queue); system("pause");break;

//можно занять 1.5GB оперативной памяти!

//case '1': for (int i = 0; i < 1000; i ++ ){ OpenBase(queue); }system("pause"); break;

case '2': showQueue(queue); system("pause");break;

case '3': sortMerge(queue); system("pause");break;

case '4': Search(); system("pause");break;

case '5': freeQueue(queue); return 0;

default: break;

}

}

freeQueue(queue);

system("pause");

return 0;

}

//Устанавливает кодировку ввода и вывода в консоли

inline void SetLoc(UINT loc)

{

SetConsoleCP(loc);

SetConsoleOutputCP(loc);

}

//Выводит на консоль меню

inline void PrintMenu()

{

puts("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

printf\_s("%s\n", TextMenuId[0]);

printf\_s("%s\n", TextMenuId[1]);

printf\_s("%s\n", TextMenuId[2]);

printf\_s("%s\n", TextMenuId[3]);

printf\_s("%s\n", TextMenuId[4]);

puts("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}

//Считывание базы данных в динамическую память

void OpenBase(T\_Queue& b)

{

readFilename();

unitBase \*p;

p = (unitBase\*)malloc(sizeof(unitBase));

while (fread\_s(p->fio, sizeof(p->fio), sizeof(p->fio), 1, fi))

{

p = (unitBase\*)malloc(sizeof(unitBase));

//fseek(fi, 1, SEEK\_CUR);

fread\_s(&p->numUnit, sizeof(p->numUnit), sizeof(p->numUnit), 1, fi);

//fseek(fi, 1, SEEK\_CUR);

fread\_s(p->job, sizeof(p->job), sizeof(p->job), 1, fi);

//fseek(fi, 1, SEEK\_CUR);

fread\_s(p->dob, sizeof(p->dob), sizeof(p->dob), 1, fi);

//fseek(fi, 2, SEEK\_CUR);

addToQueue(b, p);

}

fclose(fi);

checkIndex = 0;

}

//добавления в очередь

void addToQueue(T\_Queue &queue, unitBase \*num)

{

T\_Data \*p;

p = (T\_Data\*)malloc(sizeof(T\_Data)); // выделяем память

p->next = NULL; // присваеваем указателю на следующий элемент в списке значение NUL

p->data = num;

if (queue.head != NULL) // если голова не пустая, то меняем значение следующего элемента в хвосте на новый элемент очереди

queue.tail->next = p;

else

queue.head = p;

queue.tail = p; // присваеваем хвосту сзначение нового элемента очереди

}

//Безопасное открытие файла на чтение

void readFilename()

{

errno\_t err;

char FileName[]= "testBase2.dat";

err = fopen\_s(&fi, FileName, "rb");

if (err == 0)

{

//printf\_s("The file ""%s"" was opened\n", FileName);

}

else

{

printf\_s("The file ""%s"" was not opened, try again\n", FileName);

}

}

//отображения элементов базы

void showQueue(T\_Queue &queue)

{

T\_Data \*p;

p = queue.head;

UINT8 i=0;

printf\_s("\n%s\n", TextMenuId[5]);

while (p != NULL) {

printf\_s("%s", p->data->fio);

printf\_s("%03d\t", p->data->numUnit);

printf\_s("%s\t", p->data->job);

printf\_s("%s\n", p->data->dob);

p = p->next;

i++;

if (i==20)

{

puts("ESC for exit, anykey for continue");

if(\_getch()==27)return;

i=0;

}

}

printf("\n");

}

//освобождения памяти из под списка

void freeQueue(T\_Queue &queue)

{

T\_Data \*p, \*temp;

p = queue.head;

while (p != NULL) {

temp = p;

p = p->next;

free(temp);

}

queue.head = queue.tail = NULL;

}

//объединения списков

void merge(T\_Queue &queue\_a, int q, T\_Queue &queue\_b, int r, T\_Queue &queue\_c)

{

T\_Data \*a, \*b;

a = queue\_a.head;

b = queue\_b.head;

//DO (q ≠ 0 и r ≠ 0)

// IF (a → Data ≤ b → Data)

// <Переместить элемент из списка a в очередь c>

// q:=q-1

// ELSE

// <Переместить элемент из списка b в очередь c>

// r:=r-1

// FI

//OD

//DO (q > 0)

// <переместить элемент из списка a в очередь c>

// q:=q-1

//OD

//DO (r > 0)

// <Переместить элемент из списка b в очередь c>

// r:=r-1

//OD

while ((q != 0) && (r != 0)) { // DO (q ≠ 0 и r ≠ 0)

if (LessDayAndFio(a->data, b->data)) { // IF (a → Data ≤ b → Data)

if (queue\_c.head != NULL) // <Переместить элемент из списка a в очередь c>

queue\_c.tail->next = a;

else

queue\_c.head = a;

queue\_c.tail = a;

a = a->next;

q--; // q:=q-1

}

else { // ELSE

if (queue\_c.head != NULL) // <Переместить элемент из списка b в очередь c>

queue\_c.tail->next = b;

else

queue\_c.head = b;

queue\_c.tail = b;

b = b->next;

r--; // r:=r-1

}

}

while (q > 0) { // DO (q > 0)

if (queue\_c.head != NULL) // <переместить элемент из списка a в очередь c>

queue\_c.tail->next = a;

else

queue\_c.head = a;

queue\_c.tail = a;

a = a->next;

q--; // q:=q-1

}

while (r > 0) { // DO (r > 0)

if (queue\_c.head != NULL) // <Переместить элемент из списка b в очередь c>

queue\_c.tail->next = b;

else

queue\_c.head = b;

queue\_c.tail = b;

b = b->next;

r--; // r:=r-1

}

queue\_a.head = a;

queue\_b.head = b;

}

//разделения списка

void split(T\_Queue &queue, T\_Queue &queue\_a, T\_Queue &queue\_b, int &n)

{

T\_Data \*k, \*p;

n = 1;

queue\_a.head = queue.head;

queue\_b.head = queue.head->next;

k = queue\_a.head;

p = queue\_b.head;

while (p != NULL) {

k->next = p->next;

k = p;

p = p->next;

n++;

}

}

//Сортировка базы методом прямого включения

void sortMerge(T\_Queue &queue)

{

if (queue.head==NULL)

{

return;

}

T\_Queue queue\_a, queue\_b, queue\_c[2];

int n, p, m,

i, r, q;

// Обозначим n – количество элементов в S

// queue\_a, queue\_b – рабочие списки

// queue\_c=(c0, c1) – массив из двух очередей

// p – предполагаемый размер серии

// q – фактический размер серии в списке a

// r – фактический размер серии в списке b

// m – текущее количество элементов в списках a и b

// i – номер активной очереди

queue\_a.head = queue\_a.tail =

queue\_b.head = queue\_b.tail = NULL;

split(queue, queue\_a, queue\_b, n);

p = 1;

while (p < n) {

queue\_c[0].head = queue\_c[0].tail =

queue\_c[1].head = queue\_c[1].tail = NULL;

i = 0; m = n;

while (m > 0) {

if (m >= p) q = p;

else q = m;

m = m - q;

if (m >= p) r = p;

else r = m;

m = m - r;

merge(queue\_a, q, queue\_b, r, queue\_c[i]);

i = 1 - i;

}

queue\_a.head = queue\_c[0].head;

queue\_b.head = queue\_c[1].head;

p \*= 2;

}

queue\_c[0].tail->next = NULL;

queue.head = queue\_c[0].head;

T\_Data \*temp;

int day=0;

temp = queue.head;

for (size\_t i = 0; i < 31; i++)

{

index[i] = NULL;

}

while (temp != NULL) {

day = (temp->data->dob[0] - 48) \* 10 + (temp->data->dob[1] - 48) - 1;

if (index[day] == NULL)

{

index[day] = temp;

}

temp = temp->next;

}

checkIndex = 1;

}

//Сравнение(Лексикографическое) строк

//1 если s1<s2, 2 если S1=S2, 0 если S1>S2

int Compare(char s1[], char s2[], int number)

{

int i;

i = 0;

do

{

if (s1[i]<s2[i]) return 1;

else if (s1[i]>s2[i]) return 0;

else i++;

} while (i<number);

return 2;

}

//Сравнение даты рождения 1 если < 2 если =, 0 если >

int CompareDate(char \*s1, char \*s2)

{

char ss1[2], ss2[2];

int result;

ss1[0] = s1[6]; ss1[1] = s1[7];

ss2[0] = s2[6]; ss2[1] = s2[7];

result = Compare(ss1, ss2, 2);

if (result == 1)

{

return 1;

}

if (result == 0)

{

return 0;

}

ss1[0] = s1[3]; ss1[1] = s1[4];

ss2[0] = s2[3]; ss2[1] = s2[4];

result = Compare(ss1, ss2, 2);

if (result == 1)

{

return 1;

}

if (result == 0)

{

return 0;

}

ss1[0] = s1[0]; ss1[1] = s1[1];

ss2[0] = s2[0]; ss2[1] = s2[1];

result = Compare(ss1, ss2, 2);

if (result == 1)

{

return 1;

}

if (result == 0)

{

return 0;

}

return 2;

}

//сравнение дня рождения+ФИО

int LessDayAndFio(unitBase \*p, unitBase \*q)

{

if (p->dob[0]<q->dob[0]) return 1;

else if (p->dob[0]>q->dob[0]) return 0;

else if (p->dob[1]<q->dob[1]) return 1;

else if (p->dob[1]>q->dob[1]) return 0;

else if (Compare(p->fio, q->fio, 30)) return 1;

else return 0;

}

//поиск по ключу в отсортированной базе, затем постороение дерева и поиск в нём

void Search()

{

//Если база неотсортирована, то выходим

if (!checkIndex)

{

puts("Sort base");

return;

}

int Key = 0;

//Считываем дату рождения

while (true)

{

puts(TextMenuId[6]);

scanf\_s("%d",&Key);

if (Key <= 31 && Key >= 1 && index[Key-1] != NULL)

{

break;

}

}

Key--;

T\_Data \*temp;

T\_Queue Found;

node \*root=NULL;

Found.head = Found.tail = NULL;

temp = index[Key];

//Формируем очередь из найденых элементов в базе

while (temp!=NULL)

{

if ((temp->data->dob[0] - 48) \* 10 + (temp->data->dob[1] - 48) - 1 != Key)

{

temp = NULL;

break;

}

addToQueue(Found, temp->data);

temp = temp->next;

}

temp = Found.head;

count = 0;

//Выводим на печать содержимое очереди и заполняем дерево

while (temp != NULL)

{

printf\_s("%s ", temp->data->fio);

printf\_s("%03d\t", temp->data->numUnit);

printf\_s("%s\t", temp->data->job);

printf\_s("%s\n", temp->data->dob);

root = insert(root, temp->data);

temp = temp->next;

count++;

}

printf\_s("added lists:%u\n", count);

char date[10];

//showwtree(root);

puts(TextMenuId[7]);

fflush(stdin);

scanf\_s("%s", date, \_countof(date));

find(root, date);

count = 0;

//Очищаем очередь после окончания поиска

freeQueue(Found);

////Очищаем дерево после окончания поиска

freetree(root);

root = NULL;

printf\_s("deleted lists:%u\n",count);

}

# Скриншоты



Рисунок загруженная база данных

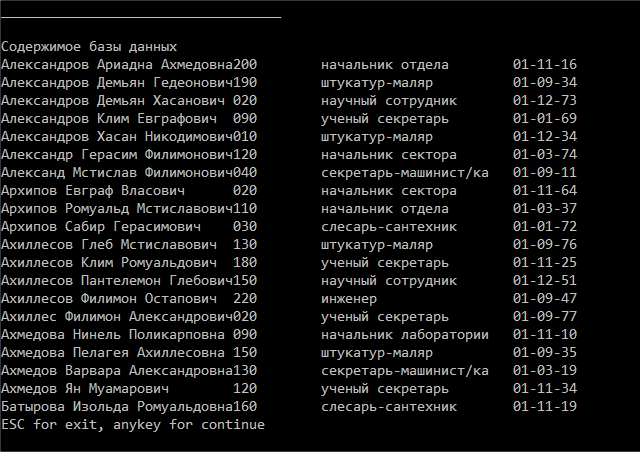


Рисунок отсортированная база

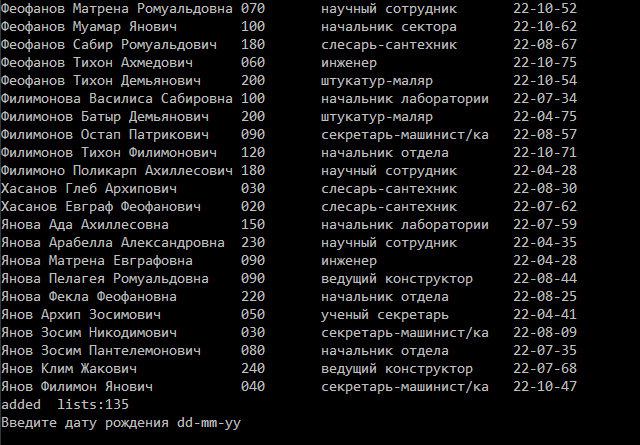


Рисунок вывод очереди после поиска 22 дня

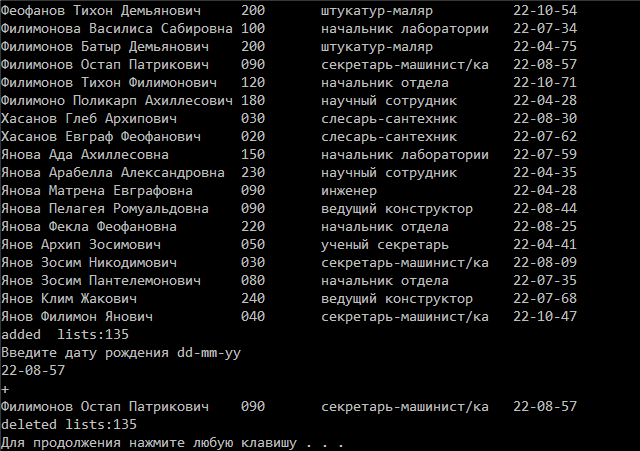


Рисунок поиск по новому ключу 22-08-57