Сибирский государственный университет телекоммуникаций и информатики

Кафедра ПМиК

Отчет по курсовой работе

по дисциплине

«Структуры и алгоритмы обработки данных»

Вариант №56

Выполнил: студент 2 курса

группы ИП-913

Босиков Владислав

Проверил:

Янченко Елена Викторовна

*Новосибирск, 2020* Содержание

1. Постановка задачи………………………………………………………...2
2. Описание программы……………………………………………………..3
3. Интерфейс работы с программой………………………………………...5
4. Результаты работы программы .................………………………………7
5. Выводы…………………………………………………………………...15
6. Текст программы………………………………………………………...16

*Постановка задачи*

Хранящуюся в файле базу данных (4000 записей) загрузить динамически в оперативную память компьютера в виде массива или списка (в зависимости от варианта), вывести на экран по 20 записей (строк) на странице с возможностью отказа от просмотра.

Упорядочить данные в соответствии с заданным условием упорядочения, используя указанный метод сортировки. Упорядоченные данные вывести на экран.

Предусмотреть возможность быстрого поиска по ключу в упорядоченной базе, в результате которого из записей с одинаковым ключом формируется очередь, содержимое очереди выводится на экран.

Из записей очереди построить дерево поиска по ключу, отличному от ключа сортировки, вывести на экран содержимое дерева и предусмотреть возможность поиска в дереве по запросу.

Закодировать файл базы данных статическим кодом, предварительно оценив вероятности всех встречающихся в ней символов. Построенный код вывести на экран, вычислить среднюю длину кодового слова и сравнить ее с энтропией исходного файла.

Дополнительное задание. Упакованную базу данных записать в файл, вычислить коэффициент сжатия данных.

*Описание программы*

Основные переменные и структуры:

const int N = 4000 – количество элементов в БД.

int code[M][M] – массив для записи закодированных символов.

int sim\_cf = 0, sim\_f = 0 – переменные для расчета степени сжатия.

struct record {

char depositor[30];

unsigned short int deposit;

char date[10];

char lawyer[22];

} – структура записи в БД.

struct list {

record \*data;

list \*next;

list \*prior;

} – структура списка из записей.

struct queue {

list \*head;

list \*tail;

} – структура для создания очереди.

struct vertex {

list \*fact;

vertex \*left;

vertex \*right;

} – структура для создания двоичного дерева.

struct GM\_code {

unsigned char a;

float p;

float q;

int l;

} – структура для кодировки методом Гилберта-Мура.

Основные функции:

void read\_base(FILE \*, list \*) – функция для считывания записей в БД и их последующего заноса в список.

void print\_record(record \*), void print\_base(list \*), void print\_stack(queue \*), void print\_tree(vertex \*&) – вспомогательные функции для вывода записи, целиком бд, очереди или дерева.

void copy\_base(list \*, list \*) – копирование списка.

void prior(list \*), void DigitalSort(list \*&, bool) – функции для сортировки методом цифровой сортировки.

void add(vertex \*&, record \*x), void DOP\_A2(vertex \*&, list \*[N], int, int), void QuickSort(int \*, list \*[N], int, int), void weight(list \*mas[N]), void tree\_search(vertex \*&, unsigned short int), void delete\_tree(vertex \*&) – функции для построения дерева оптимального поиска (алгоритм А2) и поиск в нем.

void reading(), void code\_file(), void printGMcode() – функции для кодировки методом Гилберта-Мура и добавления закодированных символов в файл.

*Интерфейс работы с программой*

Работа с программой происходит в консольном окне. При запуске программы выводится меню с 5 пунктами: «Просмотр БД», «Просмотр отсортированной БД (Digital sort)», «Поиск в отсортированной базе по ключу (вкладчик)», «Поиск в дереве (вклад)», «Статистика по кодированию». Также есть пункт «Выход».

При выборе пункта «Просмотр бд» пользователю предлагается выбор вывода: по 20 на экране, или вывести полностью весь список. При выборе вывода по 20 записей, перемещение между страницами осуществляется клавишами 1 и 2.

При выборе пункта «Просмотр отсортированной БД» пользователю предлагается выбор вывода: по 20 на экране, или вывести полностью весь список. При выборе вывода по 20 записей, перемещение между страницами осуществляется клавишами 1 - предыдущая и 2 - следующая.

При выборе пункта «Поиск в отсортированной базе по ключу (вкладчик)» пользователю предлагается ввести первые 3 буквы искомого вкладчика. После ввода найденные вкладчики выводятся на экран.

При выборе пункта «Поиск в дереве (вклад)» строится дерево из найденных при выборе 3 пункта вкладчиков, и пользователю предлагается просмотреть построенное дерево. Затем предлагается ввести сумму искомого вклада, и происходит вывод всех найденных вкладчиков с данной суммой вклада. Если такая сумма не найдена, на экран выводится соответствующее сообщение.

При выборе пункта «Статистика по кодированию» на экран выводятся все символы из БД, их кумулятивная вероятность, количество цифр в закодированном символе и, соответственно, закодированный символ. После данной таблицы выводится средняя длина закодированного символа, энтропия источника, проверка кодирования посредством сравнения энтропии+2 и средней длины закодированного символа, а также коэффициент сжатия.

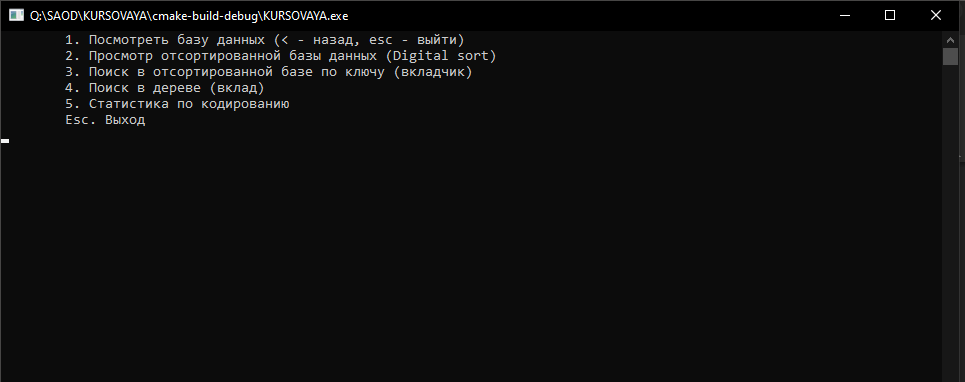
*Результаты*

Рисунок 1. Главное меню программы

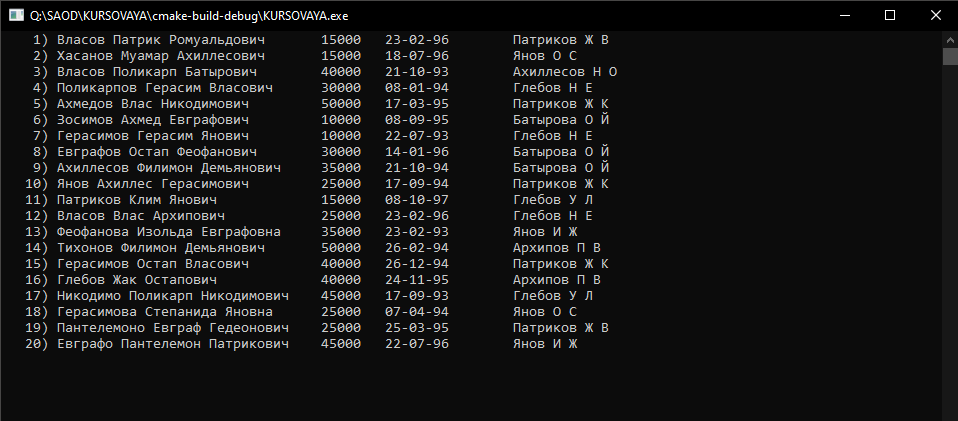
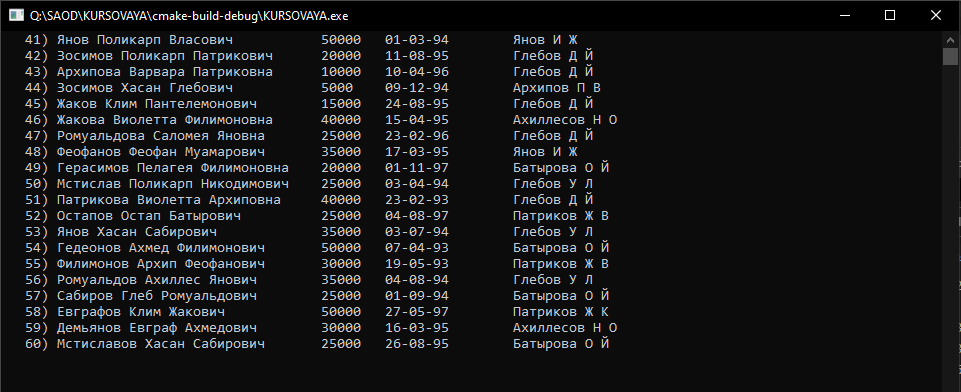
 

Рисунок 2,3. Вывод БД по 20 записей.

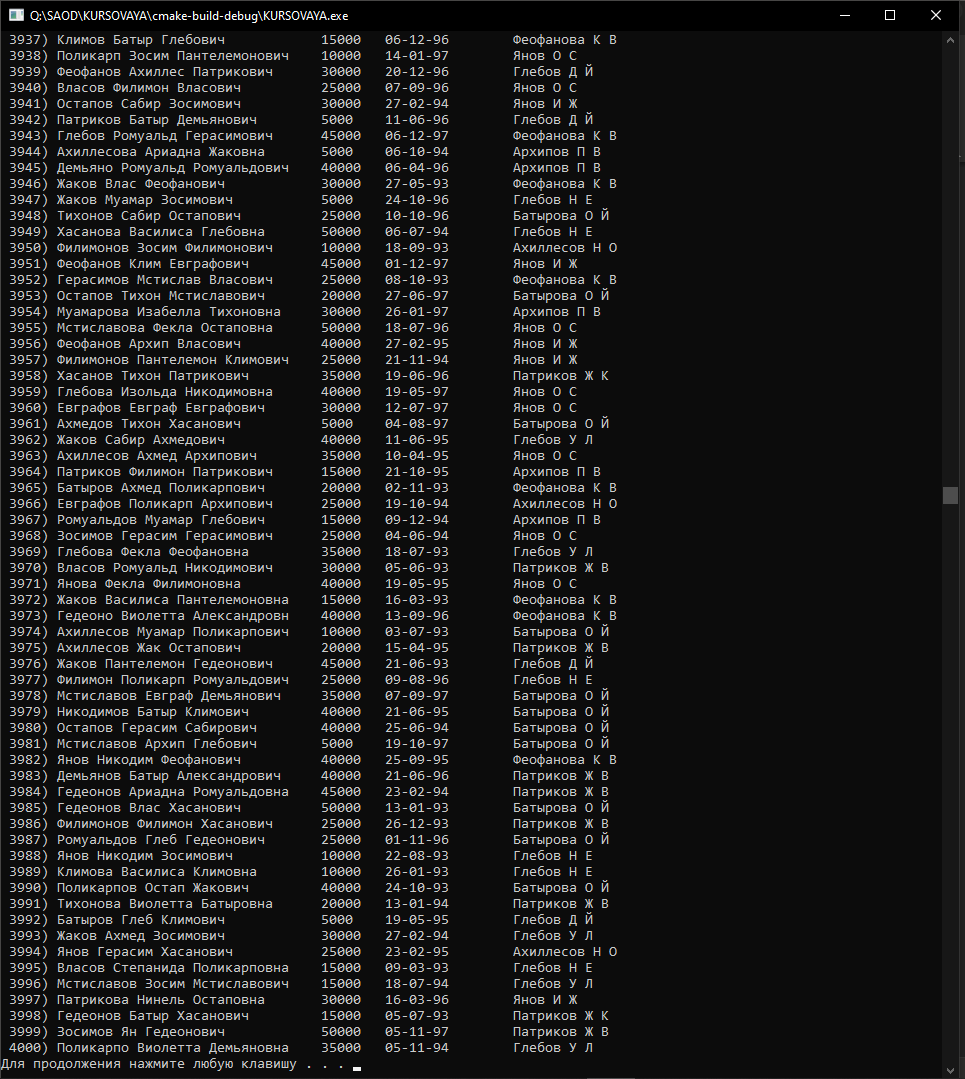


Рисунок 4. Вывод всех записей из БД

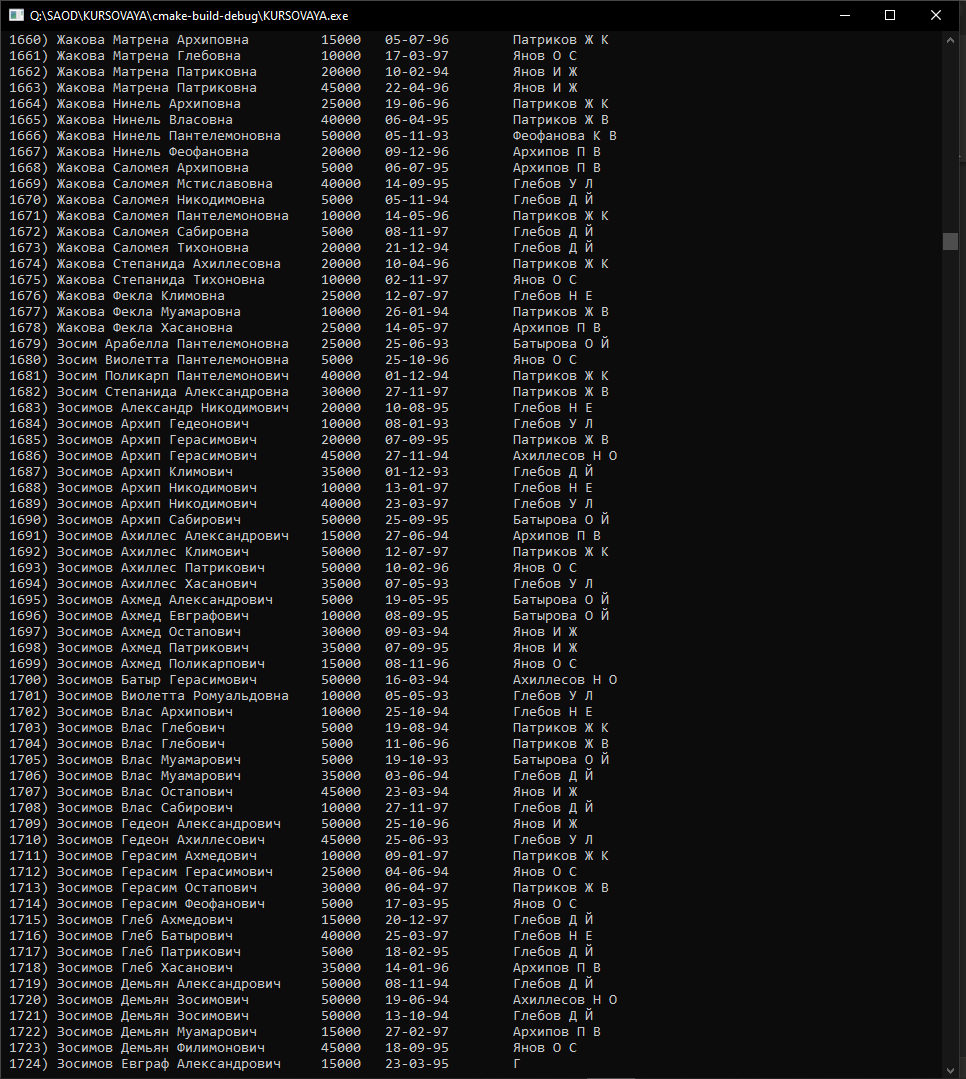


Рисунок 5. Вывод отсортированной БД

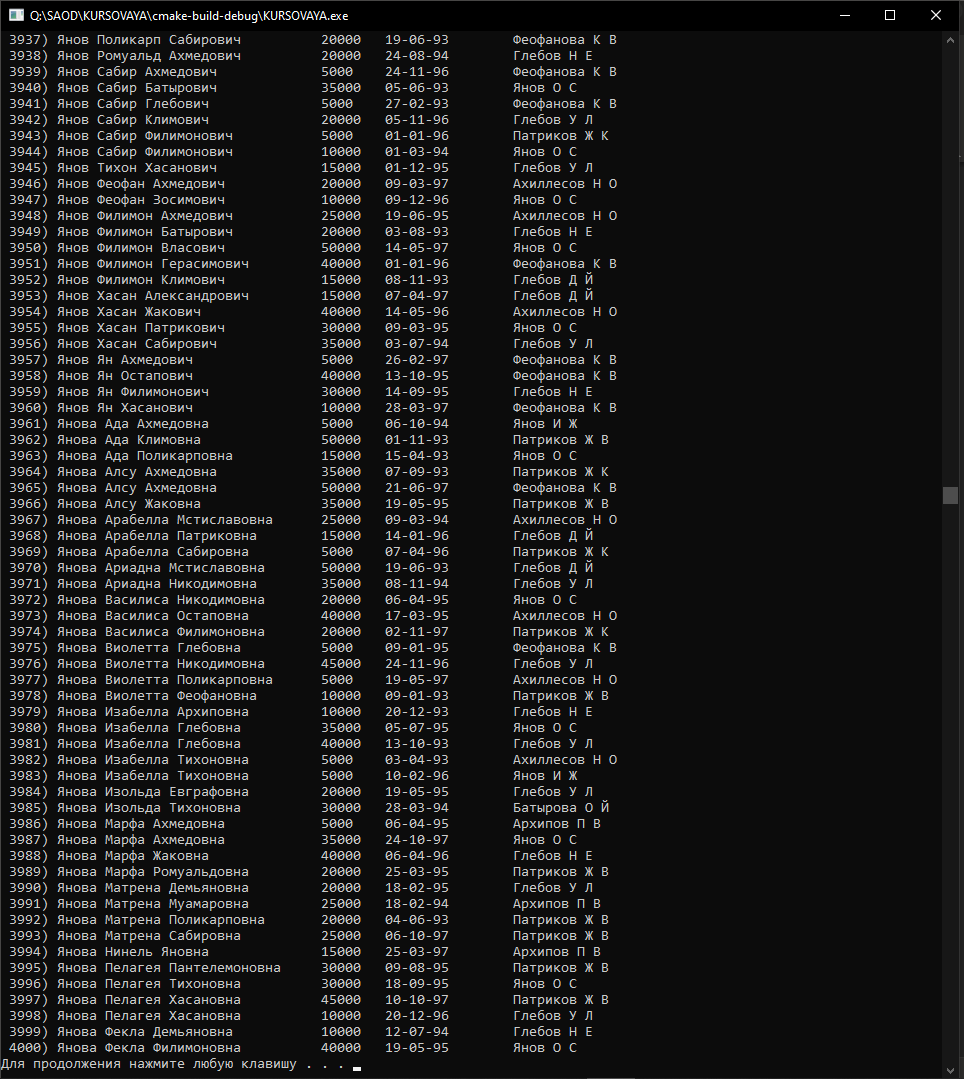


Рисунок 5. Вывод отсортированной БД

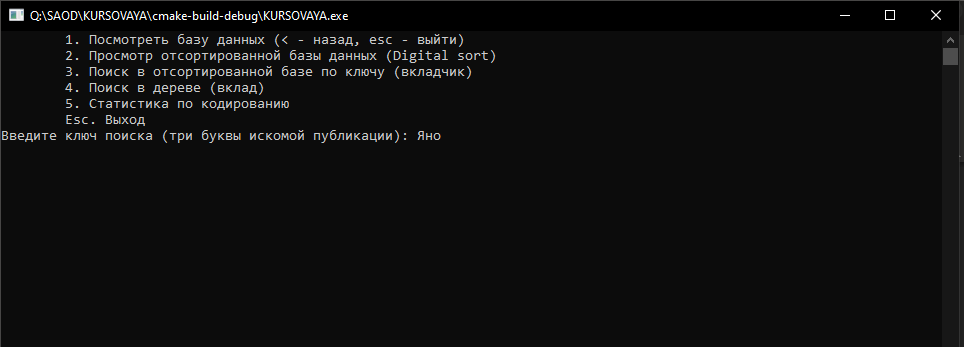


Рисунок 6. Поиск в отсортированной базе по ключу (вкладчик)

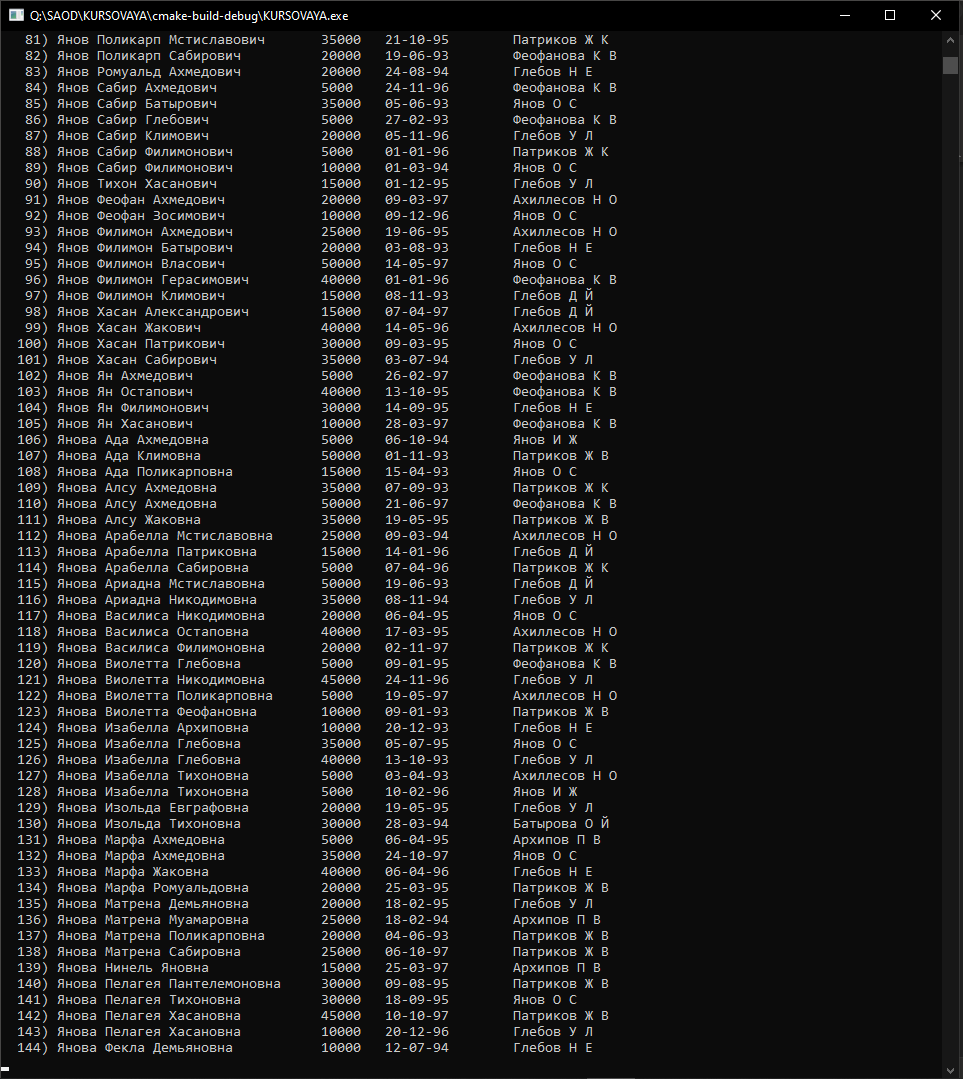


Рисунок 7. Вывод всех найденных записей

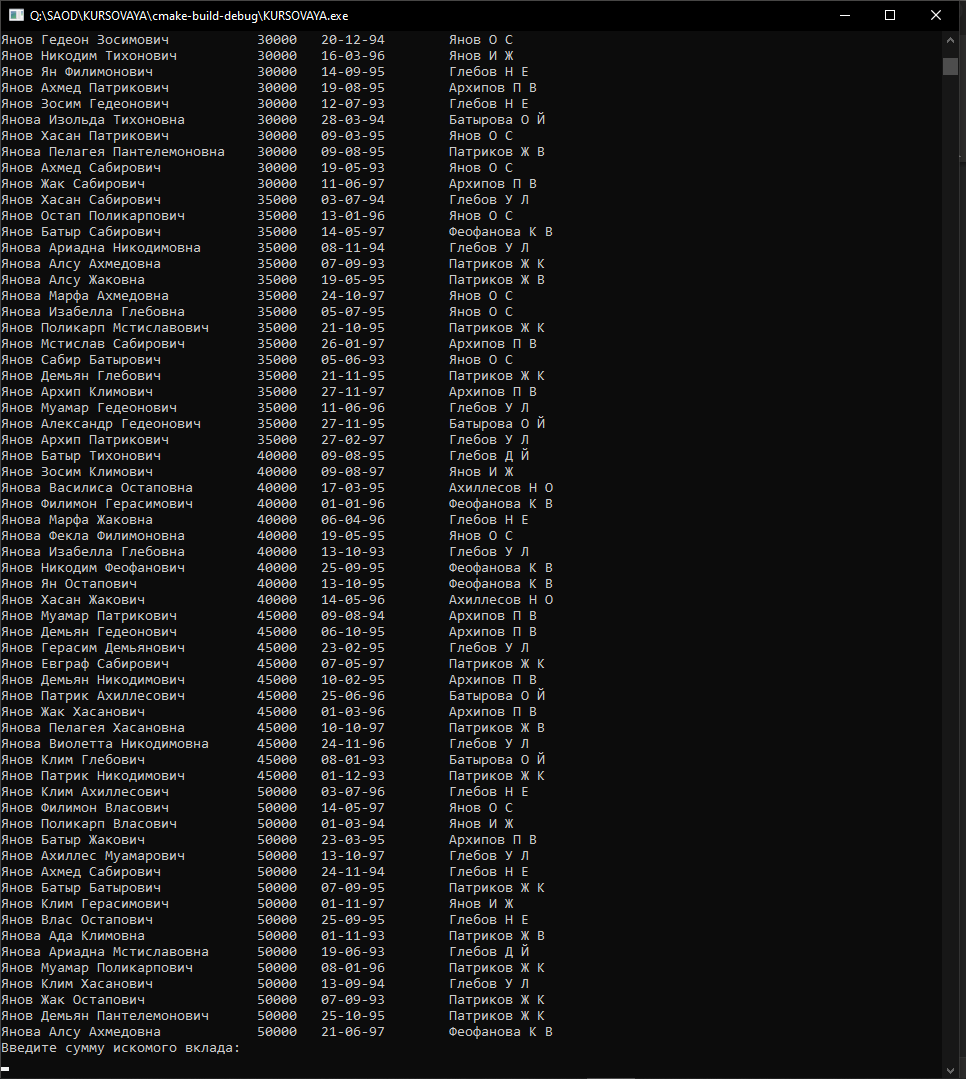
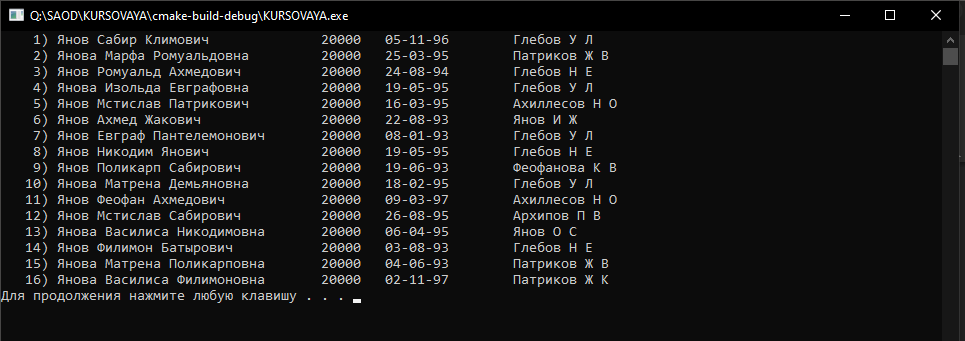
 

Рисунок 8, 9. Вывод дерева и поиск по сумме вклада

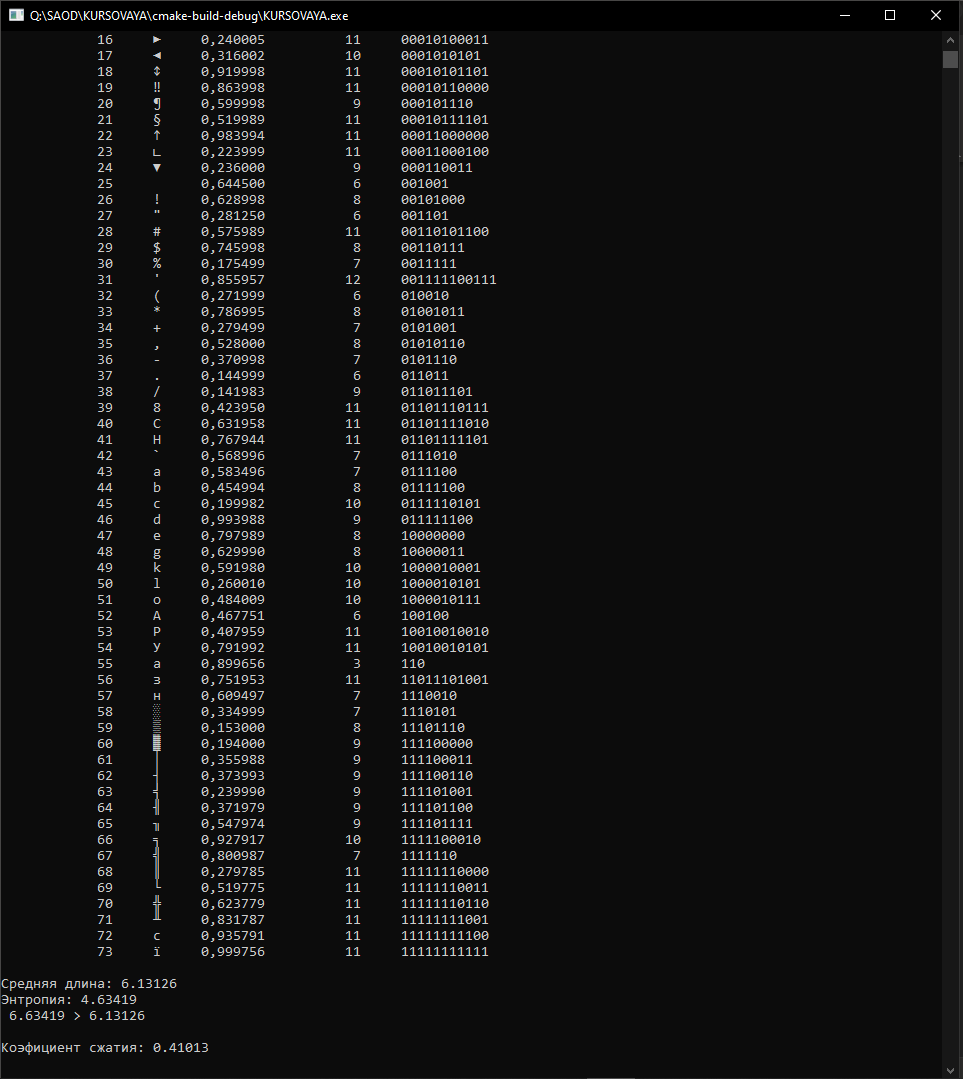


Рисунок 10. Статистика по кодированию

Выводы

В ходе выполнения курсовой работы были выполнены все поставленные задачи и реализованы необходимые алгоритмы: сортировки, поиска, построения дерева оптимального поиска, поиска по дереву и кодирование базы данных.

В результате кодирования были получены данные подтверждающие теоретические сведения. К таковым относятся: величины средней длины кодового слова и энтропии (Lср ≤ H + 2) и установлено, что при полученном значении Lср и H размер кодируемой информации увеличивается примерно в 2.5 раза.

Реализованные алгоритмы представляют минимальный набор процедур для представления и обработки базы данных, а также отличаются достаточно высоким быстродействием и эффективностью.

*Текст программы*

#include <fstream>

#include <time.h>

#include <cmath>

#include <iostream>

#include <conio.h>

#include <Windows.h>

#include <iomanip>

#include <cstdio>

#include <cstring>

#include <cstdlib>

using namespace std;

const int N = 4000;

const int M = 256;

int \*w;

int num\_h = 0, num\_t = 0;

int n = 0;

int sum = 0;

int code[M][M];

float entropy = (float) (0);

float midlength = (float) (0);

int sim\_cf = 0, sim\_f = 0;

struct record {

char depositor[30];

unsigned short int deposit;

char date[10];

char lawyer[22];

};

struct list {

record \*data;

list \*next;

list \*prior;

};

struct queue {

list \*head;

list \*tail;

};

struct vertex {

list \*fact;

vertex \*left;

vertex \*right;

};

list \*base, \*sbase;

queue \*stack;

queue Q;

struct GM\_code {

unsigned char a;

float p;

float q;

int l;

};

GM\_code A[M];

GM\_code B[M];

void read\_base(FILE \*, list \*);

void print\_record(record \*);

void print\_base(list \*);

void print\_stack(queue \*);

void print\_tree(vertex \*&);

void copy\_base(list \*, list \*);

void prior(list \*);

void DigitalSort(list \*&, bool);

void toArray(list \*l, list \*r[4000]);

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

void binSearch(char \*x, list \*l[4000], queue \*&turn);

void add(vertex \*&, record \*x);

void DOP\_A2(vertex \*&, list \*[N], int, int);

void QuickSort(int \*, list \*[N], int, int);

void weight(list \*mas[N]);

void tree\_search(vertex \*&, unsigned short int);

void delete\_tree(vertex \*&);

void reading();

void code\_file();

void printGMcode();

int main() {

setlocale(LC\_ALL, "Russian");

SetConsoleCP(866);

FILE \*Base1;

Base1 = fopen("testBase3.dat", "rb");

base = new list();

read\_base(Base1, base);

\_fcloseall();

///sort///

sbase = new list();

copy\_base(base, sbase);

DigitalSort(sbase, 0);

list \*mas\_sbase[N] = {};

///tree///

vertex \*root = NULL;

///code//

SetConsoleCP(1251);

reading();

code\_file();

///-menu-///

int enter = 0;

while (true) {

system("CLS");

enter = 0;

SetConsoleCP(866);

cout << "\t1. Просмотр базы данных (< - назад, esc - выйти)" << endl;

cout << "\t2. Просмотр отсортированной базы данных (Digital sort)" << endl;

cout << "\t3. Поиск в отсортированной базе по ключу (вкладчик)" << endl;

cout << "\t4. Поиск в дереве (вклад)" << endl;

cout << "\t5. Статистика по кодированию" << endl;

cout << "\tEsc. Выход " << endl;

while ((enter != 27) && (enter != 49) && (enter != 50) && (enter != 51) && (enter != 52) && (enter != 53)) {

if (\_kbhit()) {

enter = \_getch();

}

}

switch (enter) {

case 27:

return 0;

case 49:

print\_base(base);

break;

case 50:

print\_base(sbase);

break;

case 51:

stack = NULL;

root = NULL;

char X[3];

SetConsoleCP(866);

cout << "Введите ключ поиска (три буквы искомой публикации): ";

cin >> X;

toArray(sbase, mas\_sbase);

binSearch(X, mas\_sbase, stack);

if (stack->head != stack->tail)

print\_stack(stack);

else

cout << "Такой записи нет";

\_getch();

break;

case 52:

if (mas\_sbase[0] == NULL) break;

if (root != NULL) {

delete\_tree(root);

root = NULL;

}

unsigned short int x;

weight(mas\_sbase);

DOP\_A2(root, mas\_sbase, num\_h, num\_t);

cout << "Посмотреть дерево (1 - да, другая клавиша - нет) ";

cin >> x;

if (x == 1) {

print\_tree(root);

}

SetConsoleCP(866);

cout << "Введите сумму искомого вклада: " << endl;

cin >> x;

tree\_search(root, x);

break;

case 53:

printGMcode();

break;

}

}

return 0;

}

void read\_base(FILE \*f, list \*l) {

record \*tmp = new record();

list \*tmpl;

fread((record \*) tmp, sizeof(record), 1, f);

l->data = tmp;

l->prior = NULL;

l->next = NULL;

for (int i = 1; i < N; i++) {

tmp = new record();

fread((record \*) tmp, sizeof(record), 1, f);

tmpl = l;

l = l->next = new list();

l->data = tmp;

l->prior = tmpl;

l->next = NULL;

}

}

void print\_record(record \*t) {

SetConsoleCP(1251);///Меняем кодировку

cout << t->depositor;

cout << "\t";

cout << t->deposit;

cout << "\t";

cout << t->date;

cout << "\t";

cout << t->lawyer;

cout << "\t" << endl;

}

void print\_base(list \*l) {

int M = 20; ///По столько элементов выводится на экран

int i, flag = 1, num = 1, broke = 0;

int c = 0;

cout << "1 - вывод по 20 (управление 1 - предыдущая страница, 2 - следующая страница)" << endl

<< "2 - вывод всего списка" << endl;

while ((c != 49) && (c != 27) && (c != 50)) {

if (\_kbhit()) {///Пользоавтель нажал на 1 / 2 / ESc

c = \_getch();

}

}

if (c == 27) { ///Пользоавтель нажал на кнопку = ESC

return;

} else {

if (c == 50) {

M = N;

broke = 1;

}

}

while (flag) {

i = 0;

system("cls");

while ((i < M) && (num <= N) && (l != NULL)) {

cout << " ";

cout << setw(4) << num << ") ";

print\_record(l->data);

num++;

i++;

if (l->next == NULL)

break;

else

l = l->next;

}

SetConsoleCP(866);///Возвращаем кодировку

if (broke == 0) {

c = 0;

while ((c != 49) && (c != 27) && (c != 50)) {

if (\_kbhit()) {///Пользоавтель нажал на 1 / 2 / ESc

c = \_getch();

}

}

if (c == 49) {///Пользоавтель нажал на кнопку = 1

for (int j = 0; j < i + M; j++) {

if (l->prior == NULL) break;

l = l->prior;

num--;

}

} else if (c == 27) { ///Пользоавтель нажал на кнопку = ESC

flag = 0;

return;

}

} else {

flag = 0;

system("pause");

}

}

}

void print\_stack(queue \*p) {

list \*l = p->head;

int M = N; ///По столько элементов выводится на экран

int i, flag = 1, num = 1;

int c = 0;

i = 0;

system("cls");

while ((i < M) && (p->tail != l)) {

cout << " ";

cout << setw(4) << num << ") ";

print\_record(l->data);

num++;

i++;

if ((l->next == NULL) || (p->tail == l->next)) {

num--;

break;

} else

l = l->next;

}

SetConsoleCP(866);///Возвращаем кодировку

}

void print\_tree(vertex \*&p) {

if (p == NULL) return;

list \*tmp = p->fact;

print\_tree(p->left);

tmp = p->fact;

while (tmp != NULL) {

print\_record(tmp->data);

tmp = tmp->next;

}

print\_tree(p->right);

}

void copy\_base(list \*a, list \*b) {

b->prior = NULL;

b->data = a->data;

for (int i = 1; i < N; i++) {

a = a->next;

b->next = new list;

b->next->prior = b;

b = b->next;

b->data = a->data;

}

b->next = NULL;

}

void prior(list \*p) {

p->prior = NULL;

for (int i = 1; i < N; i++) {

p->next->prior = p;

p = p->next;

}

}

void DigitalSort(list \*&head, bool flag) {

queue Q[257];

for (int i = 0; i < 257; i++) {

Q[i].tail = (list \*) &(Q[i].head);

}

list \*p;

SetConsoleCP(1251);

for (int j = 30; j >= 0; j--) {

for (int i = 0; i < 257; i++) {

Q[i].tail = Q[i].head = NULL;

}

while (head) {

int d;

if (flag == 0) {

if (head->data->deposit == 0)

d = 0;

else

d = head->data->deposit / 1000;

} else {

if (head->data->depositor[j] == ' ')

d = 0;

else

d = head->data->depositor[j] + 129;

}

p = Q[d].tail;

if (Q[d].head == NULL)

Q[d].head = head;

else

p->next = head;

p = Q[d].tail = head;

head = head->next;

p->next = NULL;

}

cout << endl << endl;

head = NULL;

int i;

for (i = 0; i < 256; i++) {

if (Q[i].head != NULL)

break;

}

head = Q[i].head;

p = Q[i].tail;

for (int k = i + 1; k < 256; k++) {

if (Q[k].head != NULL) {

p->next = Q[k].head;

p = Q[k].tail;

}

}

}

if (flag == 0)

DigitalSort(head, 1);

else

prior(head);

}

void toArray(list \*l, list \*\*r) {

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

r[i] = l;

l = l->next;

}

}

int compare(char \*a, char \*b) {

for (int i = 0; i < 3; i++) {

if ((int) a[i] < (int) b[i]) return 1;

else if ((int) a[i] > (int) b[i]) return -1;

}

return 0;

}

void binSearch(char \*x, list \*\*l, queue \*&turn) {

int L = 1;

int R = N;

int m, h, t;

int check;

turn = new queue();

turn->head = turn->tail;

while (L < R) {

m = (L + R) / 2;

check = compare(l[m]->data->depositor, x);

if (check == 1) {

L = m + 1;

} else {

if (check == -1) {

R = m;

} else if (check == 0) {

h = m;

t = m;

while (compare(l[h - 1]->data->depositor, x) == 0) {

h--;

if (h - 1 == 0)

break;

}

while (compare(l[t + 1]->data->depositor, x) == 0) {

t++;

if (t + 1 == 4000)

break;

}

turn->head = l[h];

num\_h = h;

turn->tail = l[t];

num\_t = t;

return;

}

}

}

}

void add(vertex \*&point, record \*x) {

vertex \*\*p = &point;

list \*tmp;

while (\*p) {

if ((\*x).deposit < (\*p)->fact->data->deposit) {

p = &((\*p)->left);

} else if ((\*x).deposit > (\*p)->fact->data->deposit) {

p = &((\*p)->right);

} else if ((\*x).deposit == (\*p)->fact->data->deposit) {

tmp = (\*p)->fact;

while (tmp->next != NULL) {

tmp = tmp->next;

}

tmp->next = new list();

tmp->next->data = x;

tmp->next->prior = (\*p)->fact;

tmp->next->next = NULL;

return;

}

}

if ((\*p) == NULL) {

(\*p) = new vertex();

(\*p)->fact = new list();

(\*p)->fact->data = x;

(\*p)->fact->next = NULL;

(\*p)->fact->prior = NULL;

(\*p)->left = NULL;

(\*p)->right = NULL;

}

}

void DOP\_A2(vertex \*&q, list \*mas[N], int L, int R) {

int wes = 0, summa = 0;

int i;

L = L - num\_h;

R = R - num\_h;

if (L <= R) {

for (i = L; i <= R; i++)

wes = wes + w[i];

for (i = L; i <= R; i++) {

if ((summa <= (wes / 2)) && ((summa + w[i]) >= (wes / 2)))

break;

summa = summa + w[i];

}

add(q, mas[i + num\_h]->data);

DOP\_A2(q, mas, L + num\_h, i - 1 + num\_h);

DOP\_A2(q, mas, i + 1 + num\_h, R + num\_h);

}

}

void weight(list \*mas[N]) {

w = new int[num\_t - num\_h + 1];

for (int i = 0; i < (num\_t - num\_h + 1); i++) {

w[i] = rand() % 999 + 1;

}

QuickSort(w, mas, 0, num\_t - num\_h);

}

void QuickSort(int \*a, list \*mas[N], int left, int right) {

int current, i, j, t;

list \*tmp;

current = mas[left + num\_h]->data->deposit;

i = left;

j = right;

while (i <= j) {

while (mas[i + num\_h]->data->deposit < current) {

i++;

}

while (mas[j + num\_h]->data->deposit > current) {

j--;

}

if (i <= j) {

t = a[i];

a[i] = a[j];

a[j] = t;

tmp = mas[i + num\_h];

mas[i + num\_h] = mas[j + num\_h];

mas[j + num\_h] = tmp;

i++;

j--;

}

}

if (left < j) {

QuickSort(a, mas, left, j);

}

if (i < right) {

QuickSort(a, mas, i, right);

}

}

void tree\_search(vertex \*&p, unsigned short int x) {

vertex \*tmp = p;

while (tmp != NULL) {

if (tmp->fact->data->deposit == x) {

print\_base(tmp->fact);

return;

} else if (tmp->fact->data->deposit > x) {

tmp = tmp->left;

} else if (tmp->fact->data->deposit < x) {

tmp = tmp->right;

}

}

cout << "Вклад не найден" << endl;

}

void delete\_tree(vertex \*&p) {

if (p != NULL) {

delete\_tree(p->left);

delete\_tree(p->right);

delete p;

}

}

void reading() {

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

A[i].a = i;

A[i].p = 0;

A[i].q = (float) (0);

A[i].l = 0;

}

FILE \*f;

f = fopen("testBase3.dat", "rb");

while (!feof(f)) {

char c;

fscanf(f, "%c", &c);

if (feof(f))

break;

A[c + 128].p += 1;

sum++;

}

fclose(f);

for (int i = 0, j = 0; i < M; i++) {

if (A[i].p != 0) {

n++;

A[i].p /= sum;

B[j] = A[i];

entropy += A[i].p \* abs(log(A[i].p) / log(2));

j++;

}

}

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

B[i].q = B[i - 1].q + B[i].p / 2;

B[i].l = ceil(-log(B[i].p) / log(2)) + 1;

}

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

for (int j = 0; j <= B[i].l; j++) {

B[i].q \*= float(2);

code[i][j] = floor(B[i].q);

while (B[i].q >= 1)

B[i].q -= 1;

}

}

}

void code\_file() {

FILE \*f, \*cf;

f = fopen("testBase3.dat", "rb");

cf = fopen("CodeBase.dat", "wb");

char sim;

while (!feof(f)) {

fscanf(f, "%c", &sim);

sim\_f++;

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

if (sim == B[i].a) {

for (int j = 0; j < B[i].l; j++) {

putc(code[i][j], cf);

sim\_cf++;

}

}

}

}

\_fcloseall();

}

void printGMcode() {

int j;

int i;

float check = 0;

midlength = 0;

for (i = 0; i < n; i++) {

midlength += B[i].l \* B[i].p;

}

SetConsoleCP(1251);

for (i = 0; i < n; i++) {

printf(" %12d %c %2.6f %7d ", i, (char) B[i].a, B[i].q, B[i].l);

for (j = 1; j <= B[i].l; ++j)

cout << code[i][j];

cout << endl;

}

SetConsoleCP(866);

cout << endl;

cout << setw(8) << "Средняя длина: " << midlength << endl;

cout << setw(8) << "Энтропия: " << entropy << endl;

cout << setw(8) << entropy + 2 << " > " << midlength << endl << endl;

cout << setw(8) << "Коэфициент сжатия: " << (float) sim\_f / sim\_cf << endl;

\_getch();

}