Федеральное агентство связи

Сибирский государственный университет телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

Курсовой проект

«Структуры и алгоритмы обработки данных»

Вариант

Выполнил: студент группы

Проверил: ассистент кафедры ПМиК

Новосибирск 2021

Оглавление

[1. Постановка задачи 2](#_Toc533614154)

[2. Основные идеи и характеристики применяемых методов 3](#_Toc533614155)

[2.1. Метод сортировки 3](#_Toc533614156)

[2.2. Двоичный поиск 3](#_Toc533614157)

[2.3. Дерево и поиск по нему 4](#_Toc533614158)

[2.4. Метод кодирования 6](#_Toc533614159)

[3. Особенности реализации алгоритмов 7](#_Toc533614160)

[4. Описание программы 9](#_Toc533614161)

[4.1. Основные переменные и структуры 9](#_Toc533614162)

[4.2. Описание подпрограмм 10](#_Toc533614163)

[5. Текст программы 12](#_Toc533614164)

[6. Результаты 12](#_Toc533614165)

[7. Выводы 33](#_Toc533614166)

# Постановка задачи

Хранящуюся в файле библиогpафическую базу данных "Пpедпpиятие " загрузить динамически в оперативную память компьютера в виде списка, вывести на экран по 20 записей на странице с возможностью отказа от просмотра.

Упорядочить данные с помощью цифровой сортировки (Digital sort), построить по отсортированным данным индексный массив. 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 оптимального поиска по ключу (приближенный алгоритм А2) по должности, вывести на экран содержимое дерева и предусмотреть возможность поиска в дереве по запросу.

Закодировать файл базы данных кодом Гилберта-Мура, 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.

База данных "Пpедпpиятие"

Стpуктуpа записи:

ФИО сотpудника: текстовое поле 30 символа

фоpмат <Фамилия>\_<Имя>\_<Отчество>

Hомеp отдела: целое число

Должность: текстовое поле 22 символа

Дата pождения: текстовое поле 10 символов

фоpмат дд-мм-гг

Пpимеp записи из БД:

Петpов\_Иван\_Иванович\_\_\_\_\_\_\_\_\_\_\_\_

130

начальник\_отдела\_\_\_\_\_\_

15-03-46

Ваpианты условий упоpядочения и ключи поиска (К):

по номеру отдела и ФИО, К = номеp отдела. Ключ в дереве = дата рождения

# Основные идеи и характеристики применяемых методов

## Метод сортировки

Пирамидальная сортировка

Пирамидальная сортировка основана на алгоритме построения пирамиды. Последовательность ai, ai+1,…,ak называется (i,k)-пирамидой, если неравенство

aj≤min(a2j, а2j+1) (\*)

выполняется для каждого j, j=i,…,k для которого хотя бы один из элементов a2j, a2j+1 существует.

Например, массив А является пирамидой, а массив В ⎯ не является.

А=(а2 , а3 , а4 , а5 , а6 а7 , а8 )=(3, 2, 6, 4, 5, 7)

В=(b1, b2, b3, b4, b5, b6, b7)=(3, 2, 6, 4, 5, 7)

Свойства пирамиды

Если последовательность ai, ai+1,…,аk-1, ak является (i, k)-пирамидой, то последовательность ai+1,…,ak-1, полученная усечением элементов с обоих концов последовательности, является (i+1, k-1)пирамидой.

Если последовательность a1…an – (1, n)-пирамида, то а1 – минимальный элемент последовательности.

Если a1, a2…,an/2,an/2+1,…an-произвольная последовательность, то последовательность an/2+1,…,an является (n/2+1, n)-пирамидой.

Процесс построения пирамиды выглядит следующим образом. Дана последовательность as+1,…,ak, которая является (s+1, k)-пирамидой. Добавим новый элемент х и поставим его на s-тую позицию в последовательности, т.е. пирамида всегда будет расширяться влево. Если выполняется (\*), то полученная последовательность – (s, k)-пирамида. Иначе найдутся элементы a2s+1,a2s такие, что либо a2s < as либо a2s+1 < as.

Пусть имеет место первый случай, второй случай рассматривается аналогично. Поменяем местами элементы as и a2s. В результате получим новую последовательность as’,as+1,…, a2s’,…, ak. Повторяем все действия для элемента a2s’ и т.д. пока не получим (s, k)-пирамиду.

## Двоичный поиск

Алгоритм двоичного поиска в упорядоченном массиве сводится к следующему. Берём средний элемент отсортированного массива и сравниваем с ключом X. Возможны три варианта:

Выбранный элемент равен X. Поиск завершён.

Выбранный элемент меньше X. Продолжаем поиск в правой половине массива.

Выбранный элемент больше X. Продолжаем поиск в левой половине массива.

Из-за необходимости найти все элементы, соответствующие заданному ключу поиска в курсовой работе использовалась вторая версия двоичного поиска, которая из необходимых элементов находит самый левый, в результате чего для поиска остальных требуется просматривать лишь оставшуюся правую часть массива.

Верхняя оценка трудоёмкости алгоритма двоичного поиска такова. На каждой итерации поиска необходимо два сравнение для первой версии, одно сравнение для второй версии. Количество итераций не больше, чем f_02. Таким образом, трудоёмкость двоичного поиска в обоих случаях

f_03

## Дерево и поиск по нему

АВЛ-дерево

Как было показано выше, ИСДП обеспечивает минимальное среднее время поиска. Однако перестройка дерева после случайного включения вершины – довольно сложная операция. СДП дает среднее время поиска на 40 % больше, но процедура построения достаточно проста. Возможное промежуточное решение – введение менее строгого определения сбалансированности. Одно из таких определений было предложено Г. М. Адельсон – Вельским и Е. М. Ландисом (1962).

Дерево поиска называется сбалансированным по высоте, или АВЛ – деревом, если для каждой его вершины высоты левого и правого поддеревьев отличаются не более чем на 1.

На рисунке 39 приведены примеры деревьев, одно из которых является АВЛ-деревом, а другое – нет. В выделенной вершине нарушается баланс высот левого и правого поддеревьев.

Рисунок 39 Пример АВЛ-дерева и не АВЛ-дерева

Заметим, что ИСДП является также и АВЛ – деревом. Обратное утверждение не верно.

Адельсон – Вельский и Ландис доказали теорему, гарантирующую, что АВЛ-дерево никогда не будет в среднем по высоте превышать ИСДП более, чем на 45% независимо от количества вершин:

log(n+1) ≤ hАВЛ(n) < 1,44 log(n+2) – 0,328 при n→∞.

Таким образом, лучший случай сбалансированного по высоте дерева – ИСДП, худший случай – плохое АВЛ – дерево. Плохое АВЛ – дерево это АВЛ-дерево, которое имеет наименьшее число вершин при фиксированной высоте. Рассмотрим процесс построения плохого АВЛ-дерева. Возьмём фиксированную высоту h и построим АВЛ – дерево с минимальным количеством вершин. Обозначим такое дерево через Th. Ясно, что Т0 – пустое дерево, Т1 – дерево с одной вершиной. Для построения Тh при h > 1 будем брать корень и два поддерева с минимальным количеством вершин.

h=1

**T1**

h=2

**T2**

h=3

**T3**

h=4

**T4**

T3

T2

**T5**

T3

T4

h=5

Рисунок 40 Деревья Фибоначчи

Одно поддерево должно быть высотой h–1, а другое высотой h–2. Поскольку принцип их построения очень напоминает построение чисел Фибоначчи, то такие деревья называют деревьями Фибоначчи: Th = < Th-1, x, Th-2 >. Число вершин в Th определяется следующим образом:

n0 = 0, n1 = 1, nh = nh-1 + 1 + nh-2

Повороты при балансировке

Рассмотрим, что может произойти при включении новой вершины в сбалансированное по высоте дерево. Пусть r – корень АВЛ-дерева, у которого имеется левое поддерево (ТL) и правое поддерево (TR). Если добавление новой вершины в левое поддерево приведет к увеличению его высоты на 1, то возможны три случая:

если hL = hR, то ТL и TR станут разной высоты, но баланс не будет нарушен;

если hL < hR, то ТL и TR станут равной высоты, т. е. баланс даже улучшится;

если hL > hR, то баланс нарушиться и дерево необходимо перестраивать.

Введём в каждую вершину дополнительный параметр Balance (показатель баланса), принимающий следующие значения:

-1, если левое поддерево на единицу выше правого;

0, если высоты обоих поддеревьев одинаковы;

1, если правое поддерево на единицу выше левого.

Если в какой-либо вершине баланс высот нарушается, то необходимо так перестроить имеющееся дерево, чтобы восстановить баланс в каждой вершине. Для восстановления баланса будем использовать процедуры поворотов АВЛ-дерева.

## Метод кодирования

Оптимальный код Хаффмана

Е. Н. Гилбертом и Э. Ф. Муром был предложен метод построения Метод оптимального побуквенного кодирования был разработан в 1952 г. Д. Хаффманом. Оптимальный код Хаффмана обладает минимальной средней длиной кодового слова среди всех побуквенных кодов для данного источника с алфавитом А={a1,…,an} и вероятностями pi =P(ai).

Рассмотрим алгоритм построения оптимального кода Хаффмана, который основывается на утверждениях лемм предыдущего параграфа.

Упорядочим символы исходного алфавита А={a1,…,an} по убыванию их вероятностей p1≥p2≥…≥pn.

Если А={a1,a2}, то a1→0, a2→1.

Если А={a1,…,aj,…,an} и известны коды <aj → bj >, j = 1,…,n, то для алфавита {a1,…aj/, aj//…,an} с новыми символами aj/ и aj// вместо aj, и вероятностями p(aj)=p(aj/)+ p(aj//), код символа aj заменяется на коды aj/ → bj0, aj// →bj1.

Код Хаффмана обычно строится и хранится в виде двоичного дерева, в листьях которого находятся символы алфавита, а на «ветвях» – 0 или 1. Тогда уникальным кодом символа является путь от корня дерева к этому символу, по которому все 0 и 1 собираются в одну уникальную последовательность

# 3. Особенности реализации алгоритмов

В ходе выполнения курсовой работы, помимо основных алгоритмов, потребовалось реализовать также несколько вспомогательных, необходимых для корректной работы программы.

1. *Интерфейс программы*

Интерфейс программы реализован в бесконечном цикле с возможностью выхода из него, при нажатии клавиши «Esc». Выбор пункта меню реализован с помощью функции switch( ).

1. *Загрузка и вывод базы данных*

База данных открывается в int main( ). Считывается база данных в список структур в функции void Read\_base(FILE \*fp, list \*base), где struct base – список, в котором хранится база данных. Считывание производится независимо от желания пользователя, в то время как большинство остальных функций он может выбрать посредствам меню. После считывания в список структур, файл закрывается.

За вывод элементов базы данных отвечает процедура void Print(list \*base) которая представляет возможность просмотра базы данных постранично.

1. *Вспомогательные функции и процедуры для сортировки данных*

База данных сортируется после запуска программы. Для сортировки базы данных используется процедура void HeapSort (int n, list \*\*A). Она сортирует список сначала по полю «номер отдела», а потом для всех каждой группы записей с одинаковым полем «номер отдела» дополнительно сортирует по полю «ФИО». Для быстрого доступа к отсортированной и неотсортированной базе данных, перед вызовом процедуры сортировки, делается копия списка, хранящего элементы базы данных, с помощью процедуры void copy\_base(list \*a, list \*b).

1. *Особенности реализации бинарного поиска*

Бинарный поиск по отсортированной базе данных осуществляется в процедуре void BSearch(list \*\*A,char year[2]). Результатом работы процедуры void BSearch(list \*\*A,char year[2]) является индекс найденного элемента в индексном массиве, удовлетворяющий условию поиска. При отсутствии элементов с заданным ключом, программа выводит сообщение о том, что таких элементов нет.

1. *Вспомогательные функции и процедуры для построения дерева оптимального поиска (приближенный алгоритм А2).*

Для построения дерева используется рекурсивная функция void addAVL. В случае возникновения нарушений в балансе при построении дерева вызваются функции void ll, void lr, void rr, void rl в зависимости от ситуации. Для вывода дерева на экран используется процедура void LRprint, представляющая собой обход дерева слева – направо.

Аналогичная процедура void TreeSearch выполняет вывод результатов поиска в дереве.

1. *Кодирование данных*

Кодирование данных начинается с процедуры void BaseCoding, которая открывает файл базы данных для чтения, заполняет массив структур для алфавита кодовых слов всеми возможными символами, считывает из файла символы и считает их вероятности, закрывает файл, считает энтропию, а затем длину кодового слова и само кодовое слово. В процедуре void CodeBase создается бинарный файл, в который записывается закодированная база данных и считается ее размер и размер изначальной базы данных. В процедуре void CodePrint осуществляется подсчет и вывод средней длины кодового слова, и еще выводит энтропию, символы, их вероятности, длины кодовых слов и сами кодовые слова на монитор, а также подсчитывает и выводит коэффициент сжатия.

# Описание программы

## Основные переменные и структуры

struct record

{

char FIO[30];

short int Department;

char Position[22];

char Date[10];

}

Структура для хранения элемента базы данных. Всего таких элементов:

const int N = 4000;

struct list {

record \*data;

list \*next;

list \*prev;

}

Структура для создания списка, в котором хранятся элементы базы данных.

struct vertex

{

record \*data;

vertex \*next;

int bal;

vertex \*left;

vertex \*right;

}

Структура, представляющая АВЛ-дерево

struct HF\_code {

float p;

float q;

char a;

int l;

}

Структура, представляющая собой информацию о символе для формирования кодового слова.

HF\_code A[M] - массив для всех символов

const int M = 256 – число символов в алфавите

float entropy = 0 - энтропия

float lgm = 0 – средняя длина кодового слова

int sum =0 – счётчик всех символов в файле

## Описание подпрограмм

Процедуры начальной обработки базы данных:

1. void Read\_base(FILE \*fp, list \*base) – считывает базу данных и создает список.
2. Void Print \_base (list \*base); - визуальный вывод списка на консоль.

Процедуры сортировки:

1. void copy\_base(list \*a, list \*b) – делает копию списка, в котором хранится база данных.
2. void HeapSort (int n, list \*\*A) – сортирует базу данных по номеру отдела и ФИО сотрудника.
3. void HeapBuild (list \*\*A, int l, int r) – вспомогательная процедура для сортировки по номеру отдела
4. void HeapBuild2 (list \*\*A, int l, int r, int offset) – вспомогательная процедура для второстепенной сортировки по ФИО строк, сгруппированных на основе одинаковых номеров отделов.

Процедуры и функции для поиска в отсортированной базе данных:

1. void BSearch(list \*\*A, char year[2]) – бинарный поиск по ключу года.

Процедуры построения дерева оптимального поиска (А2):

1. void addAVL(list \*mas, vertex\*&) – рекурсивное добавление элемента в дерево
2. void ll(vertex\*&) – вспомогательный метод перестройки дерева.
3. void rr(vertex\*&) – вспомогательный метод перестройки дерева.
4. void lr(vertex\*&) – вспомогательный метод перестройки дерева.
5. void rl(vertex\*&) – вспомогательный метод перестройки дерева.
6. void LRprint(vertex\*) – вывод дерева.
7. void TreeSearch (vertex\* p, int data) – поиск в дереве.

Процедуры и функции кодирования базы данных:

1. void BaseCoding () – считывание символов базы данных, подсчет их вероятностей.
2. void Huffman(int n) - преобразование алфавита, создание кодовых слов по методу Хаффмана.
3. int Up(int n, float q) – вспомогательный метод для создания кодовых слов
4. void Down(int n, int j) – вспомогательный метод для создания кодовых слов
5. void CodeBase() – кодирование базы данных.
6. void CodePrint() - вывод статистики и алфавита с вероятностями.

Основная программа:

main( ) - основная программа, в которой выводится меню, а также в зависимости от выбранного пункта меню вызываются соответствующие процедуры и функции.

# Текст программы

# 

# #include <fstream>

# #include <iostream>

# #include <conio.h>

# #include <Windows.h>

# #include <iomanip>

# #include <cstdio>

# #include <cstring>

# #include <cstdlib>

# #include <cmath>

# #include <iomanip>

# using namespace std;

# const int N = 4000;

# const int M = 256;

# int sum = 0;

# int sym\_count = 0;

# int code[M][M];

# float entropy = 0, lgm = 0;

# int fcompression = 0, cfcompression = 0;

# struct record

# {

# char FIO[30];

# short int Department;

# char Position[22];

# char Date[10];

# };

# struct list

# {

# record \*data;

# list\* prev;

# list\* next;

# };

# struct vertex

# {

# record \*data;

# vertex \*next;

# int bal;

# vertex \*left;

# vertex \*right;

# };

# vertex \*root = NULL;

# bool up = false;

# struct HF\_code {

# float p;

# float q;

# char a;

# int l;

# };

# HF\_code A[M];

# void Read\_base(FILE \*fp, list \*base);

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

# void Print\_base(list \*base);

# int compare\_deposit(int a, int b);

# int compare\_name(char a[], char b[]);

# int compare\_date(char a[], char b[]);

# int BSearch (list \*\*A, int Number);

# void HeapBuild (list \*\*A, int l, int r);

# void HeapBuild2 (list \*\*A, int l, int r, int offset);

# void HeapSort (int n, list \*\*A);

# void Print\_sort\_index\_mas(list \*\*mas);

# void LRprint(vertex\*);

# void AVLSearch(vertex\* p, char\* data);

# void AVLNext(vertex\* p);

# void addAVL(list \*mas, vertex\*&);

# void ll(vertex\*&);

# void rr(vertex\*&);

# void lr(vertex\*&);

# void rl(vertex\*&);

# int size(vertex\*);

# int height(vertex\*);

# int max(int, int);

# int sdp(vertex\*, int);

# int Up(int n, float q);

# void Down(int n, int j);

# void Huffman(int n);

# void BaseCoding();

# void CodePrint();

# void CodeBase();

# int main()

# {

# setlocale(LC\_ALL, "Russian");

# FILE \*f;

# 

# f=fopen("testBase2.dat", "rb");

# list \*base = new list;

# list \*base\_index =new list;

# Read\_base(f,base);

# fclose(f);

# copy\_base(base, base\_index);

# list \*mas[N];

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

# mas[i]=base\_index;

# base\_index=base\_index->next;

# }

# HeapSort(N, mas);

# BaseCoding();

# CodeBase();

# 

# int enter = 0;

# while (true) {

# system("CLS");

# enter = 0;

# int search\_start=0;

# int search=N-1;

# int j=0;

# int dep\_num = 0;

# 

# SetConsoleCP(866);

# cout << "\t------------------------------МЕНЮ------------------------" << endl;

# cout << "\t1. Посмотреть базу данных" << endl;

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

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

# cout << "\t4. Поиск в дереве (Дата рождения)" << endl;

# cout << "\t5. Кодирование" << endl;

# cout << "\tEsc. Выход " << endl;

# cout << "\t----------------------------------------------------------" << 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\_sort\_index\_mas(mas);

# break;

# case 51:

# cout <<endl;

# SetConsoleCP(866);

# cout << " Введите номер отдела для поиска"<<endl;

# 

# cin >> dep\_num;

# if(dep\_num>=0){

# search = BSearch(mas, dep\_num);

# if(search == -1){

# cout << " Сотрудника из такого отдела не существует" << endl;

# }

# else{

# SetConsoleCP(1251);

# do{

# if(search==0){

# break;

# }

# else{

# search--;

# }

# if(mas[search]->data->Department != dep\_num){

# search++;

# break;

# }

# }

# while(true);

# 

# search\_start=search;

# 

# do{

# search++;

# if(search==N){

# search--;

# break;

# }

# if(mas[search]->data->Department != dep\_num){

# break;

# }

# }

# while(true);

# SetConsoleCP(866);

# cout <<endl <<endl<<" Найдено"<<search-search\_start <<" сотрудников ("<< search\_start<< " "<< search-1<<")"<<endl;

# SetConsoleCP(1251);

# for(int i=search\_start;i<search;i++){

# cout << i <<" "<<mas[i]->data->Date <<"\t"<<mas[i]->data->FIO <<"\t"<<mas[i]->data->Department <<"\t"<<mas[i]->data->Position <<endl;

# addAVL(mas[i], root);

# }

# }

# }

# break;

# case 52:

# cout << endl << "LR-print - department"<<endl;

# LRprint(root);

# cout<<endl;

# printf("+------+----------+----------------+----------+----------------+\n");

# printf("|%6d| Size | Height | Mid. height |\n", N);

# printf("+------+----------+----------------+----------+----------------+\n");

# printf("| AVL |%10d|%10d|%16.2f|\n", size(root), height(root), (double)sdp(root, 1) / size(root));

# cout<<endl<< "Element find: "<< endl;

# 

# char temp\_date[255];

# cin >> temp\_date;

# char date[10];

# strncpy(date, temp\_date, 10);

# 

# AVLSearch(root, date);

# break;

# case 53:

# CodePrint();

# break;

# }

# \_getch();

# }

# system ("Pause");

# return 0;

# }

# void CodePrint(){

# lgm=0;

# SetConsoleCP(866);

# printf("\n\nОптимальный код Хаффмана: \n");

# printf("-------------------------------------------------------------------------------\n");

# printf("| Номер Символа | Символ | Вероятность | Кодовое слово | Длина кодового|\n");

# printf("| | | | | слова |\n");

# printf("|-----------------------------------------------------------------------------|\n");

# SetConsoleCP(1251);

# for (int i = 0; i < M && A[i].p != 0; i++)

# {

# printf("| %12d | %c | %2.6f | ",i, A[i].a, A[i].q);

# for (int j = 0; j < A[i].l+1; j++)

# printf("%d", code[i][j]);

# for (int j = A[i].l + 1; j < 18; j++)

# printf(" ");

# printf(" | %7d |\n", A[i].l+1);

# printf("|-----------------------------------------------------------------------------|\n");

# lgm += A[i].p \* A[i].l;

# }

# SetConsoleCP(866);

# printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

# printf("| Энтропия | Средняя длина | Коэф сжатия |\n");

# printf("|\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

# printf("| %10f | %10.5f | %10.5f |\n", entropy, lgm, (float)fcompression/cfcompression);

# printf("|\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n\n\n");

# 

# printf(" entropy + 1 > Lsr\n");

# printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

# printf("| %8f > %8.5f |\n", entropy+1, lgm);

# printf("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

# }

# void CodeBase() {

# FILE \*f, \*fcoded;

# f = fopen("testBase2.dat", "rb");

# fcoded = fopen("BaseCoded.dat", "wb");

# char buffer;

# while (!feof(f)) {

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

# fcompression++;

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

# if (buffer == (char)(i-128)) {

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

# putc(code[i][j], fcoded);

# cfcompression++;

# }

# }

# }

# }

# fclose(f);

# fclose(fcoded);

# }

# void BaseCoding(){

# 

# int i,j;

# FILE \*f;

# f=fopen("testBase2.dat", "rb");

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

# A[i].p = 0;

# A[i].l = 0;

# A[i].q = 0;

# A[i].a = (char)(i-128);

# }

# while (!feof(f)) {

# char c;

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

# if (feof(f))

# break;

# //cout << c<<" - " << (int)c <<endl;

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

# sum++;

# }

# printf("\n");

# fclose(f);

# //ñîðòèðîâêà ïî êîëè÷åñòâó ïîâòîðåíèé ñèìâîëîâ

# bool b = true;

# while (b)

# {

# b = false;

# for (int i = 1; i < M; i++)

# {

# if (A[i - 1].p < A[i].p)

# {

# HF\_code B = A[i-1];

# A[i-1]=A[i];

# A[i]=B;

# b = true;

# }

# }

# }

# for (i = 0; i < M && A[i].p != 0; i++){

# A[i].p /=sum;

# A[i].q = A[i].p;

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

# sym\_count++;

# }

# Huffman(sym\_count-1);

# }

# int Up(int n, float q){

# int j = 0;

# for (int i = n-2; i >0 ; i--)

# {

# if (A[i - 1].q <= q)

# A[i].q = A[i - 1].q;

# else

# {

# j = i;

# break;

# }

# }

# A[j].q = q;

# return j;

# }

# void Down(int n, int j){

# int S[M];

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

# S[i] = code[j][i];

# int L = A[j].l;

# for (int i = j; i < n - 1; i++)

# {

# for (int k = 0; k < M; k++)

# code[i][k] = code[i + 1][k];

# A[i].l = A[i + 1].l;

# }

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

# {

# code[n - 1][i] = S[i];

# code[n][i] = S[i];

# }

# code[n - 1][L + 1] = 0;

# code[n][L + 1] = 1;

# A[n - 1].l = A[n].l = L + 1;

# }

# void Huffman(int n){

# if (n == 1)

# {

# code[0][0] = 0;

# code[1][0] = 1;

# A[0].l = A[1].l = 1;

# }

# else

# {

# float q = A[n - 1].q + A[n].q;

# int j = Up(n, q); // Ïîèñê è âñòàâêà ñóììû

# Huffman(n - 1);

# Down(n, j); // Äîñòðàèâàíèå êîäîâ

# }

# }

# void LRprint(vertex\* x){

# SetConsoleCP(1251);

# if (x) {

# LRprint(x->left);

# cout << " " << x->data->Date << "\t" << x->data->FIO << "\t" << x->data->Department << "\t" << x->data->Position << endl;

# LRprint(x->next);

# LRprint(x->right);

# }

# }

# void addAVL(list \*data, vertex \*&point){

# if (point == NULL) {

# point = new vertex;

# point->data = data->data;

# point->bal = 0;

# point->left = NULL;

# point->right = NULL;

# point->next = NULL;

# up = true;

# }

# else{

# if(compare\_date(point->data->Date, data->data->Date) == 0){

# addAVL(data, point->next);

# up = false;

# }

# else{

# if(compare\_date(point->data->Date, data->data->Date) > 0) {

# addAVL(data, point->left);

# if (up == true) {

# if (point->bal > 0) {

# point->bal = 0;

# up = false;

# }

# else

# if (point->bal == 0) {

# point->bal = -1;

# }

# else

# if (point->left->bal < 0) {

# ll(point);

# up = false;

# }

# else {

# lr(point);

# up = false;

# }

# }

# }

# else{

# if(compare\_date(point->data->Date, data->data->Date) < 0) {

# addAVL(data, point->right);

# if (up == true) {

# if (point->bal < 0) {

# point->bal = 0;

# up = false;

# }

# else {

# if (point->bal == 0) {

# point->bal = 1;

# }

# else {

# if (point->right->bal > 0) {

# rr(point);

# up = false;

# }

# else {

# rl(point);

# up = false;

# }

# }

# }

# }

# }

# }

# }

# }

# }

# void ll(vertex\* &point){

# vertex \*current = point->left;

# point->bal = 0;

# current->bal = 0;

# point->left = current->right;

# current->right = point;

# point = current;

# }

# void rr(vertex\* &point){

# vertex \*current = point->right;

# point->bal = 0;

# current->bal = 0;

# point->right = current->left;

# current->left = point;

# point = current;

# }

# void lr(vertex\* &point){

# vertex \*current = point->left;

# vertex \*reva = current->right;

# 

# if (reva->bal < 0) {

# point->bal = 1;

# }

# else {

# point->bal = 0;

# }

# if (reva->bal > 0) {

# current->bal = -1;

# }

# else {

# current->bal = 0;

# }

# reva->bal = 0;

# current->right = reva->left;

# point->left = reva->right;

# reva->left = current;

# reva->right = point;

# point = reva;

# }

# void rl(vertex\* &point) {

# vertex \*current = point->right;

# vertex \*reva = current->left;

# if (reva->bal < 0) {

# point->bal = 1;

# }

# else {

# point->bal = 0;

# }

# if (reva->bal > 0) {

# current->bal = -1;

# }

# else {

# current->bal = 0;

# }

# reva->bal = 0;

# current->left = reva->right;

# point->right = reva->left;

# reva->right = current;

# reva->left = point;

# point = reva;

# }

# void AVLSearch (vertex\* p, char\* data) {

# if(p!=NULL)

# {

# if(compare\_date(p->data->Date, data) == 0){

# AVLNext(p);

# }else{

# if(compare\_date(p->data->Date, data) > 0){

# AVLSearch(p->left, data);

# }else{

# AVLSearch(p->right, data);

# }

# }

# }

# }

# void AVLNext (vertex\* p) {

# if(p!=NULL)

# {

# cout << " " << p->data->Date << "\t" << p->data->FIO << "\t" << p->data->Department << "\t" << p->data->Position << endl;

# AVLNext(p->next);

# }

# }

# int compare\_department(int a, int b) {

# if(a>=b){

# return 1;

# }

# else{

# return -1;

# }

# }

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

# 

# if (strcmp(a,b) == 0)

# return 0;

# 

# char bufferA[30];

# char bufferB[30];

# strcpy(bufferA, a);

# strcpy(bufferB, b);

# //íàõîäèì ïîçèöèþ ïåðâîãî ïðîáåëà

# int space\_pos\_a = strchr(a,' ') - a;

# int space\_pos\_b = strchr(b,' ') - b;

# 

# //ìèíèìàëüíîå êîëè÷åñòâî ñèìâîëîâ â ôàìèëèè 3

# int count =4;

# //åñëè ñðàâíèâàåìûå ôàìèëèè ðàçíûõ ðàçìåðîâ âûáèðàåì ìåíüøèé ðàçìåð äëÿ ñðàâíåíèé

# if(space\_pos\_a < space\_pos\_b){

# count = space\_pos\_a;

# }

# else{

# count = space\_pos\_b;

# }

# count -= 1;

# 

# //ñðàâíèâàåì ôàìèëèè

# if(strncmp(bufferA, bufferB,count) > 0){

# return 1;

# }

# if(strncmp(bufferA, bufferB,count) < 0){

# return -1;

# }

# //åñëè ñðàâíèâàåìûå ôàìèëèè ðàçíûõ ðàçìåðîâ âûáèðàåì ìåíüøèé

# if(space\_pos\_a > space\_pos\_b){

# return 1;

# }

# if(space\_pos\_a < space\_pos\_b){

# return -1;

# }

# //åñëè ôàìèëèè îäèíàêîâû, òî...

# //íàõîäèì ïîçèöèþ âòîðîãî ïðîáåëà

# strcpy(bufferA, a);

# strcpy(bufferB, b);

# bufferA[space\_pos\_a]='a';

# bufferB[space\_pos\_b]='a';

# space\_pos\_a = strchr(bufferA,' ') - bufferA;

# space\_pos\_b = strchr(bufferB,' ') - bufferB;

# 

# count = 4;

# //åñëè ñðàâíèâàåìûå èìåíà ðàçíûõ ðàçìåðîâ âûáèðàåì ìåíüøèé ðàçìåð äëÿ ñðàâíåíèé

# if(space\_pos\_a < space\_pos\_b){

# count = space\_pos\_a;

# }

# else{

# count = space\_pos\_b;

# }

# count -= 1;

# 

# //ñðàâíèâàåì èìåíà

# if(strncmp(bufferA, bufferB,count) > 0){

# return 1;

# }

# if(strncmp(bufferA, bufferB,count) < 0){

# return -1;

# }

# //åñëè ñðàâíèâàåìûå èìåíà ðàçíûõ ðàçìåðîâ âûáèðàåì ìåíüøèé

# if(space\_pos\_a > space\_pos\_b){

# return 1;

# }

# if(space\_pos\_a < space\_pos\_b){

# return -1;

# }

# 

# //åñëè èìåíà îäèíàêîâû, òî...

# //íàõîäèì ïîçèöèþ òðåòüåãî ïðîáåëà

# bufferA[space\_pos\_a]='a';

# bufferB[space\_pos\_b]='a';

# space\_pos\_a = strchr(bufferA,' ') - bufferA;

# space\_pos\_b = strchr(bufferB,' ') - bufferB;

# 

# count = 4;

# //åñëè ñðàâíèâàåìûå îò÷åñòâà ðàçíûõ ðàçìåðîâ âûáèðàåì ìåíüøèé ðàçìåð äëÿ ñðàâíåíèé

# if(space\_pos\_a < space\_pos\_b){

# count = space\_pos\_a;

# }

# else{

# count = space\_pos\_b;

# }

# count -= 1;

# 

# //ñðàâíèâàåì îò÷åñòâà

# if(strncmp(bufferA, bufferB,count) > 0){

# return 1;

# }

# if(strncmp(bufferA, bufferB,count) < 0){

# return -1;

# }

# //åñëè ñðàâíèâàåìûå èìåíà ðàçíûõ ðàçìåðîâ âûáèðàåì ìåíüøèé

# if(space\_pos\_a >= space\_pos\_b){

# return 1;

# }

# if(space\_pos\_a < space\_pos\_b){

# return -1;

# }

# return 0;

# }

# int compare\_date(char a1[], char b1[]) {

# char a[30];

# char b[30];

# strcpy(a, a1);

# strcpy(b, b1);

# 

# if (strcmp(a, b) == 0) {

# return strcmp(a, b);

# }

# else if (a[6] - b[6] != 0) {

# return a[6] - b[6];

# }

# else if (a[3] - b[3] != 0) {

# return a[3] - b[3];

# }

# else if (a[4] - b[4] != 0) {

# return a[4] - b[4];

# }

# else if (a[0] - b[0] != 0) {

# return a[0] - b[0];

# }

# else if (a[1] - b[1] != 0) {

# return a[1] - b[1];

# }

# return 0;

# }

# void HeapBuild (list \*\*A, int l, int r) {

# int i=l, j;

# list \*x=A[l];

# 

# while (1) {

# j=2\*i;

# if (j+1>=r) break;

# int result = 0;

# if(j<r){

# result = compare\_department(A[j+1]->data->Department, A[j]->data->Department);

# if(result > -1){

# j++;

# }

# }

# 

# result = compare\_department(x->data->Department, A[j]->data->Department);

# if(result > -1){

# break;

# }

# 

# A[i]=A[j];

# i=j;

# }

# A[i]=x;

# }

# void HeapBuild2 (list \*\*A, int l, int r, int offset) {

# int i = l, j = 2 \* i;

# list \*x=A[offset+l];

# 

# while (1) {

# j=2\*i;

# if (j+1>=r) break;

# int result = 0;

# if(j<r){

# result = compare\_name(A[offset+j+1]->data->FIO, A[offset+j]->data->FIO);

# if(result > -1){

# j++;

# }

# }

# 

# result = compare\_name(x->data->FIO, A[offset+j]->data->FIO);

# if(result > -1){

# break;

# }

# A[offset+i] = A[offset+j];

# i = j;

# }

# A[offset+i]=x;

# }

# void HeapSort (int n, list \*\*A) {

# // ñîðòèðîâêà ïî ïåðâè÷íîìó êëþ÷ó - íîìåð îòäåëà

# int l=n/2, r=n;

# while (l+1>0) {

# HeapBuild (A, l, r);

# l--;

# }

# while (r>1) {

# list\* t=A[0];

# A[0]=A[r-1];

# A[r-1]=t;

# r--;

# HeapBuild (A, 0, r-1);

# }

# 

# // ñîðòèðîâêà ïî âòîðè÷íîìó êëþ÷ó - ÔÈÎ

# // ñîðèðóåò îòäåëüíî âñå ÔÈÎ äëÿ êàæäîãî èç îòäåëîâ

# int offset = 0;

# int counter\_max = 0;

# while (offset + counter\_max < n-1){

# if(A[offset+counter\_max]->data->Department != A[offset+counter\_max+1]->data->Department){

# l = counter\_max / 2;

# r = counter\_max+1;

# 

# while (l + 1 > 0) {

# HeapBuild2 (A, l, r, offset);

# l--;

# }

# while (r > 1) {

# list\* t = A[offset];

# A[offset]=A[offset+r-1];

# A[offset+r-1]=t;

# r--;

# HeapBuild2 (A, 0, r-1, offset);

# }

# offset = offset + counter\_max + 1;

# counter\_max = 0;

# }

# else{

# counter\_max += 1;

# }

# }

# 

# // äîñîðòèðîâêà ïî ÔÈÎ äëÿ ñòðîê ñ ìàêñèìàëüíûì íîìåðîì îòäåëà

# l = counter\_max / 2;

# r = counter\_max + 1;

# while (l + 1 > 0) {

# HeapBuild2 (A, l, r, offset);

# l--;

# }

# while (r > 1) {

# list\* t = A[offset];

# A[offset]=A[offset+r-1];

# A[offset+r-1]=t;

# r--;

# HeapBuild2 (A, 0, r-1, offset);

# }

# }

# int BSearch (list \*\*A, int Number) {

# int l = 0, r = N, m = 0;

# while (l < r) {

# m=(l+r)/2;

# if (A[m]->data->Department == Number) {

# return m;

# }

# if (A[m]->data->Department < Number)

# l = m+1;

# else r = m-1;

# }

# return -1;

# }

# void Read\_base(FILE \*fp, list \*base){

# record \*mas = new record();

# fread((record \*)mas, sizeof(record), 1, fp);

# base->data=mas;

# base->prev=NULL;

# base->next=NULL;

# 

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

# mas = new record();

# list \*base\_prev;

# fread((record\*)mas, sizeof(record), 1, fp);

# base\_prev = base;

# base = base->next = new list();

# base->data = mas;

# base->prev = base\_prev;

# base->next = NULL;

# }

# }

# void Print\_base(list \*base){

# int i;

# char n;

# SetConsoleCP(1251);

# cout << "1 for 4000 / 2 for 20 per click/ 3 for nothing"<< endl;

# cin>>n;

# if(n=='1'){

# i=0;

# while(i<N){

# cout<<i<<"\t"<<base->data->Date<<"\t"<<base->data->FIO<<"\t"<<base->data->Department<<"\t"<<base->data->Position <<endl;

# i++;

# if(int result=\_kbhit()){

# printf( "\nKey '%c' was pressed. Press 'e' key to exit.\n", \_getch());

# char something;

# cin>>something;

# if(something=='e'){

# return;

# }

# }

# base=base->next;

# }

# }

# if(n=='2'){

# i=0;

# int g=0;

# for(int i=0;i<N;i=i+20){

# while((g++)<20)

# {

# cout<<i+g<<"\t"<<base->data->Date<<"\t"<<base->data->FIO<<"\t"<<base->data->Department<<"\t"<<base->data->Position <<endl;

# base=base->next;

# }

# if(g>=20){

# cout << "y for continue" << endl;

# cin >> n;

# if (n=='y')

# g=0;

# else

# return;

# }

# }

# }

# 

# 

# }

# void Print\_sort\_index\_mas(list \*\*mas){

# int i;

# char n;

# SetConsoleCP(1251);

# cout << "1 for 4000 / 2 for 20 per click/ 3 for nothing"<< endl;

# cin>>n;

# if(n=='1'){

# i=0;

# while(i<N){

# cout<<i<<"\t"<<mas[i]->data->Date<<"\t"<<mas[i]->data->FIO<<"\t"<<mas[i]->data->Department<<"\t"<<mas[i]->data->Position <<endl;

# i++;

# if(int result=\_kbhit()){

# printf( "\nKey '%c' was pressed. Press 'e' key to exit.\n", \_getch());

# char something;

# cin>>something;

# if(something=='e'){

# return;

# }

# }

# }

# }

# if(n=='2'){

# i=0;

# int g=0;

# for(int i=0;i<N;i=i+20){

# while((g++)<20)

# {

# cout<<i+g<<"\t"<<mas[i+g]->data->Date<<"\t"<<mas[i+g]->data->FIO<<"\t"<<mas[i+g]->data->Department<<"\t"<<mas[i+g]->data->Position <<endl;

# }

# if(g>=20){

# cout << "y for continue" << endl;

# cin >> n;

# if (n=='y')

# g=0;

# else

# return;

# }

# }

# }

# 

# 

# }

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

# b->prev = NULL;

# b->data = a->data;

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

# a = a->next;

# b->next = new list;

# b->next->prev = b;

# b = b->next;

# b->data = a->data;

# }

# b->next = NULL;

# }

# int size(vertex \*x)

# {

# if (x == NULL) {

# return 0;

# }

# else {

# return 1 + size(x->left) + size(x->right);

# }

# }

# int max(int x, int y)

# {

# if (x > y) return x;

# return y;

# }

# int height(vertex \*x)

# {

# if (x == NULL) {

# return 0;

# }

# else {

# return 1 + max(height(x->left), height(x->right));

# }

# }

# int sdp(vertex \*x, int l)

# {

# if (x == NULL) {

# return 0;

# }

# else {

# return l + sdp(x->left, l + 1) + sdp(x->right, l + 1);

# }

# }

# Результаты

Рисунок 1. Неотсортированная база данных.

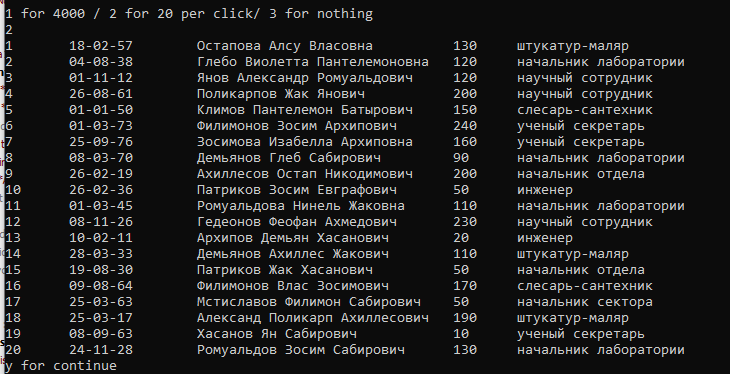


Рисунок 2. Отсортированная база данных по номеру отдела и ФИО.

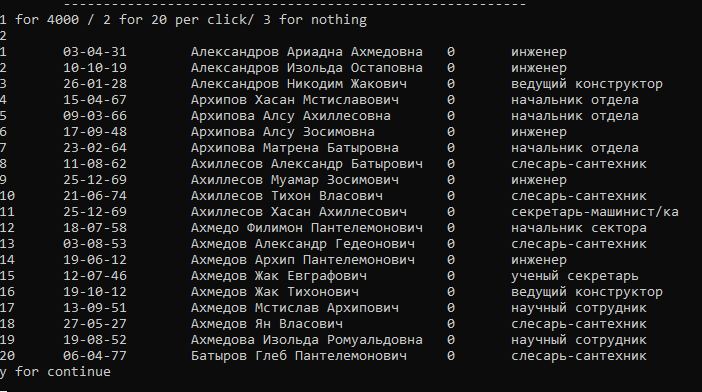


Рисунок 3. Бинарный поиск по году рождения

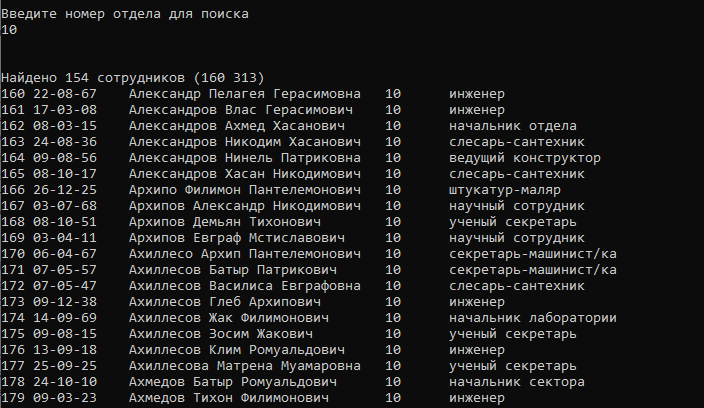


Рисунок 4. Дерево, ключ в дереве – должность.

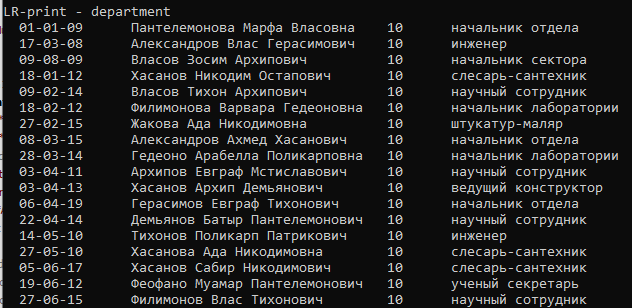


Рисунок 5. Поиск по дереву (элементы с одинаковым ключом).

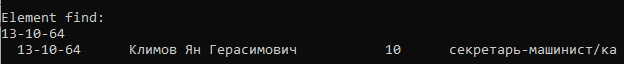
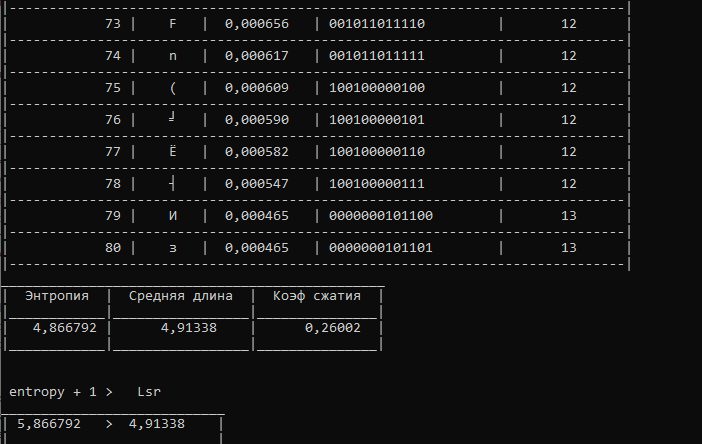


Рисунок 6. Кодовые слова, энтропия и средняя длина кодового слова.



# Выводы

В ходе выполнения курсовой работы были выполнены все поставленные задачи и реализованы необходимые алгоритмы: сортировки, поиска, построения дерева оптимального поиска (приближённый алгоритм А2), 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 + 1).

Четкая структуризация кода и грамотно подобранные имена переменных, структур данных, функций и процедур способствуют удобочитаемости программы.

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