Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации
СибГУТИ

Кафедра ПМиК

КУРСОВОЙ ПРОЕКТ
"Структуры и алгоритмы обработки данных"
ВАРИАНТ 11

Выполнил:
студент группы

Проверил:
Старший преподаватель
Кафедры ПМиК
Солодов П. С.

Новосибирск
2008

СОДЕРЖАНИЕ

1. ПОСТАНОВКА ЗАДАЧИ

Хранящуюся в файле базу данных загрузить в оперативную память компьютера и построить индексный массив, упорядочивающий данные **по дате рождения**, используя **метод прямого слияния (Merge Sort)** в качестве метода сортировки.

Предусмотреть возможность 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 **двоичное Б - дерево поиска по дням рождения**, и предусмотреть возможность поиска в дереве по запросу.

Закодировать файл базы данных статическим **кодом Хаффмена**, предварительно оценив вероятности всех встречающихся в ней символов. Построенный код вывести на экран, упакованную базу данных записать в файл, вычислить коэффициент сжатия данных.

База данных "Предприятие"

Структура записи:

ФИО сотрудника: текстовое поле 32 символа

формат <Фамилия>_<Имя>_<Отчество>

Номер отдела: целое число

Должность: текстовое поле 22 символа

Дата рождения: текстовое поле 8 символов

формат дд-мм-гг

Пример записи из БД:

Петров_Иван_Иванович_____

130

начальник_отдела_____

15-03-46

Варианты условий упорядочения и ключи поиска (К):

по дате рождения, К = год рождения.

Ключ в дереве - дата рождения (как строка).

2. ОСНОВНЫЕ ИДЕИ И ХАРАКТЕРИСТИКИ ПРИМЕНЯЕМЫХ МЕТОДОВ

2.1. Метод сортировки

Метод прямого слияния

В основе метода прямого слияния лежит операция слияния серий. p-серией называется упорядоченная последовательность из p элементов.

Пусть имеются две упорядоченные серии a и b длины q и r соответственно. Необходимо получить упорядоченную последовательность c, которая состоит из элементов серий a и b. Сначала сравниваем первые элементы последовательностей a и b. Минимальный элемент перемещаем в последовательность c. Повторяем действия до тех пор, пока одна из последовательностей a и b не станет пустой, оставшиеся элементы из другой последовательности переносим в последовательность c. В результате получим (q+r)-серию.

Для алгоритма слияния серий с длинами q и r необходимое количество сравнений и перемещений оценивается следующим образом

$\min(q, r) \le C \le q+r-1$, $M=q+r$

Пусть длина списка S равна степени двойки, т.е. 2k, для некоторого натурального k. Разобьем последовательность S на два списка a и b, записывая поочередно элементы S в списки a и b. Сливаем списки a и b с образованием двойных серий, то есть одиночные элементы сливаются в упорядоченные пары, которые записываются попеременно в очереди c0 и c1. Переписываем очередь c0 в список a, очередь c1 – в список b. Вновь сливаем a и b с образованием серий длины 4 и т. д. На каждом итерации размер серий увеличивается вдвое. Сортировка заканчивается, когда длина серии превысит общее количество элементов в обоих списках. Если длина списка S не является степенью двойки, то некоторые серии в процессе сортировки могут быть короче.

Трудоёмкость метода прямого 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 $\lceil \log n \rceil$.Тогда

$$C < n\lceil \log n \rceil,\ M=n\lceil \log n \rceil+n.$$

Дополнительные n перемещений происходят во время начального расщепления исходного списка. Асимптотические оценки для M и C имеют следующий вид

$$C=O(n \log n),\ M=O(n \log n) \text{ при } n \to \infty.$$

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.

2.2. 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 $\lceil \log_2 n \rceil$. Таким образом, трудоёмкость двоичного поиска в обоих случаях

$$C=O(\log n),\ n \to \infty.$$

2.3. Дерево и поиск по дереву

Двоичное Б-дерево

Двоичное Б-дерево состоит из вершин (страниц) с одним или двумя элементами. Следовательно, каждая страница содержит две или три ссылки на поддеревья. На рисунке 1 показаны примеры страниц Б – дерева при m = 1.

Рисунок 1 - Пример страниц Б – дерева

Поэтому вновь рассмотрим задачу построения деревьев поиска в оперативной памяти компьютера. В этом случае неэффективным с точки зрения экономии памяти будет представление элементов внутри страницы в виде массива. Выход из положения – динамическое размещение на основе списочной структуры, когда внутри страницы существует список из одного или двух элементов.

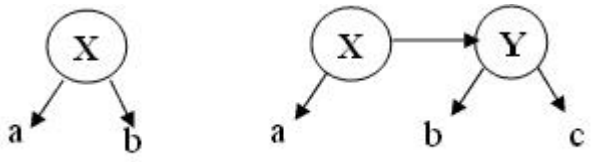


Рисунок 2 – Списочная структура

Таким образом, страницы Б-дерева теряют свою целостность и элементы списков начинают играть роль вершин в двоичном дереве. Однако остается необходимость делать различия между ссылками на потомков (вертикальными) и ссылками на одном уровне (горизонтальными), а также следить, чтобы все листья были на одном уровне.

Очевидно, двоичные Б-деревья представляют собой альтернативу АВЛ-деревьям. При этом поиск в двоичном Б-дереве происходит как в обычном двоичном дереве.

Высота двоичного Б-дерева

$$h = \frac{\log(n+1) - 1}{\log(1+1)} + 1 = \log(n+1)$$

.

Если рассматривать двоичное Б-дерево как обычное двоичное дерево, то его высота может увеличиться вдвое, т.е. $h = 2\log(n+1)$. Для сравнения, в АВЛ-дереве даже в самом плохом случае h<1.44 log n. Поэтому сложность поиска в двоичном Б-дереве и в АВЛ-дереве одинакова по порядку величины.

При построении двоичного Б-дерева реже приходится переставлять вершины, поэтому АВЛ-деревья предпочтительней в тех случаях, когда поиск ключей происходит значительно чаще, чем добавление новых элементов. Кроме того, существует зависимость от особенностей реализации, поэтому вопрос о применение того или иного тапа деревьев следует решать индивидуально для каждого конкретного случая.

2.4. Метод кодирования

Метод Хаффмена

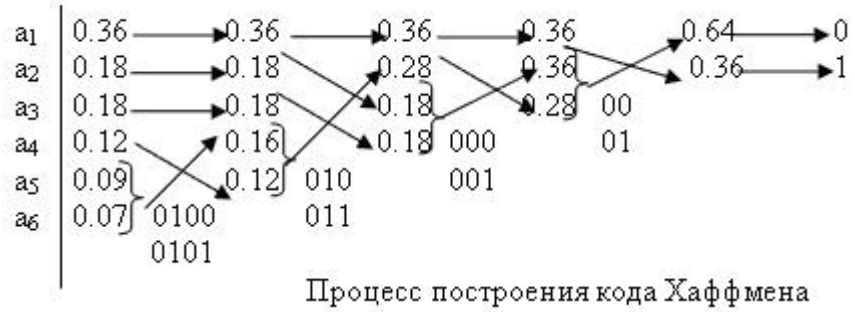
Алгоритм построения оптимального кода Хаффмена

1. Упорядочим символы исходного алфавита $A=\{a_1,\ldots,a_n\}$ по убыванию их вероятностей $p_1 \geq p_2 \geq \ldots \geq p_n$.

2. Если $A=\{a_1,a_2\}$, то $a_1 \rightarrow 0$, $a_2 \rightarrow 1$.

3. Если $A=\{a_1,\ldots,a_j,\ldots,a_n\}$ и известны коды $<a_j \rightarrow b_j>$, j = 1,…,n ,то для $\{a_1,\ldots a_j', a_j'' \ldots, a_n\}$, $p(a_j)=p(a_j')+p(a_j'')$, $a_j' \rightarrow b_j 0$, $a_j'' \rightarrow b_j 1$.

Пусть дан алфавит $A=\{a_1, a_2, a_3, a_4, a_5, a_6\}$ с вероятностями $p_1=0.36$, $p_2=0.18$, $p_3=0.18$, $p_4=0.12$, $p_5=0.09$, $p_6=0.07$. Будем складывать две наименьшие вероятности и включать суммарную вероятность на соответствующее место в упорядоченном списке вероятностей до тех пор, пока в списке не останется два символа. Тогда закодируем эти два символа 0 и 1. Далее кодовые слова достраиваются, как показано на рисунке:



Процесс построения кода Хаффмена

Рисунок 3 – Процесс построения кода Хаффмена

Таблица 1 - Реализация кода Хаффмена

a_i	P_i	L_i	кодовое слово
a_1	0.36	2	1
a_2	0.18	3	000
a_3	0.18	3	001
a_4	0.12	4	011
a_5	0.09	4	0100
a_6	0.07	4	0101

Посчитаем среднюю длину, построенного кода Хаффмена

$L_{ср}(P) = 1.0*36 + 3*0.18 + 3*0.18 + 3*0.12 + 4*0.09 + 4*0.07 = 2.44,$

при этом энтропия данного источника равна

$H = -(0.36*\log0.36 + 2*0.18*\log0.18 + 0.12*\log0.12 + 0.09*\log0.09 + 0.07*\log0.07) = 2.37$

Код Хаффмена 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.

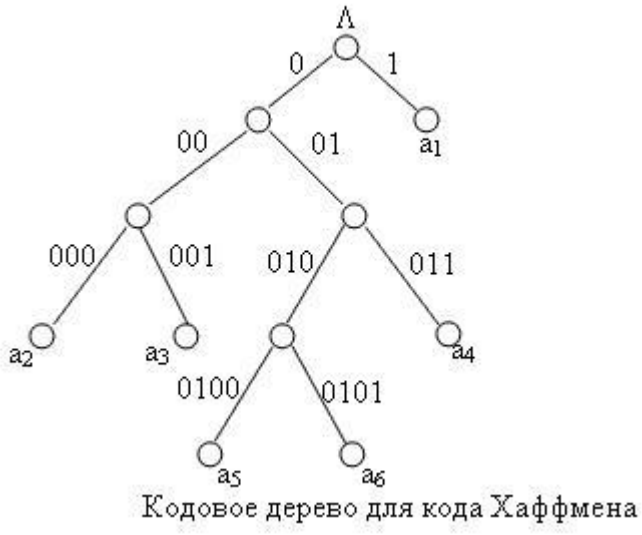


Рисунок 4 – Кодовое дерево кода Хаффмена

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

Для организации интерфейса использовалась процедура *void menu(),* которая обеспечивает корректное и незатруднительное использование программы и предоставляет возможность многократного выбора различных вариантов обработки базы данных, в зависимости от задач пользователя. Визуальное представление пунктов меню вынесено в отдельную процедуру *void info().*

2. Загрузка и вывод базы данных

Для загрузки базы данных разработана процедура *struct qel * LoadMem*, в которой производится считывание записей типа *struct Firma*(«Предприятие»), а из них формируется очередь *struct qel.* Здесь же предусмотрена проверка на наличие файла, откуда выполняется считывание. Данная процедура вызывается независимо от желания пользователя, в то время как остальные он может выбрать посредствам меню.

За вывод элементов считанной базы данных отвечает процедура *void ViewBase Она предоставляет возможность постраничного просмотра базы данных(по 4 элемента на странице), смена страниц осуществляется нажатием управляющих стрелок «вверх» и «вниз» на клавиатуре. Есть возможность прервать просмотр в любой момент времени нажатием клавиши «Esc».*

3. Вспомогательные функции и процедуры для сортировки данных

При сортировке базы данных потребовалось реализовать дополнительную процедуру *void InvertDate* для преобразования даты рождения в формат, соответствующий условию упорядочивания(«дд-мм-гг» в «гг-мм-дд»), а также определить операцию сравнения двух строк(не используя стандартные процедуры среды) для последующего применения в алгоритме сортировки. Данную функция выполняет процедура *char compare, которая поэлементно сравнивает две строки и при первом различии возвращает результат.*

4. Особенности реализации бинарного поиска

Для того чтобы без проблем многократно осуществлять поиск элементов, соответствующих разным ключам, требуется каждый раз создавать новую очередь, и чтобы постоянно не выделять память(которая, как известно, не безгранична) процедура void FreeQueue – очищает , ту что была распределена при предыдущем вызове функции построения очереди - *void MakeQueue*. Новая очередь же, строится непосредственно после выполнения

поиска. При его реализации была использована вторая версия двоичного поиска, так как в результате ее выполнения возвращается номер самого левого из найденных элементов, благодаря чему легко найти и вывести остальные элементы, лишь просмотрев оставшуюся правую часть массива.

5. Вспомогательные функции и процедуры для построения двоичного Б-дерева

Также как и для очереди, при неоднократном построении дерева требуется освобождать память, эту функцию выполняет процедура *void FreeTree.* Для вывода дерева на экран используется процедура *void PrintTree*, представляющая собой обход дерева слева – направо.

Аналогичная процедура *void PrintSearch* выполняет вывод результатов поиска в дереве.

6. Кодирование данных

При побуквенном кодировании существует необходимость знать вероятности встречаемости символов. Для их подсчета создана процедура *void Probabilities()*, в которой помимо вычисления значений вероятностей производится их вывод и и вывод алфавита кодируемого текста в сортированном по вероятностям и несортированном виде. Для сортировки массива вероятностей использовалась процедура сортировки методом Хоара, так как обладает сравнительно небольшой трудоемкостью и применима непосредственно для сортировки массивов.

И, наконец, для вычисления характеристик полученного кода разработана процедура *void parametrs*, где вычисляются основные параметры(средняя длина кодового слова и энтропия), а также производится оценка их соотношения.

✓ Подробное описание основных и вспомогательных функций и процедур, алгоритмы их работы и параметры приведены далее в разделе «ОПИСАНИЕ ПРОГРАММЫ»

4. ОПИСАНИЕ ПРОГРАММЫ

4.1. Основные переменные и структуры

*** * ***

```
struct Firma{
char Sotrudnik[32];
        unsigned short int Nomer;
        char Dolgnost[22];
        char DataR[8];
};
```

Запись, используемая для работы с базой данных «Предприятие».

*** * ***

```
struct qel{
        struct qel *next;
        struct Firma *data;
};
```

Структура(список), используемая при сортировке базы данных.

struct qel *next – указатель на следующие элемент;

struct Firma *data – поле данных(адрес элемента в основном массиве структур «Предприятие»).

*** * ***

```
struct queueS {
        qel *head;
        qel *tail;
};
```

Структура(очередь), используемая при сортировке базы данных.

qel *head – голова очереди;

qel *tail – хвост очереди.

*** * ***

```
struct queue {
        int index;
        struct queue *next;
}
```

Структура, используемая при построении очереди из элементов, полученных в результате бинарного поиска.

int index – индекс элемента в базе данных;

struct queue *next – указатель на следующий элемент.

*** * ***

```
struct derevo{
  int x;
        int balance;
        struct derevo *left;
        struct derevo *right;
}
```

Структура, представляющая двоичное Б – дерево. Где ***int x*** - индекс элемента из базы данных, а не сам элемент.

int balance – баланс в вершине(больше 0 если правое поддерево на 1 выше левого, меньше 0, если выше левое и равно 0 при равных высотах левого и правого поддеревьев);

struct derevo *left, struct derevo *right – указатели на левое и правое поддеревья.

*** * ***

int index[MAX+1];

Индексный массив на 3999+1 элементов.

*** * ***

struct Firma *el[MAX];

Указатель на массив структур «Предприятие».

*** * ***

float P[256] – массив вероятностей встречаемости символов;

P1[256] – копия массива P[] для подсчета характеристик сжатия после кодирования.

*** * ***

int maxpower – количество различных символов в базе данных**;**

long amount – общее количество символов(используется для подсчета вероятностей);

4.2. Описание подпрограмм

Процедуры, вывода меню:

1. void menu();

2. void info();

Реализуют меню, ***info*** – отвечает только за визуальное представление, ***menu*** – непосредственно за функциональное.

Процедуры начальной обработки базы данных:

3. struct qel * LoadMem();

4. void ViewBase(struct Firma **el,int iN);

LoadMem – считывание базы из файла и представление ее элементов в форме вышеперечисленных структур, возвращает указатель на массив записей «Предприятие».

ViewBase - просмотр базы. ***struct Firma **el*** – указатель на первый элемент, ***iN*** – общее количество записей в базе.

Функции и процедуры сортировки:

5. void InvertDate(char aData[],int n);

6. char compare(char aData[],char bData[],int n);

7. int Devide(qel* &s, qel* &a, qel* &b);

8. void Merging(qel* &a, qel* &b, queueS &c, int q, int r);

9. qel* MergeSort(qel* s);

InvertDate – вспомогательная процедура для преобразования даты (представленной в виде символьного массива **char aData[],** размерностью ***n***) в формат соответствующий условию упорядочивания(«дд-мм-гг» в «гг-мм-дд»).

Compare – определение операции сравнения двух массивов: **char aData[],char bData[]** размерностью ***n***. Возвращает ***1*** в случае, если $a > b$ и ***2***, если $a \leq b$.

Devide, Merging, MergeSort – реализация сортировки.

Devide – разделение последовательности ***s*** на два списка ***a и b,*** возвращает количество элементов в списке ***s.***

Merging – слияние ***q – серии*** из списка ***a*** с ***r – серией*** списка ***b***, запись результата в очередь ***c.***

MergeSort – инициализация очередей, сама процедура сортировки элементов последовательности ***s,*** возвращает голову(head) первой из двух инициализированных очередей.

Функции и процедуры для поиска в отсортированной базе данных:

10. int BinSearch(struct Firma **x,int N,int *pointers,char *value);

11. void FreeQueue(queue *p);

12. void MakeQueue(char *n,struct queue *pq,int *index,int pos);

13. void PrintQueue(struct queue *p);

BinSearch – процедура двоичного поиска (версия 2), ***struct Firma **x*** – указатель на массив записей, в котором осуществляется поиск, ***N*** – количество записей(изначально - правая граница поиска), ***pointers*** – указатель на индексный массив, через который происходит обращение к элементам, ***value*** – ключ поиска. Возвращает позицию найденного элемента и ***-1***, в случае его отсутствия.

FreeQueue – освобождение памяти для очереди ***p***, если, например, она уже создавалась.

MakeQueue – построение очереди из результатов поиска. ***n*** – самый левый из найденных элементов, ***pq*** – голова очереди, ***index*** – указатель на индексный массив, ***pos*** – позиция, в которой был найден нужный элемент, от нее просматриваем массив только вправо.

PrintQueue – вывод очереди ***p*** **(struct queue *p –** указатель на первый элемент очереди**)** на экран.

Процедуры и функции построения двоичного Б-дерева:

14. void FreeTree(derevo *p);

15. void CreateDBD(int D, struct Firma **base, struct derevo **p,int *index);

16. void PrintTree(struct Firma **x,struct derevo *p,int *index);

17. struct derevo *SearchInTree(char key[],struct Firma **x,struct derevo *p);

18. void PrintSearch(char key[],struct Firma **x,struct derevo *p);

FreeTree – освобождение памяти для построения дерева, чтобы не возникало проблем, в случае если до этого дерево уже создавалось (***derevo *p*** – указатель на корень дерева).

CreateDBD – непосредственно построение, ***D*** – данные, помещаемые в вершину (индекс элемента), ***base*** – указатель на массив структур (обращение к нему происходит при сравнении элементов через индексный массив), ***p*** – указатель на корень дерева, ***index*** - указатель на индексный массив.

PrintTree – обход дерева с корнем ***derevo *p,*** *используемый для вывода на экран отсортированных по дате рождения(дата рождения как строка) элементов базы данных(****struct Firma **x*** – указатель на массив структур «Предприятие»*), в соответствии с индексным массивом –* ***index*** (***int *index*** – указатель на него), элементы которого хранятся в вершинах дерева ***p***.

SearchInTree – поиск в дереве с корнем ***derevo *p*** элементов, соответствующих ключу ***char key[], struct Firma **x*** – указатель на массив структур, к которому обращаемся при поиске, используя вершины дерева ***p*** в качестве индексов. Возвращает адрес вершины, в которой хранится индекс найденного элемента и NULL, в случае его отсутствия.

PrintSearch – вывод на экран результатов поиска (обход поддерева, начиная с вершины с адресом **struct derevo *p**, в которой был найден первый элемент, соответствующий

ключу поиска, до того, пока не закончатся все элементы удовлетворяющие заданному условию), параметр ***struct derevo *p*** обычно изначально принимает значение, возвращаемое предыдущей функцией). Остальные параметры такие же, как и в вышеописанной процедуре.

Процедуры и функции кодирования базы данных:

19. void Probabilities();
20. void quick(float *x,int n);
22. void quicksort(float *x,int left, int right);
23. void huffman(int n);
24. int Up(int n,float q1);
25. void Down(int n,int j);
26. void parametrs(int K,float *P);

Probabilities – процедура определения вероятностей встречаемости символов. Символы считываются из файла, описанного макроопределением ***#define default "D:\\base2.dat"***

quick – основной вызов процедуры сортировки массива, представленного указателем ***float *x***, размерностью ***n***.

quicksort – сортировка вероятностей встречаемости символов методом Хоара. ***float *x*** – указатель на сортируемый массив, ***int left, int right*** – левая и правая границы сортируемого фрагмента.

Huffman – составление матрицы кодовых слов для алфавита мощностью ***int n*** – количеством различных символов в кодируемом тексте.

Up – процедура поиска подходящей позиции для суммы вероятностей двух последних символов, ее вставка и сдвиг остальных элементов. ***int n*** – нижняя граница поиска, ***float q1*** – вставляемая сумма.

Down – процедура формирования кодовых слов. ***int n*** – количество строк в матрице кодовых слов, равное количеству различных символов, ***int j*** – номер строки, которая, на данном этапе, будет являться часть нового кодового слова.

Parameters – подсчет характеристик сжатия базы данных (средней длины кодового слова, энтропии и коэффициента сжатия). ***int K*** – общее количество закодированных символов, ***float *P*** – указатель на массив вероятностей.

Основная программа

int main() – в основной программе вызывается только меню.

5. ТЕКСТ ПРОГРАММЫ

```
/*-------------------------------------------------------------
--------------*/
/* Course work SAOD ®
*/
/* Ivanteeva A.V. IVT P-64
*/
/* N 11   B  C  S  D  E
*/
/*       2  3  3  2  1
*/
/*-------------------------------------------------------------
--------------*/
#include <stdio.h>
#include <conio.h>
#include <IO.H>
#include <FCNTL.H>
#include <stdlib.h>
#include <string.h>
#include <math.h>
/*-------------------------------------------------------------
--------------*/
/* constantes
*/
/*-------------------------------------------------------------
--------------*/
#define MAX 3999
#define Base_name "D:\\base2.dat"
#define default "D:\\base2.dat"
/*-------------------------------------------------------------
--------------*/
/* variables
*/
/*-------------------------------------------------------------
--------------*/
int   index[MAX+1];
int   VR = 0, HR = 0;
/*-------------------------------------------------------------
--------------*/
struct Firma{
        char Sotrudnik[32];
        unsigned short int Nomer;
        char Dolgnost[22];
        char DataR[8];
};
/*-------------------------------------------------------------
--------------*/
struct qel{
        struct qel *next;
        struct Firma *data;
};
/*-------------------------------------------------------------
--------------*/
struct queueS {
        qel *head;
        qel *tail;
};
/*-------------------------------------------------------------
--------------*/
struct queue {
        int index;
        struct queue *next;
} *headq=NULL,*tailq,*spis;
/*-------------------------------------------------------------
--------------*/
struct derevo{
    int x;
        int balance;
        struct derevo *left;
        struct derevo *right;
} *Dbd, *q;
struct Firma *el[MAX];
/*-------------------------------------------------------------
--------------*/
FILE *fin;
float P[256],P1[256];
int       p_to_s[256];
int       s_to_p[256];
int maxpower; long amount; int i,j,k;
/*-------------------------------------------------------------
--------------*/
float q1;
unsigned char C[256][30],L[256];
unsigned char S[30],l;
/*-------------------------------------------------------------
--------------*/
/* prototypus&functions
*/
/*-------------------------------------------------------------
--------------*/
/*  1 */ void   menu();
/*  2 */ void   info();
    /*-------------------------------------------------------------
------------*/
/*  3 */ struct qel * LoadMem();
/*  4 */ void   ViewBase(struct Firma **el,int iN);
    /*-------------------------------------------------------------
------------*/
/*  5 */ void   InvertDate(char aData[],int n);
/*  6 */ char   compare(char aData[],char bData[],int
n);
/*  7 */ int    Devide(qel* &s, qel* &a, qel* &b);
/*  8 */ void   Merging(qel* &a, qel* &b, queueS &c,
int q, int r);
/*  9 */ qel*   MergeSort(qel* s);
    /*-------------------------------------------------------------
------------*/
/* 10 */ int    BinSearch(struct Firma **x,int N,int
*pointers,char *value);
/* 11 */ void   FreeQueue(queue *p);
/* 12 */ void   MakeQueue(char *n,struct queue
*pq,int *index,int pos);
/* 13 */ void   PrintQueue(struct queue *p);
    /*-------------------------------------------------------------
------------*/
/* 14 */ void   FreeTree(derevo *p);
/* 15 */ void   CreateDBD(int D,struct Firma **base,
struct derevo **p,int *index);
/* 16 */ void   PrintTree(struct Firma **x,struct
derevo *p,int *index);
/* 17 */ struct derevo *SearchInTree(char key[],struct
Firma **x,struct derevo *p);
/* 18 */ void   PrintSearch(char key[],struct Firma
**x,struct derevo *p);
    /*-------------------------------------------------------------
------------*/
/* 19 */ void   Probabilities();
/* 20 */ void   quick(float *x,int n);
/* 22 */ void   quicksort(float *x,int left, int right);
/* 23 */ void   huffman(int n);
/* 24 */ int    Up(int n,float q1);
/* 25 */ void   Down(int n,int j);
/* 26 */ void   parametrs(int K,float *P);
/*-------------------------------------------------------------
--------------*/
/* 1 - 2                                                    */
```

```
/*-------------------------------------------------------------
--------------*/
void menu(){
    int i;
    char ch;
    struct qel *head,*tail,*p;
    head=LoadMem();
    printf("\n  Press any key to back to menu...");
    getch();
    for (i=0;i<=MAX;i++) index[i]=i;
    i=0;
    for (p=head;p!=NULL;p=p->next)
el[index[i++]]=p->data;
    while (1){
        system("cls");
        info();
        ch=getch();

        switch (ch){

        case '1':
            system("cls");
            i=0;
              qel *q;
            for (p=MergeSort(head);p!=NULL;p=p-
>next){
             el[index[i++]]=p->data;
              }
            printf("\n  Base was sorted");
            printf("\n  Press any key to back to
menu...");
            getch();
            break;

        case '2':
            system("cls");
            printf("\n  BASE  \n");
            ViewBase(el,i);
            printf("\n  THE END OF BASE  \n");
            printf("\n  Press any key to back to
menu...");
            getch();
            break;

        case '3':
            system("cls");
            printf("\n  Enter key of search (format 'yy'):
");
            char x[2];
            int y;
            scanf("%s",x);
            y=BinSearch(el,MAX,index,x);
            system("cls");
            if (y==-1){ printf("  Element not
found");getch();break; }
            FreeQueue(spis);
            MakeQueue(x,spis,index,y);
            spis=headq;
            printf("\n  QUEUE  \n");
            PrintQueue(spis);
              printf("\n  THE END OF QUEUE  \n");
              printf("\n  Press any key to back to
menu...");
            getch();
            break;

        case '4':
            system("cls");
            spis=headq;
            VR = 0, HR = 0;
            FreeTree(Dbd); Dbd = NULL;
            if (spis==NULL){
           printf("\n  Tree is not created");
  getch();
  break;
              }
        while (spis!=NULL){
              CreateDBD(spis-
>index,el,&Dbd,index);
              spis=spis->next;
        }
            derevo *temp;
            temp=Dbd;
            printf("\n  TREE  \n");
            PrintTree(el,temp,index);
            printf("\n  THE END OF TREE  \n");
            printf("\n  Press any key to back to
menu...");
            getch();
            break;

        case '5':
            system("cls");
            char ch[8];
            struct derevo *tmp, *searchSubTree;
            printf("\n  Enter key of search in tree
(format 'dd-mm-yy'): ");
            scanf("%s",&ch);
            tmp=Dbd;

searchSubTree=SearchInTree(ch,el,tmp);
            if (searchSubTree!=NULL)
PrintSearch(ch,el,tmp);
            else printf("\n  Element not found");
            printf("\n  Press any key to back to
menu...");
            getch();
            break;
        case '6':
            Probabilities();
                    for (int i=0;i<maxpower;i++)
P1[p_to_s[i]]=P[p_to_s[i]];
                    huffman(maxpower);
                    parametrs(maxpower,P1);
                    printf("\n  Press any key to back to
menu...");
            getch();
            break;
        case 27:
            exit(1);
         }
    }
}
/*-------------------------------------------------------------
--------------*/
void info(){
    printf("\n");
            printf("  [ 1 ] Sort base\n");
            printf("  [ 2 ] Print base\n");
            printf("  [ 3 ] Create queue\n");
            printf("  [ 4 ] Create tree\n");
            printf("  [ 5 ] Search in tree\n");
            printf("  [ 6 ] Coding file\n");
            printf("  [ESC] Exit\n");
}
/*-------------------------------------------------------------
--------------*/
/* 3 - 4                                                      */
/*-------------------------------------------------------------
--------------*/
```

```
struct qel * LoadMem(){
        int i=0, j, f;
        struct qel *head,*p,*tail;

        printf("\n  Load base...\n");

        f = open(Base_name, O_RDONLY |
O_BINARY);
    head=NULL;
    tail=head;
    while (!eof(f)){
      p = (struct qel *)malloc(sizeof(struct qel));
            if (p==NULL){
                printf("\n  ERROR: Out
of Memory");
                getch();
              exit(0);
        }
        p->data = (struct Firma
*)malloc(sizeof(struct Firma));
            if (p->data==NULL){
                printf("\n  ERROR: Out
of Memory");
                getch();
                exit(0);
            }
                read(f, p-
>data,sizeof(struct Firma));

                p->next=NULL;
          if (head!=NULL) tail->next=p;
          else head=p;
                tail=p;
                i++;
        }
        close(f);
        printf("\n  Base was loaded\n");
    return(head);
}
/*-----------------------------------------------------------------
--------------*/
void ViewBase(struct Firma **el,int iN){
    int i,j,end;
    char key;
    j=0;
    do {
      if (iN>=4*(j+1)) end=4*(j+1);
      else end=iN;
      for (i=4*j;i<end;i++){
        printf("\n%4d. ",index[i]+1);
        printf("%.32s",el[index[i]]->Sotrudnik);
        printf("\n     %d",el[index[i]]->Nomer);
        printf("\n     %.22s",el[index[i]]->Dolgnost);
        printf("\n     %.8s",el[index[i]]->DataR);
        printf("\n");
      }
      printf("\n  ESC - back to menu");
      key=getch();
      if ((key==72)&&(j>0)) {j--; system("cls");}
      if ((key==80)&&((j+1)*3<iN)){ j++;
system("cls");}
    } while(key!=27);
}
/*-----------------------------------------------------------------
--------------*/
/* 5 - 9                                                          */
/*-----------------------------------------------------------------
--------------*/
void InvertDate(char aData[],int n){
    char a[n];
    int i=0, j=n-2;
    while (i<n){
        a[i]=aData[j];
        a[++i]=aData[j+1];
        a[++i]='-';
        i++;
        j=j-3;
    }
    for (i=0;i<n;i++) aData[i]=a[i];
}
/*-----------------------------------------------------------------
--------------*/
char compare(char aData[],char bData[],int n){
    char a[n], b[n]; int i;
    for (i=0;i<n;i++){
      a[i]=aData[i];
      b[i]=bData[i];
      }
    InvertDate(a,n);
    InvertDate(b,n);
    for (i=0;i<n;i++){
      if(a[i]>b[i])return 2;
      else if(a[i]<b[i])return 1;
    }
    return 1;
}
/*-----------------------------------------------------------------
--------------*/
int Devide(qel* &s, qel* &a, qel* &b){
        qel *k,*p;
        int n;

        a=s;
        b=s->next;

        n=1;
        k=a;
        p=b;
   while (p){
            n++;
          k->next=p->next;
            k=p;
            p=p->next;
        }
        return n;
}
/*-----------------------------------------------------------------
--------------*/
void Merging(qel* &a, qel* &b, queueS &c, int q,
int r){
         qel *p;

        while ((q!=0)&&(r!=0)){
                if (compare(a->data->DataR,b-
>data->DataR,8) == 1){
                    p=a;
                    a=a->next;
                    p->next=NULL;
                    if (c.head) c.tail->next=p;
                    else c.head=p;
                    c.tail=p;
                    q--;
                }
                else{
                     p=b;
                     b=b->next;
                     p->next=NULL;
                     if (c.head) c.tail-
>next=p;
                     else c.head=p;
```

```
                        c.tail=p;
                        r--;
        }
    }
        while (q>0){
                p=a;
                a=a->next;
            p->next=NULL;
                if (c.head) c.tail->next=p;
                else c.head=p;
                c.tail=p;
                q--;
    }
        while (r>0){
                p=b;
                b=b->next;
                p->next=NULL;
                if (c.head) c.tail->next=p;
                else c.head=p;
                c.tail=p;
                r--;
        }
}
/*-------------------------------------------------------------
--------------*/
qel* MergeSort(qel* s){
        int n,p,q,r,m,i;
        struct queueS Cs[2];
        struct qel *a,*b;
        n=Devide(s,a,b);
        p=1;
        while (p<n){
                Cs[0].head=NULL;
                Cs[1].head=NULL;
                Cs[0].tail=Cs[0].head;
                Cs[1].tail=Cs[1].head;
                i=0;
                m=n;
                while (m>0){
                        if (m>=p) q=p;
                        else q=m;
                        m-=q;
                        if (m>=p) r=p;
                        else r=m;
                        m-=r;

Merging(a,b,Cs[i],q,r);
                        i=1-i;
        }
            a=Cs[0].head;
            b=Cs[1].head;
                p*=2;
    }
        Cs[0].tail->next=NULL;
        return Cs[0].head;
}
/*-------------------------------------------------------------
--------------*/
/* 10 - 13
*/
/*-------------------------------------------------------------
--------------*/
int BinSearch(struct Firma **x,int N,int
*pointers,char *key){
                int m,L,R;
                int i; char y[2];
                L=0;
                R=N;
                while(L<R){
                                m=(L+R)/2;
                                for(i=0;i<2;i++)
y[i]=x[pointers[m]]->DataR[i+6];
                                if
(strncmp(y,key,2)<0) L=m+1;
                                else R=m;
                        }
     for(i=0;i<2;i++) y[i]=x[pointers[R]]-
>DataR[i+6];
                        if(strncmp(y,key,2)==0)return R;
                        else return -1;
}
/*-------------------------------------------------------------
--------------*/
void FreeQueue(queue *p){
    queue *q;
    while (p!=NULL){
        q=p;
        p=p->next;
        free(q);
    }
    p=NULL;
}
/*-------------------------------------------------------------
--------------*/
void MakeQueue(char *n,struct queue *pq,int
*index,int pos){
    char y[2];
    headq=NULL;
    while (1){
        for (int i=0;i<2;i++) y[i]=el[index[pos]]-
>DataR[i+6];
        if (strncmp(y,n,2)!=0) break;
        pq = (queue *)malloc(sizeof(queue));
        pq -> next = NULL;
        pq -> index = index[pos];
        if (headq != NULL) tailq -> next = pq;
        else
        headq = pq;
        tailq = pq;
        pos++;
        if(pos==4000)break;
    }

}
/*-------------------------------------------------------------
--------------*/
void PrintQueue(struct queue *p){
            while (p!=NULL){
        printf("\n  %.32s",el[p->index]->Sotrudnik);
        printf("  %4d",el[p->index]->Nomer);
        printf("  %.22s",el[p->index]->Dolgnost);
        printf("  %.8s \n",el[p->index]->DataR);
        p=p->next;
            }
}
/*-------------------------------------------------------------
--------------*/
/* 14 - 18
*/
/*-------------------------------------------------------------
--------------*/
void FreeTree(derevo *p){
    if (p!=NULL){
      FreeTree(p->left);
      FreeTree(p->right);
      free(p);
    }
    p=NULL;
}
```

```
/*-------------------------------------------------------------
--------------*/
void CreateDBD(int x,struct Firma **base, struct
derevo **p,int *index){
      if (!(*p)){
            (*p)=new(struct derevo);
            (*p)->x=x; (*p)->left=NULL; (*p)-
>right=NULL;
            (*p)->balance=0; VR=1;
          }
          else{
                    if (strcmp(base[x]-
>DataR,base[(*p)->x]->DataR)<=0){
                     CreateDBD(x ,base, &(*p)-
>left,index);
                     if (VR==1){
                                  if ((*p)-
>balance==0){
                                    q=(*p)->left;
                                (*p)->left=q->right;
                                    q->right=(*p);
                                    (*p)=q;
                                    (*p)-
>balance=1;
                                    VR=0; HR=1;
                                  }
                                  else{
                                     (*p)-
>balance=0; HR=0; VR=1;
                                  }
                           }
                           else{
                               HR=0;
                           }
                  }
                  else{
                           if (strcmp(base[x]-
>DataR,base[(*p)->x]->DataR)>0){
                             CreateDBD(x, base,
&(*p)->right,index);
                             if (VR==1){
                                    (*p)-
>balance=1;
                                    VR=0;
                                    HR=1;
                                    }
                             else{
                                          if
(HR==1){

          if ((*p)->balance>0){

q=(*p)->right;

(*p)->right=q->left;

(*p)->balance=0;

q->balance=0;

q->left=(*p);

(*p)=q;

VR=1;

HR=0;

          }
          else{

HR=0;

          }
                                                  }
                                          }
                                  }
                          }
                  }
          }
}
/*-------------------------------------------------------------
--------------*/
void PrintTree(struct Firma **x,struct derevo *p,int
*index){
          if (p!=NULL){
     PrintTree(x,p->left,index);
     printf("\n  %.32s",x[p->x]->Sotrudnik);
     printf("  %4d",x[p->x]->Nomer);
     printf("  %.22s",x[p->x]->Dolgnost);
     printf("  %.8s \n",x[p->x]->DataR);
                  PrintTree(x,p->right,index);
          }
}
/*-------------------------------------------------------------
--------------*/
struct derevo *SearchInTree(char key[],struct
Firma **x,struct derevo *p){
            while (p!=NULL){
                     if (strncmp(key,x[p->x]-
>DataR,8)<0)p=p->left;
                                else if (strncmp(key,x[p-
>x]->DataR,8)>0)p=p->right;
                                          else return p;
     }
          return NULL;
}
/*-------------------------------------------------------------
--------------*/
void PrintSearch(char key[],struct Firma **x,struct
derevo *p){
          if (p!=NULL){
     PrintSearch(key,x,p->left);
                  if (strncmp(key,x[p->x]-
>DataR,8)==0){
                    printf("\n  %.32s",x[p->x]-
>Sotrudnik);
        printf("  %4d",x[p->x]->Nomer);
        printf("  %.22s",x[p->x]->Dolgnost);
        printf("  %.8s \n",x[p->x]->DataR);
                  }
                  PrintSearch(key,x,p->right);
          }
}
/*-------------------------------------------------------------
--------------*/
/* 19 - 26
*/
/*-------------------------------------------------------------
--------------*/
void Probabilities(){
   unsigned char ch[1];
   int i, nsymbols=0;
   maxpower=0;
   system("cls");
   if ((fin=fopen(default,"r+"))==NULL) printf("File
not found!");
   printf("\n   Probablities of symbols: \n");
   printf("   ------------------------\n\n   ");
```

```
    while (!feof(fin)){
        fread(ch,1,1,fin);
        i=ch[0];
        if (P[i]==0){
          p_to_s[maxpower]=i;
          s_to_p[i]=maxpower;
          maxpower++;
        }
        P[i]++;
        amount++;
    }
    for (i=0;i<256;i++){
       if(nsymbols==6){ printf("\n   "); nsymbols=0; }
       P[i]/=(float)amount;
       printf("%0.3f (%1c)  ",P[i],i);
       nsymbols++;
    }
    printf("\n\n   N symbols in file: %ld\n   N different
symbols in file: %d\n",amount,maxpower);
    quick(P,maxpower);
    printf("\n   Sorting symbols:\n");
    printf("   ----------\n\n");
    for (i=0;i<maxpower;i++) printf("(%c)
",p_to_s[i]);
    fclose(fin);
    getch();
}
/*-------------------------------------------------------------
--------------*/
void quick(float *x,int n){
                    quicksort(x,0,n-1);
                    return;
}
/*-------------------------------------------------------------
--------------*/
void quicksort(float *x,int left, int right){
    register int i,j;
    float xx,tmp;

    i=left;
    j=right;
    xx=x[p_to_s[(left+right)/2]];
    do{
      while (x[p_to_s[i]]>xx&&i<right) i++;
      while (xx>x[p_to_s[j]]&&j>left)  j--;
      if (i<=j){
        tmp=s_to_p[p_to_s[i]];
        s_to_p[p_to_s[i]]=s_to_p[p_to_s[j]];
        s_to_p[p_to_s[j]]=tmp;
        tmp=p_to_s[i];
        p_to_s[i]=p_to_s[j];
        p_to_s[j]=tmp;
        i++; j--;
      }
    }
    while (i<=j);
    if (left<j) quicksort(x,left,j);
    if (i<right) quicksort(x,i,right);
}
/*-------------------------------------------------------------
--------------*/
void huffman(int n){
          int j;
          if (n==2){
                  C[0][0]=0;L[0]=1; C[1][0]=1;
L[1]=1;
    }
          else {
                   q1=P[p_to_s[n-2]]+P[p_to_s[n-
1]];
                   j=Up(n,q1);
                   huffman(n-1);
                   Down(n,j);
    }
}
/*-------------------------------------------------------------
--------------*/
int       Up(int n,float q1){
          for (i=n-2;i>=1;i--){
                  if (P[p_to_s[i-1]]<q1)
        P[p_to_s[i]]=P[p_to_s[i-1]];
                  else { k=i; break; }
                  if ((i-1)==0&&P[p_to_s[i-1]]<q1){
k=0;break; }
          }
          P[p_to_s[k]]=q1;
          return (k);
}
/*-------------------------------------------------------------
--------------*/
void Down(int n,int j){
          for(k=0;k<30;k++)S[k]=C[j][k];
          l=L[j];
          for (i=j;i<n-2;i++){

for(k=0;k<30;k++)C[i][k]=C[i+1][k];
                   L[i]=L[i+1];
    }
          for (k=0;k<30;k++){
      C[n-2][k]=S[k]; C[n-1][k]=S[k];
    }
          C[n-2][l]=0; C[n-1][l]=1;
          L[n-2]=l+1; L[n-1]=l+1;
}
/*-------------------------------------------------------------
--------------*/
void parametrs(int K,float *P){
          float S=0;
          printf("\n\n   Coding words:\n");
          printf("   ----------\n\n   ");
          for(i=0;i<maxpower;i++){
                  for(j=0;j<L[i];j++){
                          printf("%d",C[i][j]);
                  }
                                  printf(" %c
(%d)\n   ",p_to_s[i],p_to_s[i]);
           }
    for (i=0;i<K;i++) S += P[p_to_s[i]]*(float)L[i];
    printf("\n\n   ----------");
          printf("\n    L sr = %.4f",S);
          float M=0;
    for (i=0;i<K;i++)
       M += ((-
(log(P[p_to_s[i]])/log(2)))*P[p_to_s[i]]);
          printf("\n    H = %.4f",M);
          S=100-S*8.0;
          printf("\n    Pack constant = %.4f\%",S);
          printf("\n   ----------\n");
          getch();
}
/*-------------------------------------------------------------
--------------*/
int main(){
   menu();
}
/*-------------------------------------------------------------
--------------*/
```

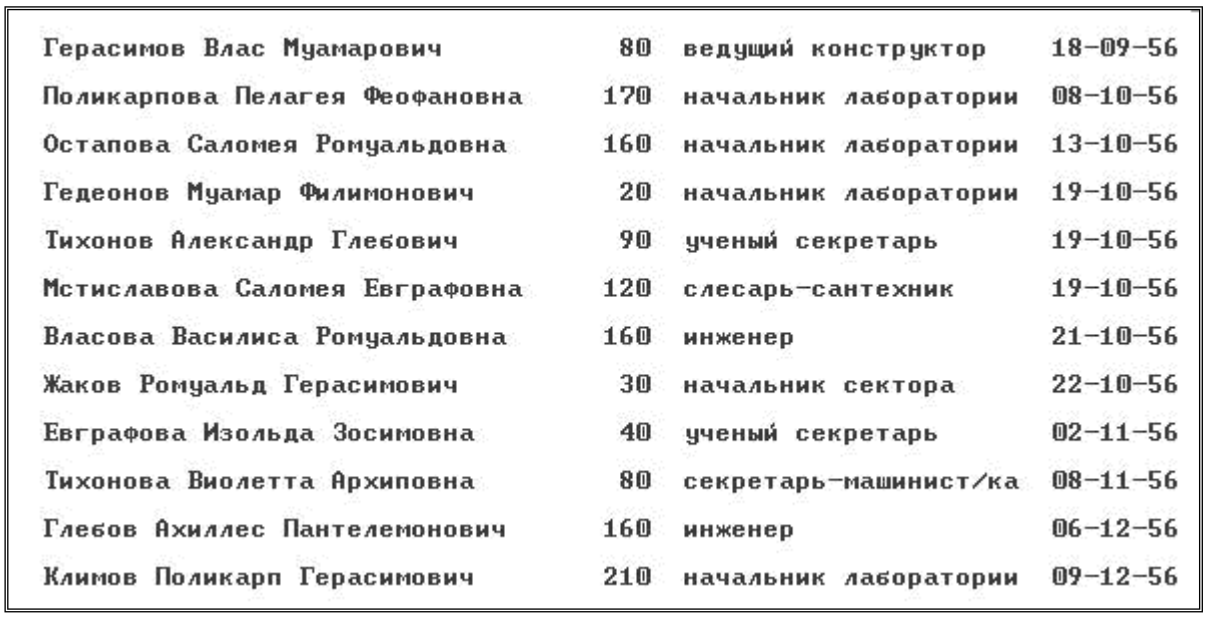
6. РЕЗУЛЬТАТЫ

```
153. Климов Александр Гедеонович
     230
     слесарь-сантехник
     03-07-32

154. Жаков Клим Герасимович
     240
     начальник сектора
     27-05-24

155. Патриков Демьян Хасанович
     150
     научный сотрудник
     09-01-71

156. Мстиславов Никодим Батырович
     220
     начальник отдела
     05-07-34

 ESC - back to menu_
```

Рисунок 5 - Несортированная база данных

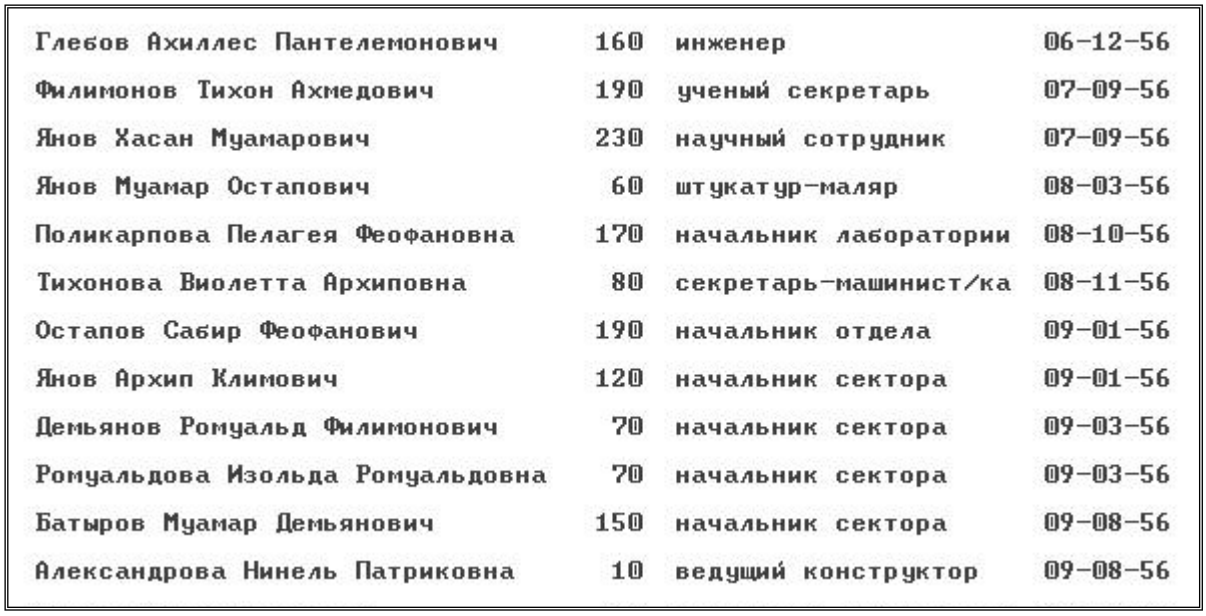
```
 29. Климов Ахмед Александрович
     80
     начальник сектора
     03-08-08

 30. Пантелемонова Матрена Демьяновна
     120
     инженер
     09-08-08

 31. Евграфов Герасим Герасимович
     100
     секретарь-машинист/ка
     10-08-08

 32. Евграфова Арабелла Герасимовна
     60
     слесарь-сантехник
     17-08-08

 ESC - back to menu
```

Рисунок 6 - Сортированная по дате рождения база данных



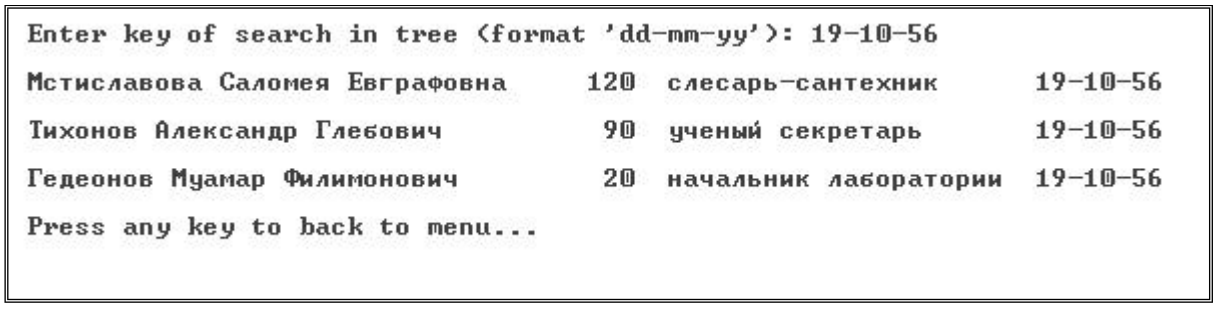
Герасимов Влас Муамарович	80	ведущий конструктор	18-09-56
Поликарпова Пелагея Феофановна	170	начальник лаборатории	08-10-56
Остапова Саломея Ромуальдовна	160	начальник лаборатории	13-10-56
Гедеонов Муамар Филимонович	20	начальник лаборатории	19-10-56
Тихонов Александр Глебович	90	ученый секретарь	19-10-56
Мстиславова Саломея Евграфовна	120	слесарь-сантехник	19-10-56
Власова Василиса Ромуальдовна	160	инженер	21-10-56
Жаков Ромуальд Герасимович	30	начальник сектора	22-10-56
Евграфова Изольда Зосимовна	40	ученый секретарь	02-11-56
Тихонова Виолетта Архиповна	80	секретарь-машинист/ка	08-11-56
Глебов Ахиллес Пантелемонович	160	инженер	06-12-56
Климов Поликарп Герасимович	210	начальник лаборатории	09-12-56

Рисунок 7 - Очередь из элементов, полученных в результате поиска(56 год рождения)



Глебов Ахиллес Пантелемонович	160	инженер	06-12-56
Филимонов Тихон Ахмедович	190	ученый секретарь	07-09-56
Янов Хасан Муамарович	230	научный сотрудник	07-09-56
Янов Муамар Остапович	60	штукатур-маляр	08-03-56
Поликарпова Пелагея Феофановна	170	начальник лаборатории	08-10-56
Тихонова Виолетта Архиповна	80	секретарь-машинист/ка	08-11-56
Остапов Сабир Феофанович	190	начальник отдела	09-01-56
Янов Архип Климович	120	начальник сектора	09-01-56
Демьянов Ромуальд Филимонович	70	начальник сектора	09-03-56
Ромуальдова Изольда Ромуальдовна	70	начальник сектора	09-03-56
Батыров Муамар Демьянович	150	начальник сектора	09-08-56
Александрова Нинель Патриковна	10	ведущий конструктор	09-08-56

Рисунок 8 - Дерево, ключ в дереве – дата рождения, как строка



```
Enter key of search in tree (format 'dd-mm-yy'): 19-10-56
Мстиславова Саломея Евграфовна     120  слесарь-сантехник       19-10-56
Тихонов Александр Глебович          90  ученый секретарь        19-10-56
Гедеонов Муамар Филимонович         20  начальник лаборатории   19-10-56
Press any key to back to menu...
```

Рисунок 9 - Поиск по дереву (элементы с одинаковым ключом)

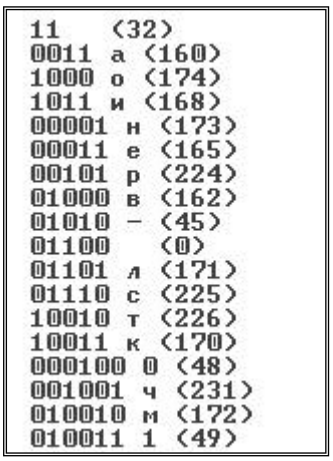


Рисунок 10 - Примеры кодовых слов

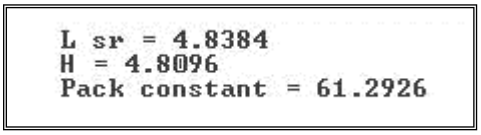


Рисунок 11 - Средняя длина, энтропия и коэффициент сжатия данных

7. ВЫВОДЫ

В ходе выполнения курсовой работы были выполнены все поставленные задачи и реализованы необходимые алгоритмы: сортировки, поиска, построения двоичного Б – дерева, 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 ($Lcp \leq H + 2$) и установлено, что при полученном значении Lcp и H размер кодируемой информации уменьшается примерно на 40%.

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.