ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра прикладной математики и информатики

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ К САМОСТОЯТЕЛЬНОЙ РАБОТЕ ПО КУРСУ "АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ "

(направление подготовки 6.050103 ”Программная инженерия”)

Донецк 2014

ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра прикладной математики и информатики

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ К САМОСТОЯТЕЛЬНОЙ РАБОТЕ ПО КУРСУ "АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ "

(направление подготовки 6.050103 ”Программная инженерия”)

Рассмотрено на заседании кафедры
Прикладной математики и информатики
Протокол № 1 от 25.09.14

Утверждено на заседании
учебно- издательского совета ДонНТУ

Донецк 2014

УДК 518.551071

Методические указания и задания к самостоятельной работе по курсу "Алгоритмы и структуры данных" (направление подготовки 6.050103 ”Программная инженерия”). Сост. Г.Г.Шалдырван, Н.С.Костюкова. – Донецк, 2014. – 20 с.

Излагаются необходимые сведения для выполнения индивидуального задания по разделу «Хеш- таблицы». Рассматриваются вопросы, связанные с теоретическими основами структуры данных, способами представления записей хеш- таблицы в оперативной памяти компьютера, методами обработки. Приведены описания алгоритмов и процедур обработки хеш-таблиц.

В приложении методических указаний приводится пример программы по теме индивидуального задания.

Составители: доц. Г.Г. Шалдырван
доц. Н.С. Костюкова

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

ХЕШ - ТАБЛИЦЫ

Цель работы: ознакомление с методами организации и использования переменных таблиц.

Задание

На языке Паскаль написать программу для работы с хеш-таблицей. Занесение и поиск осуществлять с помощью указанного метода разрешения конфликтов, используя заданные хеш-функцию и метод для вычисления вторичного индекса. Вариант задания выбрать в соответствии с номером по журналу.

Варианты заданий

Варианты заданий приведены в таблице 1.

Хеш- функции для вычисления первичного индекса обозначены буквами *а) - л),* см.стр. 65- 66.

Для обозначения методов разрешения конфликтов используются цифры:

1 - метод открытого перемешивания;

2 - использовать метод прямого связывания (с цепочками переполнения).

Методы определения вторичного индекса обозначены цифрами:

1) метод линейных проб с единичным шагом;

2) метод линейных проб с простым шагом *p*;

3) метод квадратичных проб.

Таблица 1

Вариант	Хеш- функция	Метод разрешения конфликтов	Метод определения вторичного индекса
1	А	1	1
2	Б	2	-
3	В	1	2
4	Г	1	1
5	Д	1	3
6	Е	2	-
7	Ж	1	1
8	З	2	-
9	И	1	2
10	К	2	-
11	Л	1	3
12	А	2	-

13	Б	1	1
14	В	2	-
15	Г	1	2
16	Д	2	-
17	Е	1	3
18	Ж	2	-
19	З	1	1
20	И	2	-
21	К	1	2
22	Л	2	-
23	А	1	2
24	Б	1	2
25	В	1	3
26	Г	1	3
27	Д	1	2
28	Е	1	2
29	Ж	1	2
30	З	1	3

Требования к выполнению индивидуального задания

1. Алгоритмы, реализующие вычисление **ХФ**, занесение записей в хеш – таблицу, поиск записей, нужно оформить в виде процедур (функций) с соответствующими формальными параметрами.

2. Занесение записей выполнять в интерактивном режиме, причем на экран должны выводиться первичный индекс, а также вся последовательность вторичных индексов и соответствующих ключей (или цепочка переполнения) в случае коллизий.

3. При поиске записей по задаваемому ключу на экран должны выводиться не только найденная запись, но также значение первичного индекса и всей последовательности вторичных индексов и соответствующих ключей (или элементов цепочки переполнения), если имели место коллизии.

4. Подготовить перечень ключей, вызывающих коллизии.

Контрольные вопросы

1. Определение хеш – таблицы; таблицы с прямым доступом.
2. Методы разрешения конфликтов.
3. Функция расстановки; требования, которым она должна удовлетворять.
4. Методы вычисления функции расстановки.
5. Достоинства и недостатки организации данных в виде таблиц с вычисляемым входом.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ РАБОТЫ

Определение хеш-таблицы

Если операции занесения записей в таблицу и операции поиска записей должны выполняться поочередно, то таблицу нужно организовать как *переменную*. Специальное название такой таблицы: *хеш – таблица* или *таблица с вычисляемым входом*.

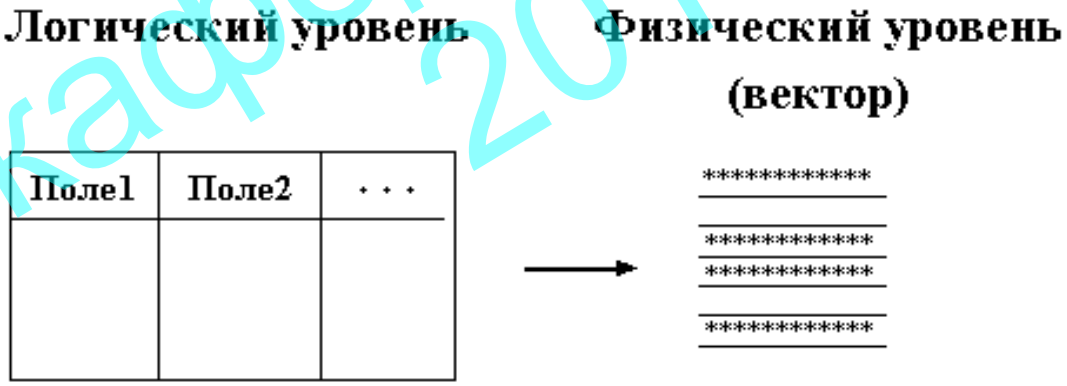
Определение:

Хеш-таблица (таблица с вычисляемым входом) – это такая структура, обращение к записям которой осуществляется **не по ключу записи непосредственно,** а **по номеру элемента** отображающего вектора, в который преобразуется значение ключа. Алгоритм преобразования ключа в номер (*индекс*) называется *хеш-функцией (функцией расстановки, функцией хеширования).*

Обозначим: *K* – ключ записи,
f(K) – хеш-функция,
i=f(K) – индекс (номер позиции, номер элемента вектора), по которому должна располагаться запись с ключом *K*.

Таким образом, хеш-функция (**ХФ**) обеспечивает для каждой табличной записи вычисление номера элемента вектора, на который отображается эта таблица.

Представление хеш-таблицы:



Хеш-таблица так же, как и постоянная таблица, отображается на вектор, но при этом в векторе могут оставаться "дырки", то есть не занятые записями позиции.

Внимание! В хеш-таблице значения ключевого поля должны быть только **уникальными**.

Если для хеш – таблицы выполняется условие однозначности:

$$K1 \neq K2,$$
$$f(K1) \neq f(K2)$$

(для разных ключей значения функции расстановки различны, и, таким образом, каждая запись претендует на отдельную позицию вектора), то таблицу называют таблицей с *прямым доступом*. Это частный случай хеш-таблицы.

Однако выполнение условия однозначности обеспечить трудно. Обычно возникает такая ситуация, когда для двух различных ключей *K1* и *K2* значения **ХФ** равны:

$$K1 \neq K2,$$
$$f(K1) = f(K2)$$

(то есть на некоторую позицию вектора претендуют несколько записей).

Это состояние называется *коллизией (конфликтной ситуацией, переполнением позиций отображающего вектора, наложением записей*).

Для разрешения конфликтов (устранения переполнений, разрешения коллизий) применяются следующие методы:

- метод открытого перемешивания,
- метод прямого связывания.

Метод открытого перемешивания

Пусть вектор, на который отображается хеш-таблица, рассчитан на *n* записей. Алгоритмы поиска в таблице и включения в нее записи с заданным ключом *K* предусматривают следующие действия:

1) вычисляется $i=f(K)$, что соответствует номеру позиции элемента вектора (то есть определяется *первичный индекс* записи, в дальнейшем обозначаемый как $i_{перв}$);

2)

а) при **включении** в таблицу: если *i* – я позиция вектора свободна, то запись с ключом *K* помещается в нее;

б) при **поиске** в таблице: если запись, находящаяся в позиции *i*, содержит ключ *K*, то поиск закончен;

3) если пункт 2) не выполняется, то вычисляется *вторичный индекс* ($i_{вт}$) и осуществляется возврат к пункту 2).

Для вычисления **вторичного индекса** используются:

1). *Метод линейных проб с единичным шагом*

$i_{вт} = (i_{перв}+1) \bmod n$ – первое значение $i_{вт}$;

$i_{вт} = (i_{вт} + 1) \bmod n$ – последующие значения.

Операция ***i mod n*** обозначает деление *i* по модулю *n*; она определяет остаток от деления *i* на *n*.

Согласно методу вектор просматривается с *i* - ой позиции до *n*-ой, а затем с первой позиции и далее. Недостаток единичного шага: записи группируются

вокруг первичных ключей, что приводит к более частому возникновению коллизий.

2). *Метод линейных проб с простым шагом p*

$i_{вт} = (i_{перв} + p) \bmod n$ – первое значение $i_{вт}$;

$i_{вт} = (i_{вт} + p) \bmod n$ – последующие значения;

p – простое число, ближайшее к *n* ($p < n$).

3). *Метод квадратичных проб*

$i_{вт} = (i_{перв}+j^2) \bmod n, \quad j=1,2,3\ldots \quad (j<n).$

Данный метод позволяет избежать группировки записей возле первичного индекса, но могут использоваться не все позиции отображающего вектора, то есть при занесении записи место для нее может не найтись, хотя на самом деле оно есть.

Замечания:

1. Если границы отображающего вектора указываются в интервале *[1..n]*, то операцию деления по модулю нужно взять в виде ***i mod n + 1***, чтобы индекс не получился нулевым.

2. Для операции ***i mod n*** рекомендуется брать вместо *n* простое число *p*, ближайшее к *n* (***i mod p***, $p<n$), так как при делении на простое число остатки реже совпадают между собой и , следовательно, реже возникают коллизии.

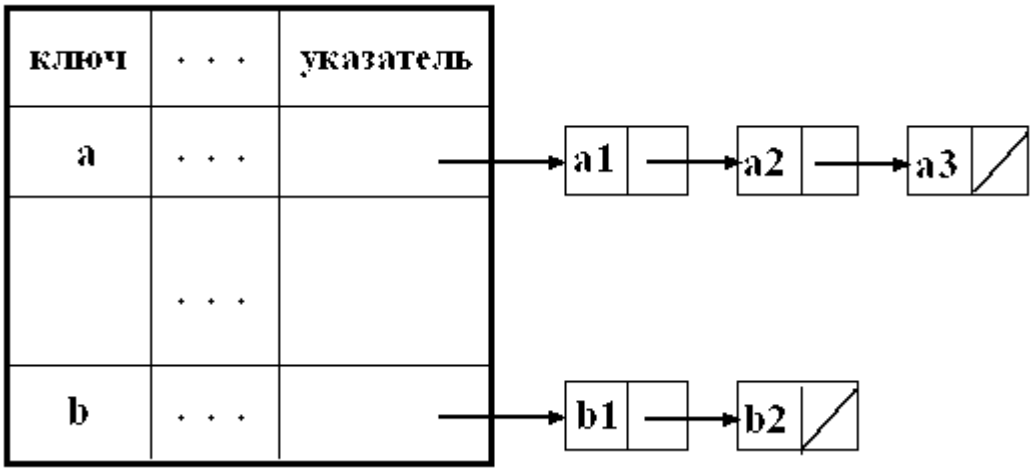
3. Для того чтобы зафиксировать момент переполнения вектора (случай, когда все его позиции будут заняты), нужно накапливать число занятых позиций и сравнивать его с *n*. При поиске также нужно контролировать количество просмотренных позиций: вторичный индекс должен вычисляться не более *(n-1)* раз.

Метод открытого перемешивания работает хорошо, пока таблица не слишком заполнена. Но постепенно процесс замедляется, так как начинают образовываться скопления записей вокруг первичных индексов. Поэтому при применении данного метода нужно увеличить размер памяти, отводимой под таблицу, примерно, на 20%. Средняя длина поиска в этом случае равна ≈3**..**5 (длина поиска определяется числом сравнений ключей, требующимся для нахождения определенной записи).

Метод прямого связывания

В методе открытого перемешивания переполняющие записи включаются в пустые позиции того же отображающего вектора. Обработка переполнений станет эффективнее, если переполняющие записи организовывать в виде списков, указатели на которые находятся в основной таблице.

Например, если **ХФ** равна порядковому номеру первой буквы ключа, то хеш–таблица может быть организована так:



Хеш-функция

При организации хеш-таблицы выбор **ХФ** в значительной степени зависит от особенностей **ключа.** Хорошая хеш-функция должна удовлетворять двум требованиям:

1) должна обеспечивать равномерное распределение записей по позициям отображающего вектора (то есть коллизии возникают как можно реже);
2) вычисление функции должно быть быстрым, так как время вычисления включается во время поиска.

Хеш-функция (функция расстановки) – это некоторый алгоритм, преобразующий значение ключа записи в номер элемента вектора.

Разработано достаточно много методов вычисления **ХФ** как для числовых, так и для символьных ключей. Например:

Для числового ключа:

а) *Метод середины квадратов*

Возводится в квадрат часть значения ключа или, если возможно, все значение ключа. Из середины полученного значения выделяются *n* десятичных цифр, которые и определяют индекс (номер позиции) записи. Значения индексов располагаются в интервале от 1 до (10^n-1). Значение *n* выбирается в зависимости от возможного количества записей в таблице (например, *n*=2, 3, …). Данный метод неплох, если ключи не содержат много подряд идущих нулей.

Например, пусть в качестве ключа записи взят код детали, состоящий из трех цифр. Если реальных кодов может быть всего около 100, то трехзначный код нужно преобразовать в двузначный индекс:

1) K=125; $f(125)=125^2=15625$ => 56; i=56

(запись с ключом 125 размещается в позиции вектора с номером 56);

2) K=271; $f(271)=271^2=73441$ => 34; i=34;
3) K=275; $f(275)=275^2=75625$ => 56; i=56 - коллизия!
4) K=525; $f(525)=525^2=275625$ => 56; i=56 - коллизия!

```
Function Med_Sq (key: longint; n: integer): integer;
{метод середины квадратов: преобразование трехзначного ключа в двузначный номер; размер таблицы [1. . n],  где n - двузначное число}
var sq:longint;
 begin
   sq:=sqr(key);
   {выделение из шестизначного числа двух центральных цифр}
   sq:=sq div 100;
   sq:=sq mod n +1;
   Med_Sq:=sq;
end;
```

б) *Метод свертки*

Ключ разбивается на несколько частей, которые затем суммируются таким образом, чтобы сформировать число в требуемом диапазоне. Например, если восьмизначный ключ нужно отобразить в трехзначный номер, то можно поступить так:

f(97434658) = 658 +434+97= 1189 =>189, i=189;
f(31269857) = 857+269+31 = 1157 =>157, i=157;
f(97658434) = 434 +658+97= 1189 =>189, i=189 - коллизия!

Здесь сложение выполняется по модулю 1000, так как нужно получить трехзначный номер. Если же реальное количество записей не превышает, например, 500, то нужно складывать по *mod 500*.

```
Function Convol(key:longint; n: integer):integer;
{метод свертки: преобразование восьмизначного ключа в трехзначный номер; размер  таблицы [1. . n], где n - трехзначное число}
var s:longint;
begin
  s:=0;
  while key>0 do
  begin
     s:=s+(key mod 1000);
     key:=key div 1000
  end;
  s:=s mod n+1;
  Convol:=s;
end;
```

в) *Метод деления по модулю*

В качестве значения функции берется остаток от деления кода ключа на *m*: *f(K) = K mod m* или *f(K) = K mod m + 1*.

В качестве *m* берется *n* (длина вектора) или некоторое простое число *p*, ближайшее к *n* *(p<n)*.

Для символьного ключа:

г) Первый и последний символы преобразуются в числа, суммируются, результат умножается на количество символов в ключе и делится по *mod m* (ключи могут быть разной длины). В качестве *m* берется *n* (длина вектора) или некоторое простое число *p*, ближайшее к *n (p<n)*. Реализация данного метода приведена в **Приложении Г** (функция *hfunc*).

д) Все символы ключа преобразуются последовательно в числа, суммируются, результат умножается на количество символов в ключе и делится по *mod m* (ключи могут быть разной длины). В качестве *m* берется *n* (длина вектора) или некоторое простое число *p*, ближайшее к *n (p<n)*.

е) Из середины ключа выделяются *k* символов, которые последовательно преобразуются в десятичные числа, суммируются и результат делится по *mod m* (все ключи имеют одинаковую длину). В качестве *m* берется *n* (длина вектора) или некоторое простое число *p*, ближайшее к *n (p<n)*.

ж) Второй и предпоследний символы преобразуются в числа, суммируются, результат умножается на количество символов в ключе и делится по *mod m* (ключи могут быть разной длины). В качестве *m* берется *n* (длина вектора) или некоторое простое число *p*, ближайшее к *n (p<n)*.

з) Символы ключа, стоящие на четных позициях, преобразуются в числа, суммируются, результат умножается на количество символов в ключе и делится по *mod m* (ключи могут быть разной длины). В качестве *m* берется *n* (длина вектора) или некоторое простое число *p*, ближайшее к *n (p<n)*.

и) Символы ключа, стоящие на нечетных позициях, преобразуются в числа, суммируются, результат умножается на количество символов в ключе и делится по *mod m* (ключи могут быть разной длины). В качестве *m* берется *n* (длина вектора) или некоторое простое число *p*, ближайшее к *n (p<n)*.

к) Первые *k* символов ключа последовательно преобразуются в десятичные числа, суммируются, результат делится по *mod m* (все ключи имеют одинаковую длину). В качестве *m* берется *n* (длина вектора) или некоторое простое число *p*, ближайшее к *n (p<n)*.

л) Второй и последний символы преобразуются в числа, суммируются,

результат умножается на количество символов в ключе и делится по *mod m* (ключи могут быть разной длины). В качестве *m* берется *n* (длина вектора) или некоторое простое число *p*, ближайшее к *n (p<n)*.

Замечания

1) Выбор метода вычисления **ХФ** зависит от особенностей ключей.
2) При использовании любой **ХФ** возможны коллизии.

Рассмотрим на примерах методы организации хеш – таблиц.

Пример 1. Для разрешения коллизий используем **метод открытого перемешивания:**

1) границы вектора заданы интервалом *[1..n];*
2) **ХФ** вычисляется по методу *г);*
3) для вычисления **вторичного индекса** выбран *метод линейных проб с простым шагом p* (*n*=100, *p*=97):

$$i_{вт} = (i_{вт} + p) \bmod n + 1$$

Занесение записей

Пусть в хеш-таблицу заносятся записи с ключами ‘abcd’, ‘acbd’, ‘akld’, ‘amtd’.

Для ключа ‘abcd’:

ord(‘a’) + ord(‘d’)=97+100=197;

$i_{перв}$ =(197*4) mod 100 +1 =**89**.

Для ключа ‘acbd’:

$i_{перв}$=89; $i_{вт}$= (89+97) mod 100 +1=**87**.

Для ключа ‘akld’:

$i_{перв}$=89; $i_{вт}$=87;

$i_{вт}$=(87+97) mod 100 +1=**85**.

Для ключа ‘amtd’:

$i_{перв}$=89; $i_{вт}$=87; $i_{вт}$=85;

$i_{вт}$=(85+97) mod 100 +1=**83**.

Значения функции расстановки для заданных ключей совпадают, следовательно, возникает состояние коллизий.

номер позиции	key	info
1	. . .	. . .
. . .		
83	amtd	. . .

85	akld	. . .
87	acbd	. . .
89	abcd	. . .
. . .		
100	. . .	. . .

Поиск записей

1). Найти запись с ключом ‘amtd’:

$i_{перв}$=89; K=‘abcd’;
$i_{вт}$= 87; K=‘acbd’;
$i_{вт}$= 85; K=‘akld’;
$i_{вт}$= 83; K=‘**amtd**’ – найдено!

2). Найти запись с ключом ‘acba’:

ord(‘a’) + ord(‘a’)=97+97=194;
$i_{перв}$ =(194*4) mod 100 +1 =77 →
позиция пуста (запись не найдена).

Пример 2. Для разрешения коллизий используем **метод прямого связывания** (**ХФ** вычисляем по методу *г*).

Пусть в хеш-таблицу заносятся записи с теми же ключами, что и в примере 1 (‘abcd’, ‘acbd’, ‘akld’, ‘amtd’). Переполняющие записи организуем в виде списковой структуры. Включение записей в цепочки выполним по принципу стека (более быстрый способ).

Занесение записей

Для ключа ‘abcd’:
ord(‘a’) + ord(‘d’)=97+100=197;
i =(197*4) mod 100 +1 =89.

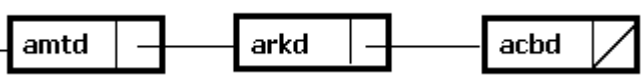
номер позиции	key	info	ukz
1			nil
	. . .		
89	abcd		nil
	. . .		

Для ключа ‘acbd’: i=89 → позиция занята, помещаем эту запись в цепочку переполнения:

Записи с ключами ‘akld’ и ‘amtd’ также помещаются в цепочку переполнения.

Окончательный результат:

номер позиции	key	info	ukz
1			nil
. . .			
89	abcd	. . .	
. . .			



Поиск записей

Найти запись с ключом ‘acbd’. Для ключа ‘acbd’: $i=89 \rightarrow$ позиция занята, ищем эту запись в цепочке переполнения (поиск в списке).

Программа, в которой реализован метод прямого связывания, приведена в **Приложении А**.

Достоинства хеш-таблиц

1) возможность чередования операций занесения и поиска;
2) простота алгоритмов работы с хеш-таблицами;
3) время поиска значительно меньше, чем при бинарном поиске.

Недостатки хеш-таблиц

1) подбираемые **ХФ** эффективны лишь в среднем, то есть не гарантировано равномерное распределение записей по позициям отображающего вектора (возможны коллизии);
2) трудно рассчитать объем памяти, необходимый для хеш-таблицы, так как заранее неизвестно число записей;
3) выполняется поиск только на сравнение ключей; поиск по близости и по интервалу невозможен.

Приложение А

Пример программы, реализующей операции с хеш-таблицей

```
Program hash;
{занесение и поиск записей в хеш-таблице; для разрешения коллизий
используется метод прямого  связывания}
Uses Crt;
Const n=100;
Type u=^rec;
        rec=record    {структура записи хеш-таблицы}
                key:string;     {ключ записи}
                inf :integer;
                 ukz:u; {указатель на цепочку переполнения}
              end;
        htab=Array[1..n] of rec;
Var   T: htab;     {основная таблица}
       ch: char;
        i: integer;
        s: string;

Function hfunc(s:string; n: integer):integer;
{хеш-функция использует  первый и последний символы  ключа}
Var len:integer;
Begin
   len:=length(s);   {длина ключа}
   hfunc:=((ord(s[1])+ord(s[len]))*len) mod n+1;
End;

Procedure Add_rec (Var T:htab);
 {занесение записи в хеш-таблицу}
Var tmp: u;
          i: integer;
          r: rec;
Begin
   {ввод добавляемой записи}
    WriteLn ('Введите ключ добавляемой записи:');
     ReadLn (r.key);

    WriteLn ('Введите информационное поле');
    ReadLn (r.inf);
    r.ukz:=nil;
```

```
    i:=hfunc(r.key, n);     {вычисление хеш-функции }
   {попытка занесения записи в таблицу}
   If T[i].key=' ' then
      begin
         T[i]:=r;
         WriteLn('Запись помещена в позицию ', i);
      end
   else
     begin
        New (tmp);
        tmp^:=r;
        tmp^.ukz:=T[i].ukz;
        T[i].ukz:=tmp;
        WriteLn ('Позиция ',i,' занята');
        WriteLn ('Запись помещена в цепочку переполнения');
     end;
End;


Procedure Find_rec (Var T:htab; kl:string);
{процедура поиска в хеш- таблице записи с ключом kl}
Var   i:integer;
       p:u;
Begin
   i:=hfunc(kl, n);     {вычисление хеш-функции }
   WriteLn ('Поиск в таблице...');
   If T[i].key=kl then
     begin
        WriteLn ('Запись найдена в позиции ', i);
        WriteLn ('Значение информ. поля ',  T[i].inf);
        WriteLn ('Нажмите любую клавишу');
        ReadKey;
        exit;
     end;


   If T[i].key=' ' then
     begin
        WriteLn ('Записи в таблице нет');
        WriteLn ('Нажмите любую клавишу');
```

```
        ReadKey;
        Exit;
      end;
    {поиск в цепочке переполнения}
    WriteLn (i, T[i].key);
    WriteLn ('Поиск в цепочке переполнения...');
     p:=T[i].ukz;
     While (p<>nil) and (p^.key<>kl) do
        begin
          Write (p^.key,' ');
           p:=p^.ukz;
        end;
     WriteLn;
     If p<>nil then      {поиск успешен}
        begin
           WriteLn ('Запись найдена');
           WriteLn ('Значение информац. поля = ', p^.inf);
        end
     else
        WriteLn ('Запись не найдена');
        WriteLn ('Нажмите любую клавишу');
         ReadKey;
End;

Begin
  {очистка ключевого поля }
   For i:=1 to n do
      T[i].key:=' ';
   Repeat
        Clrscr;
        WriteLn ('1 - Занесение записи в хеш-таблицу');
        WriteLn ('2 - Поиск записи в хеш-таблице');
        WriteLn('-----------------------------------------');
        WriteLn ('0 - Выход');
        WriteLn;

        Write ('Ваш выбор: ');
       ch:=Readkey;
       WriteLn (ch);
       WriteLn;
```

```
        Case ch of
            '1': begin
                    Add_rec(T);
                    WriteLn ('Нажмите любую клавишу');
                    ReadKey
                 end;
            '2': begin
                    WriteLn ('Введите ключ искомой записи');
                    ReadLn (s);
                    Find_rec(T,s)
                 end;
        end;
  until ch='0';
End.
```