Глава 6. МЕТОДЫ ПОИСКА

Предметы (объекты), составляющие множество, называются его элементами. Элемент множества будет называться ключом, и обозначаться латинской буквой k_i с индексом, указывающим номер элемента.

Алгоритмы поиска можно разбить на несколько групп (рис. 6.1).

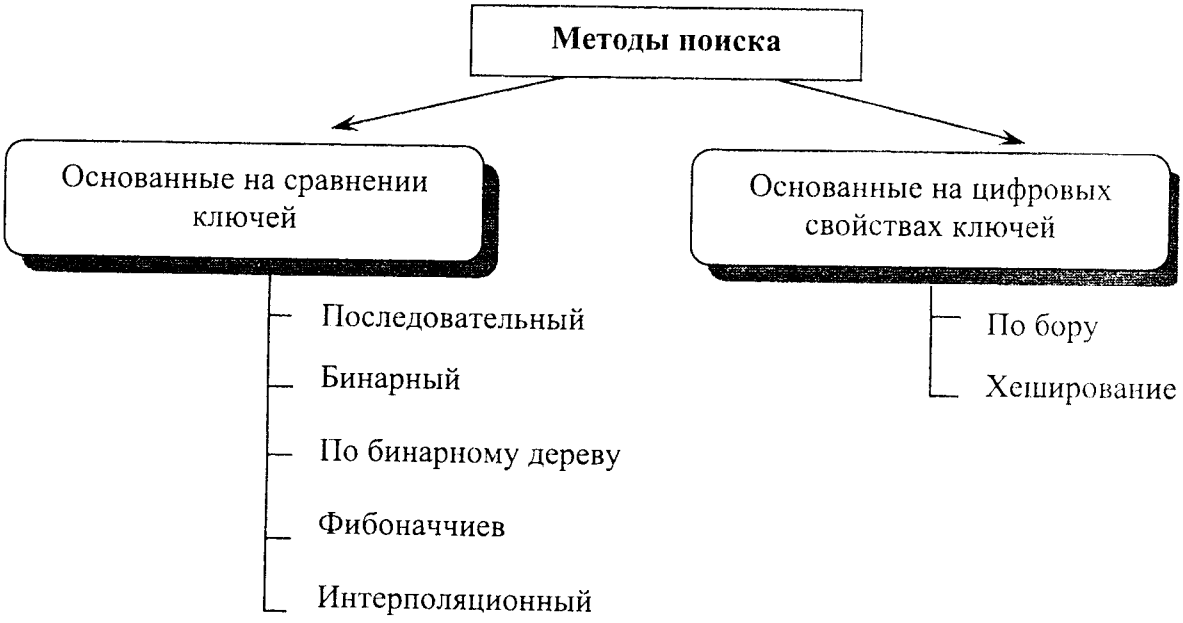

Рис. 6.1. Методы поиска

Задача поиска. Пусть задано множество ключей $\{k_1, k_2, k_3,..., k_n\}$. Необходимо отыскать во множестве ключ k_i. Поиск может быть завершен в двух случаях: ключ во множестве отсутствует либо ключ во множестве найден.

6.1. Последовательный поиск

В последовательном поиске исходное множество не упорядочено, т.е. имеется произвольный набор ключей $\{k_1, k_2, k_3,..., k_n\}$. Метод состоит в том, что отыскиваемый ключ k_i последовательно сравнивается со всеми элементами множества. При этом поиск заканчивается досрочно, если ключ найден.

Алгоритм поиска сводится к последовательности шагов:

Шаг 1. [Начальная установка.] Установить $i := 1$;

Шаг 2. [Сравнение.] Если $k = k_i$, алгоритм заканчивается удачно;

Шаг 3. [Продвижение.] Увеличить i на 1;

Шаг 4. [Конец цикла?] Если $i \le N$, то вернуться к шагу 2. В противном случае алгоритм заканчивается неудачно.

6.2. Бинарный поиск

В бинарном поиске исходящее множество должно быть упорядочено по возрастанию. Иными словами, каждый последующий ключ больше предыдущего, т.е. $\{k_1 \le k_2 \le k_3 \le k_4, \ldots, k_{n-1} \le k_n\}$. Отыскиваемый ключ сравнивается с центральным элементом множества, и если он меньше центрального, то поиск продолжается в левом подмножестве, в противном случае — в правом подмножестве.

Медианой последовательности из n элементов считается элемент, значение которого меньше (или равно) половины n элементов и больше (или равно) другой половины. Задачу поиска медианы принято связывать с сортировкой, так как медиану всегда можно найти следующим способом: отсортировать n элементов и затем выбрать средний элемент.

В алгоритме К. Хоара для отыскания медианы выполняется операция разделения, используемая при быстрой сортировке, с $L = 1$, $R = n$ и с $a[k]$, выбранным в качестве разделяющего значения x. Получаются значения индексов i и j такие, что:

1) разделяющее значение x было слишком мало; в результате граница между двумя частями ниже искомого значения k. Процесс разделения следует повторить для элементов $a[i], \ldots, a[R]$ (рис. 6.2);

2) выбранная граница x была слишком велика. Операцию разбиения следует повторить на подмассиве $a[L], \ldots, a[j]$ (рис. 6.3);

3) значение k лежит в интервале $j < k < i$: элемент $a[k]$ разделяет массив в заданной пропорции и, следовательно, является искомым (рис. 6.4).

Процесс разбиения повторяется до появления последнего случая. Центральный элемент находится по формуле $N_{\text{эл-та}} = [n/2] + 1$ (либо $N_{\text{эл-та}} = [n/2]$), где квадратные скобки обозначают, что от де-

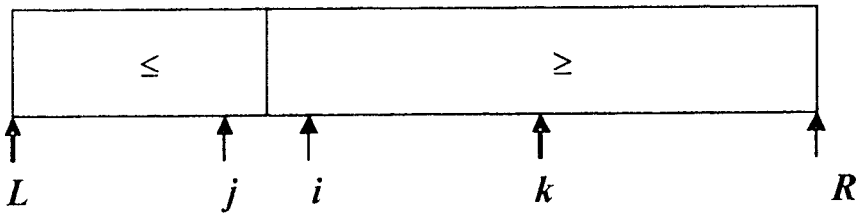

Рис. 6.2. Смещение границы влево

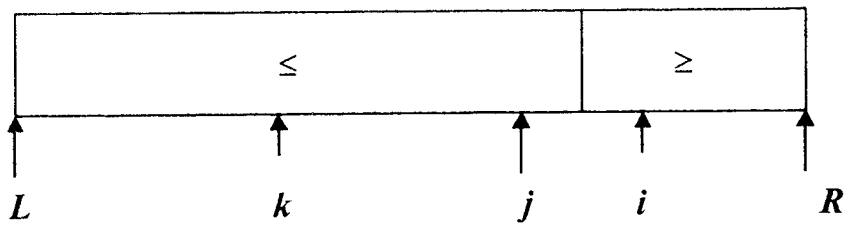

Рис. 6.3. Смещение границы вправо

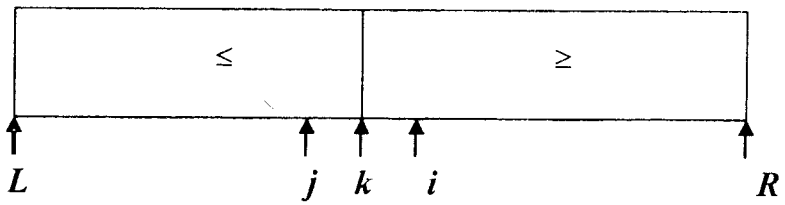

Рис. 6.4. Граница определена точно

ления берется только целая часть (всегда округляется в меньшую сторону). В методе бинарного поиска анализируются только центральные элементы подмножеств.

Пример 1. Во множестве элементов отыскать ключ, равный 653. В квадратных скобках выделены множества анализируемых элементов. Центральные элементы подмножеств подчеркнуты. Поиск ключа $K = 653$ осуществляется за четыре шага:

[061 087 154 170 275 426 503 509 512 612 653 677 703 765 897 908]
061 087 154 170 275 426 503 509 **[512 612 653 677 703 765 897 908]**
061 087 154 170 275 426 503 509 **[512 612 653]** 677 703 765 897 908
061 087 154 170 275 426 503 509 512 612 **[653]** 677 703 765 897 908

Пример 2. Дано упорядоченное множество элементов

{7, 8, 12, 16, 18, 20, 30, 38, 49, 50, 54, 60, 61, 69, 75,
79, 80, 81, 95, 101, 123, 198}.

Найти в этом множестве ключ $K = 61$.

Шаг 1. $N_{\text{эл-та}} = [n/2] + 1 = [22/2] + 1 = 12.$

{7, 8, 12, 16, 18, 20, 30, 38, 49, 50, 54, **60**, 61, 69,
75, 79, 80, 81, 95, 101, 123, 198}.

$K \vee k_{12}$. Значок «$\vee$» обозначает сравнение элементов (чисел, значений). $61 > 60$. Дальнейший поиск ведется в правом подмножестве.

Шаг 2. $N_{\text{эл-та}} = [n/2] + 1 = [10/2] + 1 = 6.$

{61, 69, 75, 79, 80, **81**, 95, 101, 123, 198}. $K \vee k_{18}$; $61 < 81$.

Дальнейший поиск ведется в левом подмножестве.

Шаг 3. $N_{\text{эл-та}} = [n/2] + 1 = [5/2] + 1 = 3.$

{61, 69, **75**, 79, 80}. $K \vee k_{15}$; $61 < 75$.

Дальнейший поиск ведется в левом подмножестве.

Шаг 4. $N_{\text{эл-та}} = [n/2] + 1 = [2/2] + 1 = 2.$

{61, **69**}. $K \vee k_{14}$; $61 < 69$.

Дальнейший поиск ведется в левом подмножестве.

Шаг 5. {**61**}. $K \vee k_{13}$; 61=61. Вывод: искомый ключ найден под номером 13.

6.3. Фибоначчиев поиск

В этом поиске анализируются элементы, находящиеся в позициях, равных числам Фибоначчи. Числа Фибоначчи получаются по следующему правилу: последующее число равно сумме двух предыдущих чисел, например,

$$\{1, 1, 2, 3, 5, 8, 13, 21, 34, 55,...\}.$$

Поиск продолжается до тех пор, пока не найден интервал между двумя ключами, где может располагаться отыскиваемый ключ. Фибоначчиев поиск предназначается для отыскания аргумента K среди расположенных в порядке возрастания ключей $K_1 < K_2 < ... < K_n$. Для удобства описания предполагается, что $n+1$ есть число Фибоначчи F_{k+1}. Подходящей начальной установкой данный метод можно сделать пригодным для любого n.

Шаг 1. [Начальная установка.] Установить $i := F_k$, $p := F_{k-1}$, $q := F_{k-2}$. (В алгоритме p и q обозначают последовательные числа Фибоначчи.)

Шаг 2. [Сравнение.] Если $K < K_i$, то перейти на шаг 3; если $K > K_i$, то перейти на шаг 4; если $K = K_i$, алгоритм заканчивается удачно.

Шаг 3. [Уменьшение i.] Если $q = 0$, алгоритм заканчивается неудачно. Если $q \neq 0$, то установить $i := i - q$, заменить (p, q) на $(q, p - q)$ и вернуться на шаг 2.

Шаг 4. [Увеличение i.] Если $p = 1$, то алгоритм заканчивается неудачно. Если $p \neq 1$, установить $i := i + q$, $p := p - q$, $q := q - p$ и вернуться на шаг 2.

Пример. Дано исходное множество ключей

$\{3, 5, 8, 9, 11, 14, 15, 19, 21, 22, 28, 33, 35, 37, 42, 45, 48, 52\}$.

Пусть отыскиваемый ключ равен 42, т.е. $K = 42$. Последовательное сравнение отыскиваемого ключа будет проводиться с элементами исходного множества, расположенными в позициях, равных числам Фибоначчи: $\{1, 2, 3, 5, 8, 13, 21,...\}$.

Шаг 1. $K \vee k_1$, $42 > 3$ => отыскиваемый ключ сравнивается с ключом, стоящим в позиции, равной числу Фибоначчи.

Шаг 2. $K \vee k_2$, $42 > 5$ => сравнение продолжается с ключом, стоящим в позиции, равной следующему числу Фибоначчи.

Шаг 3. $K \vee k_3$, $42 > 8$ => сравнение продолжается.

Шаг 4. $K \vee k_5$, $42 > 11$ => сравнение продолжается.

Шаг 5. $K \vee k_8$, $42 > 19$ => сравнение продолжается.

Шаг 6. $K \vee k_{13}$, $42 > 35$ => сравнение продолжается.

Шаг 7. $K \vee k_{18}$, $42 < 52$ => найден интервал, в котором находится отыскиваемый ключ, т.е. он может находиться в исходном множестве

между 13-й и 18-й позициями, т.е. {35, 37, 42, 45, 48, 52}. В найденном интервале поиск вновь ведется в позициях, равных числам Фибоначчи.

6.4. Интерполяционный поиск

Исходное множество должно быть упорядочено по возрастанию весов. Первоначальное сравнение осуществляется на расстоянии шага d, который определяется по формуле

$$d = \left[\frac{(j-i)(K-K_i)}{K_j - K_i} \right],$$

где i — номер первого рассматриваемого элемента; j — номер последнего рассматриваемого элемента; K — отыскиваемый ключ; K_i, K_j — значения ключей в i и j позициях; [ ] — целая часть от числа.

Идея метода заключается в следующем: шаг d меняется после каждого этапа по формуле, приведенной выше (рис. 6.5).

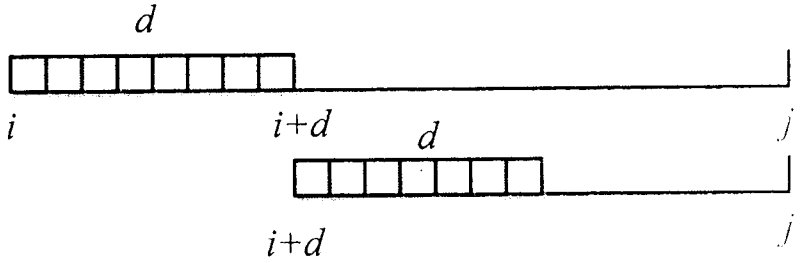

Рис. 6.5. Схема интерполяционного поиска

Алгоритм заканчивает работу при $d = 0$, при этом анализируются соседние элементы, после чего принимается окончательное решение о результатах поиска. Этот метод прекрасно работает, если исходное множество представляет собой арифметическую прогрессию или множество, приближенное к ней.

Пример 1. Дано множество ключей

{2, 9, 10, 12, 20, 24, 28, 30, 37, 40, 45, 50, 51, 60, 65, 70, 74, 76}.

Пусть искомый ключ $K = 70$. Определяется шаг d для исходного множества ключей ($i = 1$; $j = 18$): $d = [(18-1)(70-2)/(76-2)] = 15$. Сравнивается ключ, стоящий под 16-м порядковым номером в данном множестве с искомым ключом: $k_{16} \vee K$; $70 = 70$; ключ найден.

Пример 2. Дано множество ключей.

{4, 5, 10, 23, 24, 30, 47, 50, 59, 60, 64, 65, 77, 90, 95, 98, 102}.

Требуется отыскать ключ $K = 90$.

Шаг 1. Определяется шаг d для исходного множества ключей ($i = 1$; $j = 17$):

$$d = [(17-1)(90-4)/(102-4)] = 14.$$

Сравнивается ключ, стоящий под 15-м порядковым номером, с отыскиваемым ключом: $k_{15} \vee K$; $95 > 90$, следовательно, сужается область поиска

$$\{4, 5, 10, 23, 24, 30, 47, 50, 59, 60, 64, 65, 77, 90, 95\}.$$

Шаг 2. Определяется шаг d для множества ключей ($i = 1; j = 15$):

$$d = [(15-1)(90-4)/(95-4)] = 13.$$

Сравниваем ключ, стоящий под 14-м порядковым номером, с отыскиваемым ключом: $k_{14} \vee K$; $k_{14} = K$; $90 = 90$; ключ найден.

6.5. Поиск по бинарному дереву

Дерево двоичного поиска для множества чисел S — это размеченное бинарное дерево, каждой вершине которого сопоставлено число из множества S, причем все пометки удовлетворяют следующему простому правилу: «если больше — направо, если меньше — налево».

Например, для набора чисел {7, 3, 5, 2, 8, 1, 6, 10, 9, 4, 11} получится бинарное дерево (рис. 6.6). Для того чтобы правильно учесть повторения элементов, можно ввести дополнительное поле, которое будет хранить число вхождений для каждого элемента.

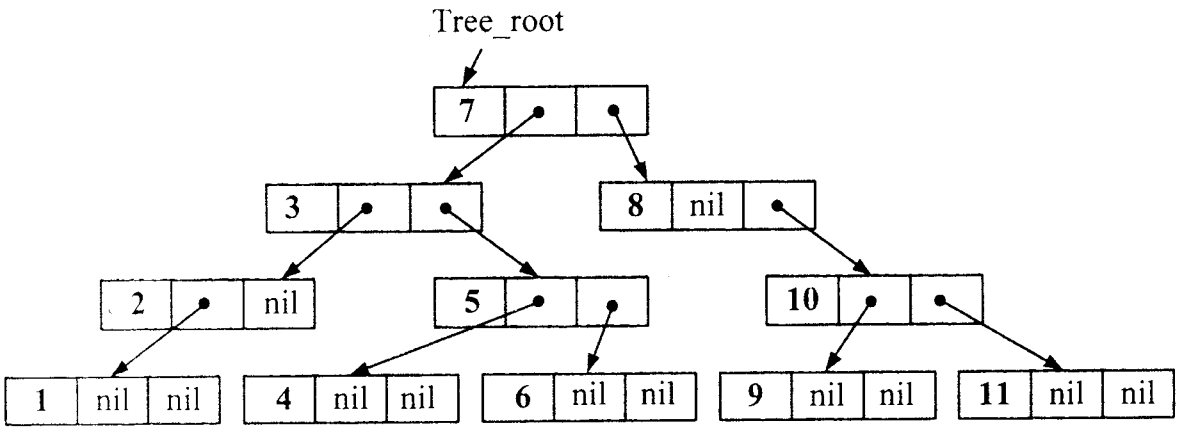

Рис. 6.6. Дерево двоичного поиска

Бинарное дерево, соответствующее бинарному поиску среди n записей, можно построить следующим образом: при $n = 0$ дерево сводится к узлу 0. В противном случае корневым узлом является $[n/2]$, левое поддерево соответствует бинарному дереву с $[n/2]-1$ узлами, а правое — дереву с $[n/2]$ узлами и числами в узлах, увеличенными на $[n/2]$ (рис. 6.7).

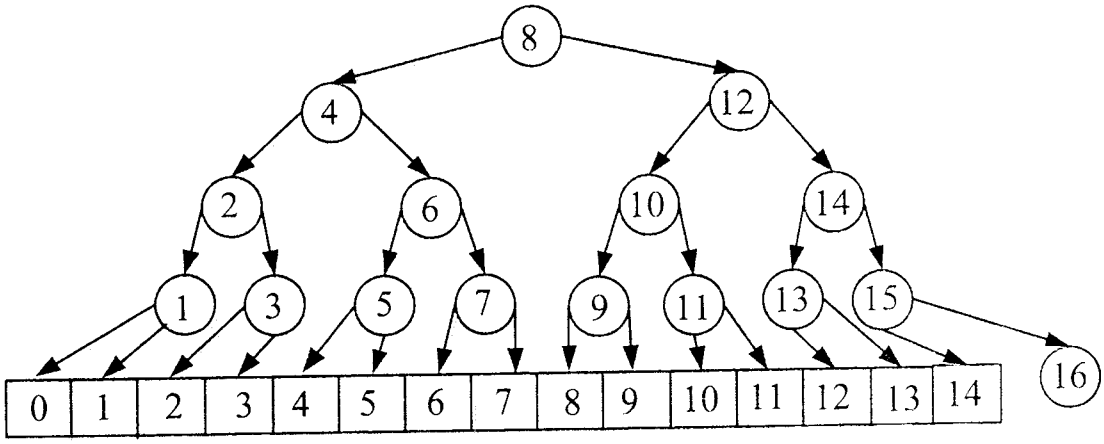

Рис. 6.7. Бинарное дерево, соответствующее бинарному поиску

Число узлов, порожденных отдельным узлом (число поддеревьев данного корня), называется его *степенью*. Узел с нулевой степенью называют листом, или концевым узлом. Максимальное значение степени всех узлов данного дерева называется *степенью дерева*.

Алгоритм поиска по бинарному дереву: вначале аргумент поиска сравнивается с ключом, находящимся в корне. Если аргумент совпадает с ключом, поиск закончен, если же не совпадает, то в случае, когда аргумент оказывается меньше ключа, поиск продолжается в левом поддереве, а в случае, когда больше ключа, — в правом поддереве. Увеличив уровень на единицу, повторяют сравнение, считая текущий узел корнем. Использование структуры бинарного дерева позволяет быстро вставлять и удалять записи и вести эффективный поиск по таблице. Такая гибкость достигается добавлением в каждую запись двух полей для хранения ссылок.

Пусть дано бинарное дерево поиска (рис. 6.8).

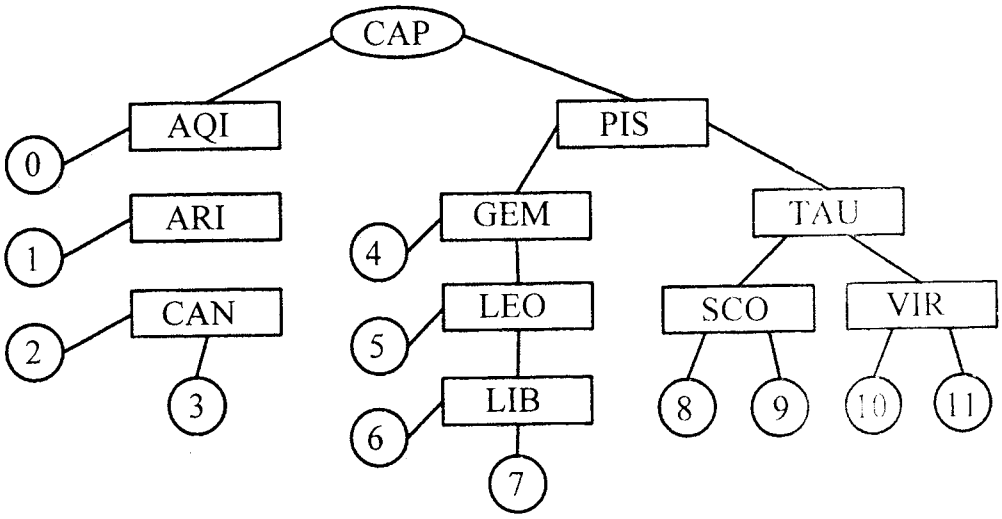

Рис. 6.8. Бинарное дерево

Требуется по бинарному дереву отыскать ключ SAG. При просмотре от корня дерева видно, что по первой букве латинского алфавита название SAG больше, чем CAP. Следовательно, дальнейший поиск будет осуществляться в правой ветви. Это слово больше, чем PIS, — снова продвижение вправо; оно меньше, чем TAU, — продвижение влево; оно меньше, чем SCO, и отбирается узел 8. Таким образом, название SAG должно находиться в узле 8. При этом узлы дерева имеют структуру, показанную в табл. 6.1.

Таблица 6.1

Структура узлов дерева

<table>
<tr><td rowspan="2">Ключ</td><td>Информационная часть</td><td>Указатель на левое поддерево</td><td>Указатель на правое поддерево</td></tr>
<tr><td>(может отсутствовать)</td><td rowspan="2">LLINK</td><td rowspan="2">RLINK</td></tr>
<tr><td colspan="2">KEY</td></tr>
</table>

Пример. Пусть дано исходное множество ключей

{2, 4, 5, 6, 7, 9, 12, 14, 18, 21, 24, 25, 27, 30, 32, 33, 34, 37, 39}.

Оно должно быть упорядочено по возрастанию. Переходим от линейного списка к построению бинарного дерева поиска (рис. 6.9), где корнем дерева является центральный элемент множества.

$N_{\text{ц.эл}} = [n/2] + 1$, где n — число элементов множества.

Вершиной по левой ветке является центральный элемент левого подмножества, а по правой — правого подмножества и т.д. Отыскиваемый ключ $K = 24$. Исходное множество имеет 19 элементов. $N_{\text{ц.эл}} = [19/2] + 1 = 10$. Поиск ключа $K = 24$ по бинарному дереву от корня до листьев показан на рис. 6.9.

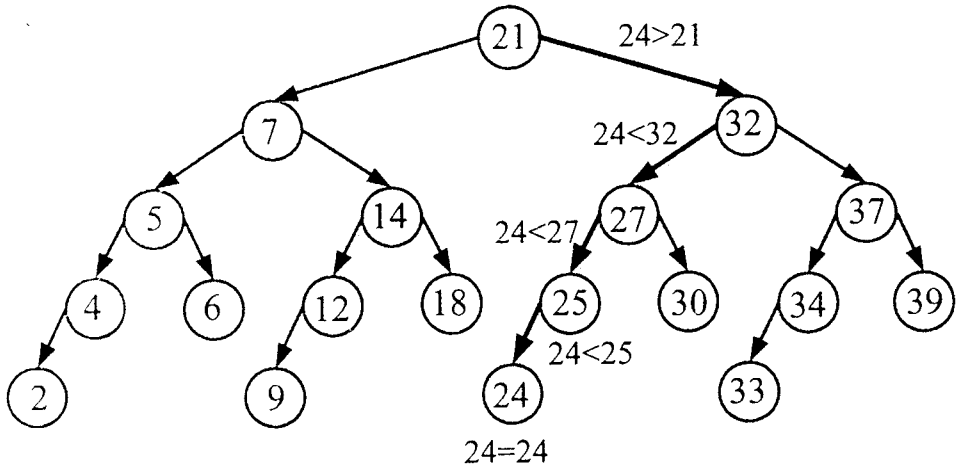

Рис. 6.9. Бинарное дерево поиска

Преобразование произвольного дерева в бинарное. Упорядоченные деревья степени 2 называются *бинарными деревьями*. Бинарное дерево состоит из конечного множества элементов (узлов), каждый из которых либо пуст, либо состоит из корня (узла), связанного с двумя различными бинарными деревьями, называемыми *левым* и *правым* поддеревьями корня. Деревья, у которых $d > 2$, называются *d-арными*.

Преобразование произвольного дерева с упорядоченными узлами в бинарное дерево сводится к следующим действиям. Сначала в каждом узле исходного дерева вычеркиваются все ветви, кроме самых левых ветвей. После этого в получившемся графе соединяют горизонтальными ветвями те узлы одного уровня, которые являются «братьями» в исходном дереве. (Заметим, что если несколько узлов имеют общего предка, то такие узлы иногда называют «братьями».) В получившемся таким образом дереве левым потомком каждого узла x считается непосредственно находящийся под ним узел (если он есть), а в качестве правого потомка — соседний справа «брат» для x, если таковой имеется.

На рис. 6.10 показаны этапы такого преобразования некоторого d-арного дерева в бинарное.

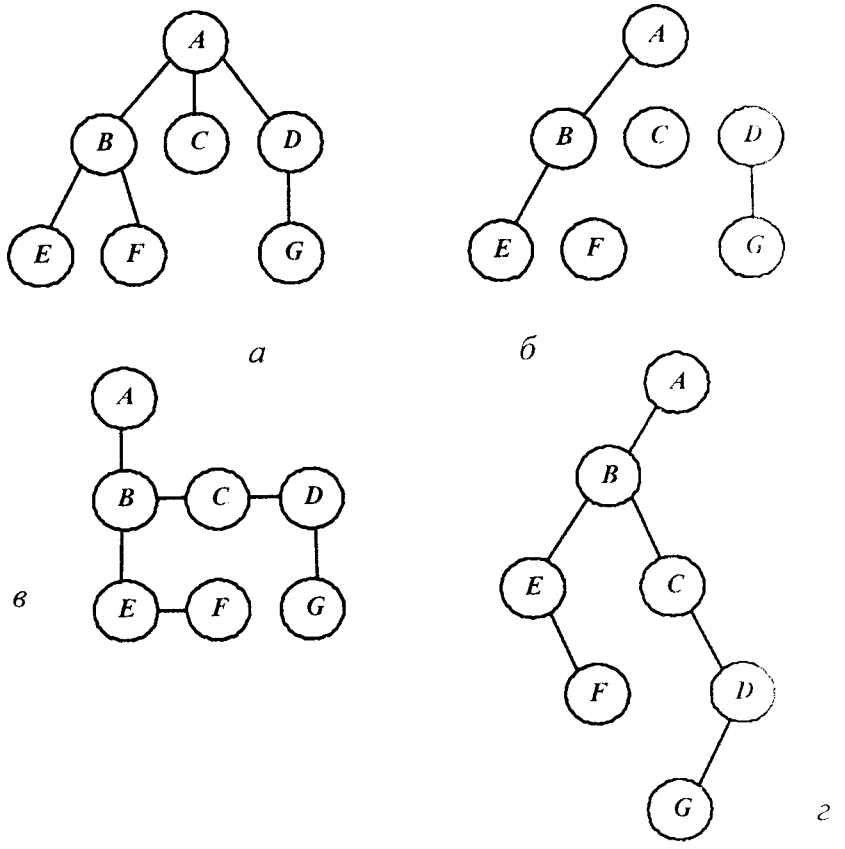

Рис. 6.10. Этапы преобразования d-арного дерева в бинарное

Переход от произвольного дерева к его бинарному эквиваленту не только облегчает анализ логической структуры, но и упрощает машинное представление, т.е. физическую структуру дерева.

Сбалансированное бинарное дерево. Бинарное дерево называется сбалансированным (*B-balanced*), если высота левого поддерева каждого узла отличается от высоты правого не более чем на единицу (рис. 6.11).

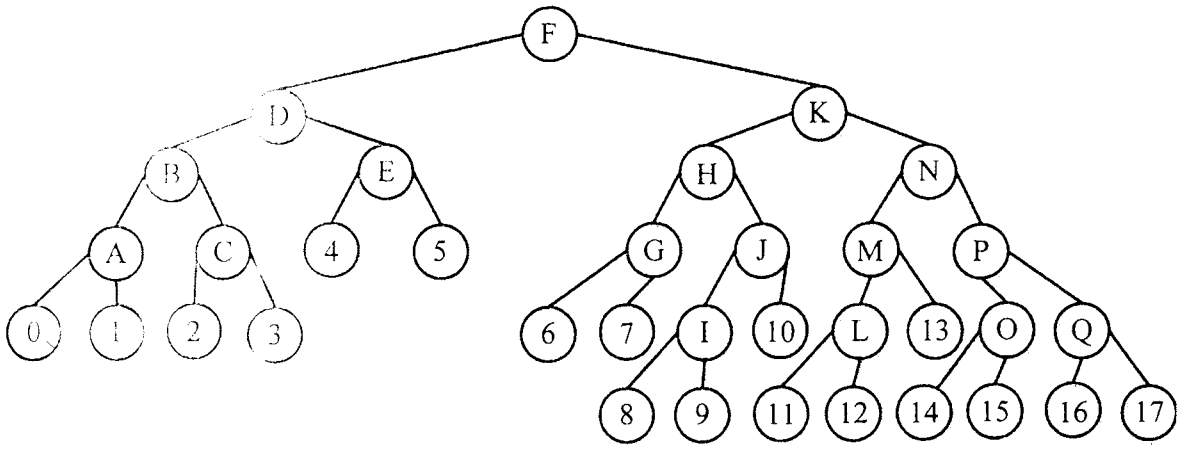

Рис. 6.11. Пример сбалансированного дерева

Сбалансированные бинарные деревья занимают промежуточное положение между оптимальными бинарными деревьями (все внешние узлы которых расположены на двух смежных уровнях) и произвольными бинарными деревьями. Рассмотрим структуру узлов сбалансированного бинарного дерева (табл. 6.2).

Таблица 6.2

Структура узлов сбалансированного дерева

Ключ	Указатель на левое поддерево	Указатель на правое поддерево	Показатель сбалансированности узла
KEY	LLINK	RLINK	B

B — показатель сбалансированности узла, т.е. разность высот правого и левого поддерева (*B* = +1; ∅; −1). При восстановлении баланса дерева по высоте учитывается показатель *B* (рис. 6.12).

Символы +1, ∅, −1 указывают, что левое поддерево выше правого, поддеревья равны по высоте, правое поддерево выше левого.

6.6. Поиск по бору

Особую группу методов поиска образует представление ключей в виде последовательности цифр и букв. Рассмотрим, например, име-

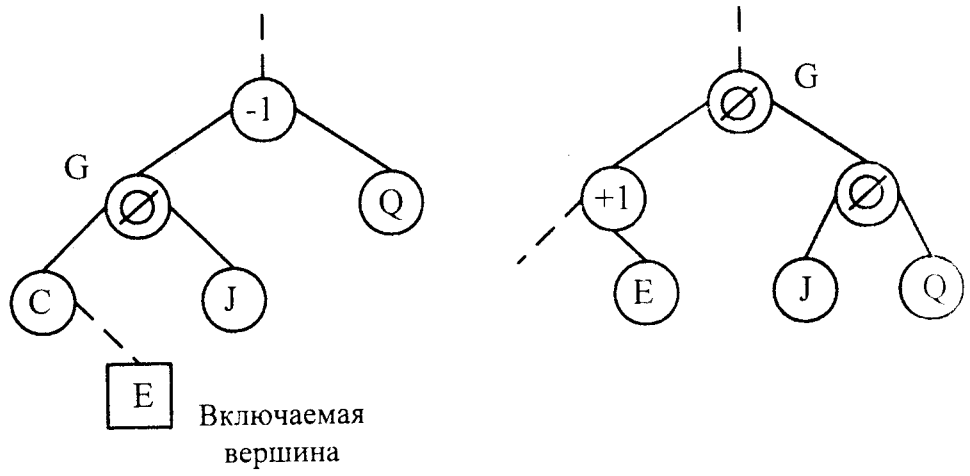

Рис. 6.12. Деревья с показателями сбалансированности узлов

ющиеся во многих словарях буквенные высечки. Тогда по первой букве данного слова можно отыскать страницы, содержащие все слова, начинающиеся с этой буквы. Развивая идею побуквенных высечек, получим схему поиска, основанную на индексации в структуре бора (термин использует часть слова выборка).

Бор представляет собой m-арное дерево. Каждый узел уровня h — это множество всех ключей, начинающихся с определенной последовательности из h литер. Узел определяет m-путевое разветвление в зависимости от $(h + 1)$-й литеры. Бор представляет собой таблицу следующего вида (табл. 6.3).

Таблица 6.3

Структура бора

Символ	Узел				
	1	2	3	...	*N*
Пробел (_)					

Пробел (_) — обязательный символ таблицы. В первом узле записывается первая буква или цифра ключа. Во втором узле ключей добавляется еще один символ и т.д. Если слово, начинающееся с определенной буквы (цифры), единственное, то оно сразу записывается в первом узле.

Пример 1. Дано множество

{A, AA, AB, ABC, ABCD, ABCA, ABCC, BOR, C, CC, CCC, CCCD, CCCB, CCCA}.

От исходного множества перейдем к построению бора. Исходный алфавит — {A, B, C, D}. BOR — единственное слово на букву «B» и оно побуквенно не разбивается. Узлы бора представляют собой векторы, каждая компонента которых представляет собой либо ключ, либо ссылку (возможно, пустую), как показано в табл. 6.4. Узел 1 — корень, и первую букву следует искать здесь. Если первой оказалась, например, буква «B», то из таблицы видно, что ему соответствует слово BOR.

Таблица 6.4

Пример поиска по бору

Символ	Узел						
	1	2	3	4	5	6	7
_		A_	AB_	ABC_	C_	CC_	CCC_
A	2	AA		ABCA			CCCA
B	BOR	3					CCCB
C	5		4	ABCC	6	7	
D				ABCD			CCCD

Если же первая буква A, то первый узел передает управление к узлу 2, где аналогичным образом отыскивается вторая буква. Узел 2 указывает, что вторыми буквами будут (пробел), A, B и т.д.

Пример 2. Пусть задано множество слов {воск, сок, оса, сто, сев, век, ост, сотка}. Построим *m*-арное дерево для этого списка (рис. 6.13).

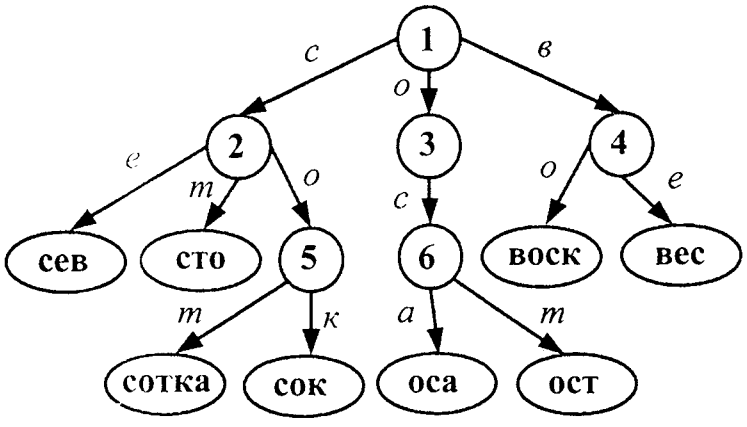

Рис. 6.13. *M*-арное дерево

На основе анализа этого дерева может быть построена табл. 6.5.

Следует отметить, что при построении таблицы-бора порядок букв в первом столбце таблицы может быть произвольным. Естественно, что

Таблица 6.5

Пример поиска по бору

Символ	Узел					
	1	2	3	4	5	6
С	(2)		(6)			
О	(3)	(5)		воск		
Т		сто			сотка	ост
К					сок	
А						оса
В	(4)					
Е		сев		вес		

для каждого конкретного порядка букв заполнение таблицы будет разным. Допустим, для рассматриваемого примера в списке слов нужно найти слово «ост». Оно начинается на букву «о». Смотрим пересечение первого столбца со строкой, обозначенной буквой «о». Там записана ссылка (3); следовательно, обращаемся к третьему столбцу «бора». Вторая буква слова «ост» — буква «с». На пересечении третьего столбца и строки с буквой «с» записана ссылка (6), которая отправляет нас на просмотр шестого столбца таблицы. В этом столбце видим два слова: «ост» и «оса». Но поскольку третьей буквой искомого слова является буква «т», на строке, обозначенной этой буквой, находим слово «ост».

6.7. Поиск хешированием

В основе поиска лежит переход от исходного множества к множеству хеш-функций $h(k)$. Хеш-функция имеет вид $h(k) = k \bmod m$, где k — ключ; m — целое число; mod — остаток от целочисленного деления.

Например, пусть дано исходное множество {9, 1, 4, 10, 8, 5}. Определим для него хеш-функцию $h(k) = k \bmod m$.

1) Пусть $m = 1$, тогда $h(k) = \{0, 0, 0, 0, 0, 0\}$. Множество хеш-функций состоит из нулей.

2) Пусть $m = 20$, отсюда $h(k) = \{9, 1, 4, 10, 8, 5\}$. Множество хеш-функций повторяет исходное множество.

3) Пусть m равно половине максимального ключа: $m = [K_{max}/2]$, тогда

$m = [10/2] = 5$, отсюда $h(k) = \{4, 1, 4, 0, 3, 0\}$.

Хеш-функция указывает адрес, по которому следует искать ключ. Для разных ключей хеш-функция может принимать одинаковые зна-

чения, и такая ситуация называется коллизией. Таким образом, поиск хешированием заключается в устранении коллизий методом цепочек (рис. 6.14).

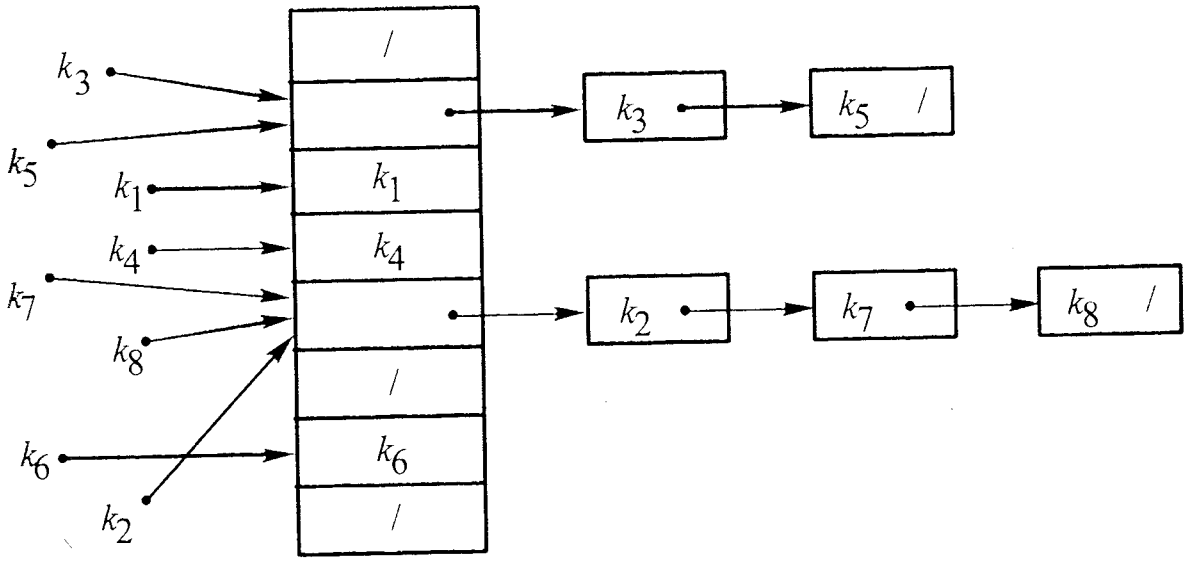

Рис. 6.14. Устранение коллизий методом цепочек

Пример 1. Дано множество ключей {7, 13, 6, 3, 9, 4, 8, 5}. Найти ключ $K = 27$. Хеш-функция $h(k) = K \bmod m$; $m = [13/2] = 6$ (поскольку 13 — максимальный ключ) $h(k) = \{1, 1, 0, 3, 3, 4, 2, 5\}$. Для устранения коллизий построим табл. 6.6. Попарным сравнением множества хеш-функций и множества исходных ключей заполняем таблицу (устанавливаем взаимно однозначное соответствие (биекцию) между двумя множествами).

Таблица 6.6

Разрешение коллизий

$h(k)$	Цепочка ключей
0	6
1	7, 13
2	8
3	3, 9
4	4
5	5

При этом хеш-функция указывает адрес, по которому следует отыскивать ключ. Например, если отыскивается ключ $K = 27$, тогда

$h(k) = 27 \bmod 6 = 3$. Это значит, что ключ $K = 27$ может быть только в строке, где $h(k) = 3$. Поскольку его там нет, то данный ключ отсутствует в исходном множестве.

Пример 2. Дано множество ключей {7, 1, 8, 5, 14, 9, 16, 3, 4}. Найти ключ $K = 14$. Хеш-функция равна $h(k) = K \bmod m$; $m = [16/2] = 8$ (поскольку 16 — максимальный ключ).

Следовательно, $h(k) = \{7, 1, 0, 5, 6, 1, 0, 3, 4\}$.

Для разрешения коллизий, которые присутствуют во множестве хеш-функций, построим табл. 6.7.

Таблица 6.7

Разрешение коллизий

$h(k)$	Цепочка ключей
0	8, 16
1	1, 9
3	3
4	4
5	5
6	14
7	7

При заполнении таблицы установим соответствие между исходным множеством и множеством хеш-функций. Поиск осуществляется по таблице: $K = 14$; $h(k) = 14 \bmod 8 = 6$. Это значит, что ключ $K = 14$ может быть только в строке со значением $h(k) = 6$.

Поиск с вставкой по рассеянной таблице с цепочками. Предлагаемый алгоритм позволяет отыскать в таблице из M элементов ключ K (например, при $M = 9$ имеем последовательность семи ключей $K =$ $= (EN, TO, TRE, FIRE, FEM, SEK, SYV)$. Если K нет в таблице и она не полна, то K вставляется в нее. Элементы таблицы обозначаются через $TABLE\,[i]$, $0 \le i \le M$ и могут быть двух различных типов: *свободный* и *занятый*. Занятый узел содержит ключевое поле $KEY\,[i]$, поле ссылки $LINK\,[i]$ и, возможно, другие поля. Алгоритм использует хеш-функцию $h(K)$. Для облегчения поиска свободного пространства используется вспомогательная переменная R; если таблица пуста, $R = M + 1$; по мере проведения вставок будет оставаться в силе утверждение, что узлы $TABLE\,[j]$ заняты для всех j в диапазоне $R \le j \le M$. Условимся, что узел $TABLE\,[0]$ всегда будет свободен.

Шаг 1. [Хеширование] Установить $i := h(K) + 1 (1 \le i \le M)$.

Шаг 2. [Список] Если узел *TABLE* [*i*] свободен, то перейти на шаг 6. (В противном случае этот узел занят, и исследуем начинающийся здесь список занятых узлов).

Шаг 3. [Сравнение] Если $K = KEY[i]$, поиск завершен удачно.

Шаг 4. [Переход к следующему.] Если $LINK[i] \neq 0$, установить $i := LINK[i]$ и вернуться на шаг 3.

Шаг 5. [Найти свободный узел.] (Поиск был неудачным, и необходимо найти в таблице свободное место.) Уменьшать R до тех пор, пока не будет получено такое значение, что узел *TABLE* [*R*] свободен. Если $R = 0$, алгоритм заканчивается по переполнению (свободных узлов больше нет); в противном случае установить $LINK[i] := R$; $i := R$.

Шаг 6. [Вставить новый ключ.] Пометить *TABLE* [*i*] как занятый узел с $KEY[i] := K$ и $LINK[i] := 0$.

В алгоритме допускается срастание нескольких списков, так что после вставки в таблицу записи перемещать не нужно. На рис. 6.15 *SEK* попадает в список, содержащий *TO* и *FIRE*, так как последний ключ уже вставлен в позицию 9.

Разрешение коллизий «открытой адресацией». Другой способ разрешения коллизий состоит в том, чтобы полностью отказаться от ссылок и просто просматривать один за другим различные элементы таблицы, пока не будут найдены ключ *K* или свободная позиция. Не плохо было бы иметь правило, согласно которому

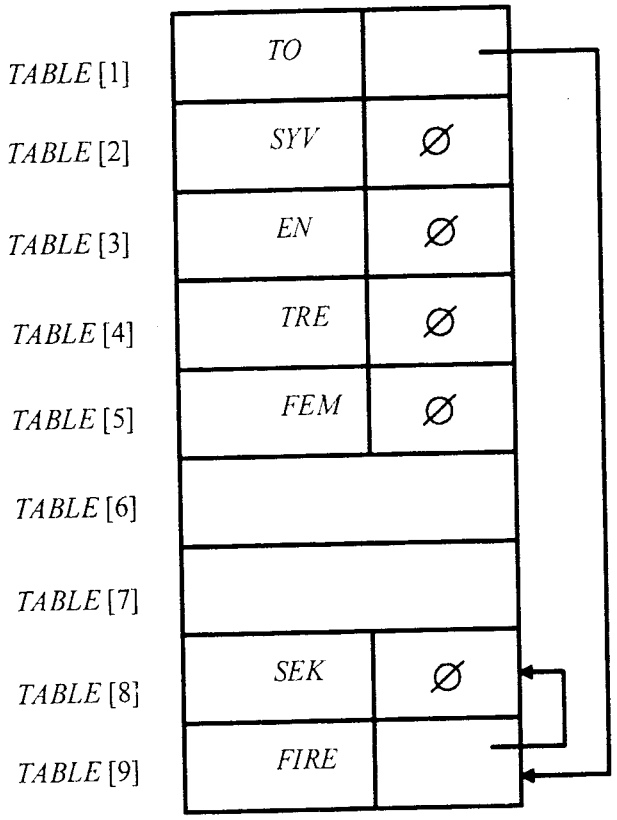

Рис. 6.15. Сросшиеся списки

каждый ключ K определяет последовательность проб, т.е. последовательность позиций в таблице, которые нужно просматривать всякий раз при вставке или поиске K. Если, используя определяемую K последовательность проб, натолкнемся на свободную позицию, то можно сделать вывод, что K нет в таблице, так как та же последовательность проб выполняется каждый раз при обработке данного ключа. Этот общий класс методов называется *открытой адресацией*.

Простейшая схема открытой адресации, известная как линейное опробование, использует циклическую последовательность проб:

$$h(K),\ h(K)-1,\ \ldots,\ 0,\ M-1,\ M-2,\ \ldots,\ h(K)+1.$$

Алгоритм поиска с вставкой по открытой рассеянной таблице. Алгоритм позволяет разыскать данный ключ K в таблице из M узлов. Если K нет в таблице и она не полна, ключ K вставляется.

Узлы таблицы обозначаются через $TABLE\,[i]$, $0 \le i < M$ и принадлежат двум различным типам узлов — свободных и занятых. Занятый узел содержит ключ $KEY\,[i]$ и, возможно, другие поля. Значение вспомогательной переменной N равно числу занятых узлов; эта переменная рассматривается как часть таблицы, и при вставке нового ключа ее значение увеличивается на единицу.

Данный алгоритм использует хеш-функцию $h(K)$ и линейную последовательность проб для адресации.

Шаг 1. [Хеширование.] Установить $i := h(K)$, $0 \le i < M$.

Шаг 2. [Сравнить.] Если узел $TABLE\,[i]$ свободен, то перейти на шаг 4. В противном случае, если $KEY\,[i] = K$, алгоритм заканчивается удачно.

Шаг 3. [Перейти к следующему.] Установить $i := i-1$; если теперь $i < 0$, установить $i := i + M$. Вернуться на шаг 2.

Шаг 4. [Вставить.] (Поиск был неудачным.) Если $N = M-1$, алгоритм заканчивается по переполнению. (В данном алгоритме считается, что таблица полна при $N = M-1$, а не при $N = M$). В противном случае установить $N := N + 1$, отметить, что узел $TABLE\,[i]$ занят и установить $KEY\,[i] := K$.

В табл. 6.8 показано, что происходит при вставке с помощью алгоритма L семи «норвежских» ключей, имеющих коды хеширования 2, 7, 1, 8, 2, 8, 1 соответственно. Последние три ключа — *FEM, SEK* и *SYV* — смещены по сравнению со своими начальными адресами $h(K)$.

Таблица 6.8

Линейная открытая адресация

Код хеширования	Ключи
0	*FEM*
1	*TRE*
2	*EN*
3	
4	
5	*SYV*
6	*SEK*
7	*TO*
8	*FIRE*

■ КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что понимается под поиском?
2. Каковы особенности поиска последовательного и бинарного?
3. В чем состоят особенности поиска интерполяционного и Фибоначчиевого?
4. Каковы особенности поиска по бинарному дереву?
5. К чему сводятся особенности поиска по бору и хешированием?
6. В чем состоит методика анализа сложности алгоритмов поиска?
7. В чем особенность алгоритмов поиска словесной информации?
8. Что такое коллизия?
9. Какой метод используется для разрешения коллизий?
10. Что определяет показатель сбалансированности узла дерева?