6.3. ЦИФРОВОЙ ПОИСК

ВМЕСТО методов поиска, основанных на сравнении ключей, можно воспользоваться представлением ключей в виде последовательности цифр или букв. Рассмотрим, например, "побуквенные метки" в больших словарях (или записных адресных книжках), которые позволяют мгновенно найти страницы со словами, начинающимися с определенной буквы*.

Развивая идею побуквенных меток до ее логического завершения, можно получить схему поиска, основанную на повторяемом индексировании, которое проиллюстрировано в табл. 1. Предположим, что необходимо проверить, является ли данный аргумент поиска одним из 31 наиболее употребительного английского слова (см. рис. 12 и 13 в разделе 6.2.2). Данные представлены в табл. 1 в виде *структуры луча***. Луч, по сути, — это M-арное дерево, узлы которого представляют M-местные векторы с компонентами, соответствующими цифрам или буквам. Каждый узел уровня l является набором всех ключей, начинающихся с определенной последовательности l символов, которая называется его *префиксом* (*prefix*); узел определяет разветвление на M путей в зависимости от $l+1$ символа.

Например, луч из табл. 1 содержит 12 узлов; узел (1) представляет собой корень, в котором мы ищем первую букву. Если первой буквой является, скажем, `N`, из таблицы следует, что либо это слово `NOT`, либо такого слова вообще нет в таблице. В то же время, если первая буква — `W`, то узел (1) отправит нас к узлу (9) для поиска второй буквы тем же способом. Узел (9) указывает, что вторая буква должна быть `A`, `H` или `I`. Префикс узла (10) — `HA`. Пустые записи в таблице означают отсутствие связей.

Узлы-векторы в табл. 1 расположены в соответствии с кодами символов `MIX`. Это означает, что "лучевой поиск" будет весьма быстрым, поскольку следует просто выбирать слова из массивов с использованием символов наших ключей в качестве индексов. Технологии быстрого многопутевого принятия решения по индексу называются просмотром таблицы (table look-at) в отличие от поиска по таблице (table look-up) [см. P. M. Sherman, *CACM* **4** (1961), 172–173, 175].

Алгоритм T (*"Лучевой поиск"* (*Trie search*)). Дана таблица записей в форме M-арного луча. Алгоритм осуществляет поиск по заданному аргументу K. Узлы луча представляют собой векторы, индексы которых изменяются от 0 до $M-1$; каждый компонент такого вектора представляет собой ключ или ссылку (возможно, пустую).

T1. [Инициализация.] Установить ссылочную переменную `P` таким образом, чтобы она указывала на корень луча.

T2. [Ветвление.] Установить k равным следующему символу входного аргумента ключа K слева направо. (Если аргумент полностью просканирован, установить

---

* Наличие у страницы книги трех свободных сторон позволило однажды переводчику этого раздела оснастить свой словарь метками поиска до третьей буквы слова включительно. — *Прим. перев.*

** В оригинале используется термин *trie* (произносится как английское слово "try"), предложенный Э. Фредкиным (E. Fredkin) [*CACM* **3** (1960), 490–500]. Этот термин представляет собой часть слова "re*trie*val". В русском переводе будет использоваться часть слова "по*луч*ение" (информации). И хотя при этом теряется определенная игра слов, основанная на схожести слов *trie* и *tree*, приобретается некоторый смысл, заложенный в слове *луч*: быстрый четко определенный по направлению движения поиск. — *Прим. перев.*

k равным "пустому" символу или символу "конец слова". Символ должен быть представлен в виде числа в диапазоне $0 \le k < M$). Обозначим через X k-й элемент в NODE(P). Если X представляет собой ссылку, перейти к шагу T3, если X — ключ, перейти к шагу T4.

T3. [Продвижение.] Если $X \neq \Lambda$, установить P $\leftarrow X$ и вернуться к шагу T2; в противном случае алгоритм завершается неудачей.

T4. [Сравнение.] Если $X = K$, алгоритм завершается успешно; в противном случае алгоритм завершается неудачей. ▮

Таблица 1

ЛУЧ ДЛЯ 31 НАИБОЛЕЕ УПОТРЕБИТЕЛЬНОГО АНГЛИЙСКОГО СЛОВА

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
␣		A				I					HE	
A	(2)				(10)				WAS			THAT
B	(3)											
C												
D										HAD		
E			BE		(11)							THE
F	(4)						OF					
G												
H	(5)							(12)	WHICH			
I	(6)				HIS				WITH			THIS
Θ												
J												
K												
L												
M												
N	NOT	AND				IN	ON					
O	(7)			FOR				TO				
P												
Q												
R		ARE		FROM			OR				HER	
Φ												
Π												
S		AS				IS						
T	(8)	AT				IT						
U			BUT									
V										HAVE		
W	(9)											
X												
Y	YOU		BY									
Z												

Заметим, что при неудачном поиске будет найден элемент, *более всего совпадающий* с искомым, что может оказаться полезным в некоторых приложениях.

Для сравнения скорости работы данного алгоритма со скоростями других алгоритмов из этой главы напишем коротенькую MIX-программу, в которой предполагается, что символы представляют собой байты, а максимальная длина ключа — 5 байт.

Программа T (*"Лучевой поиск"* (*Trie search*)). В этой программе предполагается, что все ключи занимают одно слово машины MIX; в случае, если в ключе содержится

менее 5 символов, он дополняется пробелами справа. Поскольку мы используем коды символов MIX, каждый байт аргумента поиска содержит число, меньшее 30. Ссылки представлены в виде отрицательных чисел в поле 0:2 узла слова. rI1 ≡ P, rX ≡ непросканированная часть K.

```
01  START   LDX  K          1   T1. Инициализация.
02          ENT1 ROOT       1   P ← указатель на корень луча.
03  2H      SLAX 1          C   T2. Ветвление.
04          STA  *+1(2:2)   C   Получение следующего символа, т. е. k.
05          ENT2 0,1        C   Q ← P + k.
06          LD1N 0,2(0:2)   C   P = LINK(Q).
07          J1P  2B         C   T3. Продвижение. Переход к T2, если P≠Λ.
08          LDA  0,2        1   T4. Сравнение. rA ← KEY(Q).
09          CMPA K          1
10          JE   SUCCESS    1   Успешное завершение при rA = K.
11  FAILURE EQU  *              Выход при отсутствии в луче.
```

Время работы программы составляет $8C + 8$ единиц, где C — количество проверяемых символов. Поскольку $C \leq 5$, время поиска никогда не превышает 48 единиц времени.

Если сравнить эффективность этой программы (при поиске на луче из табл. 1) и программы 6.2.2T (с использованием *оптимального* бинарного дерева поиска (см. рис. 13)), можно сделать следующие выводы.

1. Луч занимает гораздо больше памяти; мы использовали 360 слов для представления 31 ключа, в то время как бинарное дерево поиска использует только 62 слова памяти (однако в упр. 4 будет показано, что можно ухитриться втиснуть луч из табл. 1 в 49 слов).
2. Успешный поиск требует 26 единиц времени в обеих программах. Неудачный поиск выполняется быстрее в случае луча и медленнее — в бинарном дереве поиска. Для наших конкретных данных неудачный поиск будет осуществляться гораздо чаще, чем успешный, а потому для них "лучевой поиск" предпочтительнее с точки зрения скорости работы.
3. В случае применения луча для указателя KWIC (см. рис. 15) метод теряет свою привлекательность в силу природы используемых данных. Например, для различения слов COMPUTATION и COMPUTATIONS луч потребует 12 итераций. Поэтому было бы более разумно строить луч таким образом, чтобы сканирование слов происходило справа налево.

Абстрактная концепция луча для представления семейства строк была предложена Акселем Тью (Axel Thue) в статье о строках, которые не содержат смежных повторяющихся подстрок [*Skrifter udgivne af Videnskabs-Selskabet i Christiania*, Mathematisk-Naturvidenskabelig Klasse (1912), No. 1; перепечатана в книге Тью *Selected Mathematical Papers* (Oslo: Universitetsforlaget, 1977), 413–477].

"Лучевая память" для компьютерного поиска была впервые рекомендована Рене де ла Бриандe (René de la Briandais) [*Proc. Western Joint Computer Conf.* **15** (1959), 295–298]. Он указал, что можно сохранить память (за счет времени работы) при использовании связанного списка для каждого узла-вектора, поскольку большинство элементов вектора обычно пусто. На самом деле эта идея приводит к замещению луча из табл. 1 лесом, показанным на рис. 31. Поиск в таком лесу осуществляется

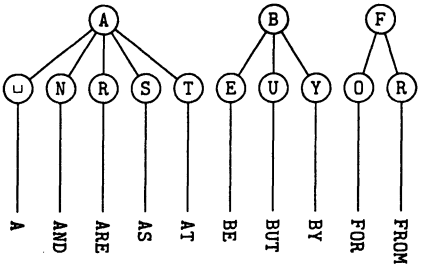
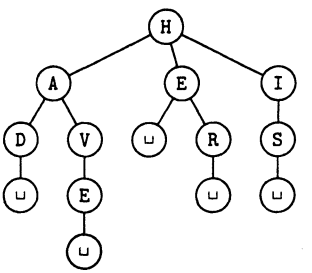


Рис. 31. Луч из табл. 1, преобразованный в "лес".

путем нахождения корня, соответствующего первому символу, затем дочернего узла этого корня, соответствующего второму символу, и т. д.

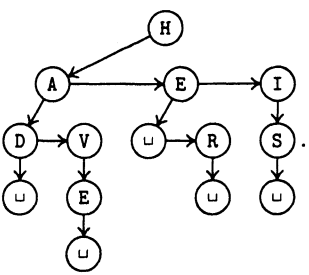
В своей статье де ла Брианде в действительности не прерывал ветвление дерева, как показано в табл. 1 или на рис. 31; вместо этого он продолжал представление каждого ключа, символ за символом, до достижения конца слова. Таким образом, де ла Брианде использовал бы



(1)

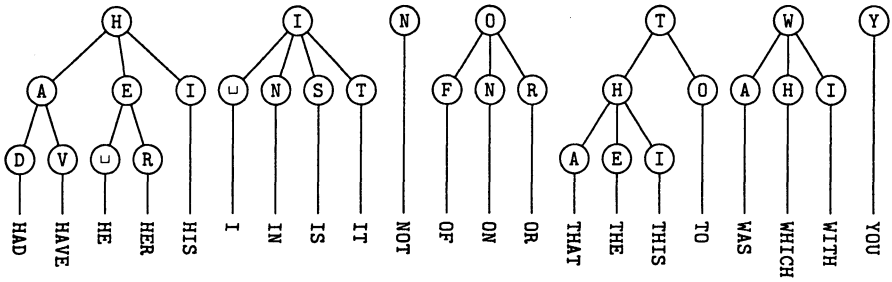
вместо дерева H, показанного на рис. 31. Для такого представления требуется больше памяти, но при этом существенно упрощается работа с ключами переменной длины. При использовании двух полей ссылок на каждый символ легко осуществляются динамическая вставка и удаление.

Если использовать обычный путь представления деревьев как бинарных деревьев, (1) становится бинарным деревом



(2)

(В представлении полного леса на рис. 31 следовало бы добавить указатель, ведущий вправо от H к соседнему корню I.) Поиск в этом бинарном дереве выполняется посредством сравнения аргумента с символом в дереве и следования RLINK до поиска соответствия; после этого мы находим LLINK и точно так же работаем с очередным символом аргумента.



Поиск в таком бинарном дереве в большей или меньшей степени сводится к поиску путем сравнений, но ветвление осуществляется по признаку "равно-не равно", а не "меньше-больше". Элементарная теория из раздела 6.2.1 гласит, что необходимо выполнить в среднем хотя бы $\lg N$ сравнений для того, чтобы различить N ключей. Среднее количество сравнений, сделанных при поиске в дереве, которое подобно изображенному на рис. 31, должно быть не меньше количества сравнений, выполняемых при бинарном поиске с использованием описанных в разделе 6.2 технологий.

С другой стороны, луч из табл. 1 способен выполнять M-путевое ветвление за один раз; мы увидим, что среднее время поиска для больших N включает всего около $\log_M N = \lg N / \lg M$ итераций при случайных входных данных. Мы также увидим, что "чистая" схема луча (подобная алгоритму T) требует, в целом, примерно $N/\ln M$ узлов для различения N случайных ключей; следовательно, общее количество необходимой памяти пропорционально $MN/\ln M$.

Из этих рассуждений становится ясно, что идея луча хороша только для нескольких первых уровней дерева. Повысить производительность можно за счет комбинирования двух стратегий: луча для нескольких первых символов и переключения на другую стратегию — для оставшихся. Например, Э. Г. Сассенгат (мл.) (E. H. Sussenguth, Jr.) [*CACM* **6** (1963), 272–279] предложил использовать посимвольную схему до достижения части дерева, в которой возможны, скажем, не более шести ключей, а затем проходить последовательно по этому списку. Далее мы увидим, что такая смешанная стратегия позволяет уменьшить количество узлов луча примерно в шесть раз без существенного изменения времени работы.

Т. Н. Турба (T. N. Turba) [*CACM* **25** (1982), 522–526] указал, что иногда при поиске с ключами переменной длины удобно иметь по одному поисковому дереву или лучу для каждой длины ключа.

Бинарный цифровой поиск. Допустим, что $M = 2$, и аргумент поиска сканируется по одному биту. Для этого случая разработаны два специальных интересных метода.

Первый метод, именуемый цифровым *поиском по дереву* (*digital tree search*), разработан Э. Г. Коффманом (E. G. Coffman) и Дж. Ивом (J. Eve) [*CACM* **13** (1970), 427–432, 436]. Его суть заключается в хранении полных ключей в узлах так же, как в алгоритмах поиска по дереву из раздела 6.2.2, но с использованием битов аргумента вместо результатов сравнения для выбора левой или правой ветви. На рис. 32 изображено бинарное дерево, построенное по этому методу путем вставки 31 наиболее употребительного английского слова в порядке уменьшения частот появления слов. Чтобы получить данные для иллюстрации метода, слова были

представлены в кодах символов MIX и конвертированы в двоичные числа с пятью битами на байт. Так, слово WHICH представляется битовой последовательностью 11010 01000 01001 00011 01000.

Для поиска слова WHICH на рис. 32 сравним его со словом THE в корне дерева. Так как они не совпадают и первый бит слова WHICH равен 1, мы двигаемся вправо и сравниваем его со словом OF. Слова опять не совпадают, и следующий бит слова WHICH равен 1, поэтому мы снова перемещаемся вправо и сравниваем наше слово WHICH со словом WITH, и т. д. Алфавитный порядок ключей в цифровом поиске по дереву больше не соответствует симметричному порядку узлов.

Интересно обратить внимание на разницу между деревьями, представленными на рис. 32 и 12 (из раздела 6.2.2), так как последнее дерево было построено тем же способом, но с использованием сравнения ключей вместо битов. Если принять во внимание данные часто́ты появления слов, цифровой поиск по дереву на рис. 32 потребует в среднем 3.42 сравнений при успешном завершении поиска. Это несколько лучше, чем в случае дерева, показанного на рис. 12, для которого требуется 4.04 сравнения (хотя, конечно же, время самого сравнения различно для этих двух ситуаций).

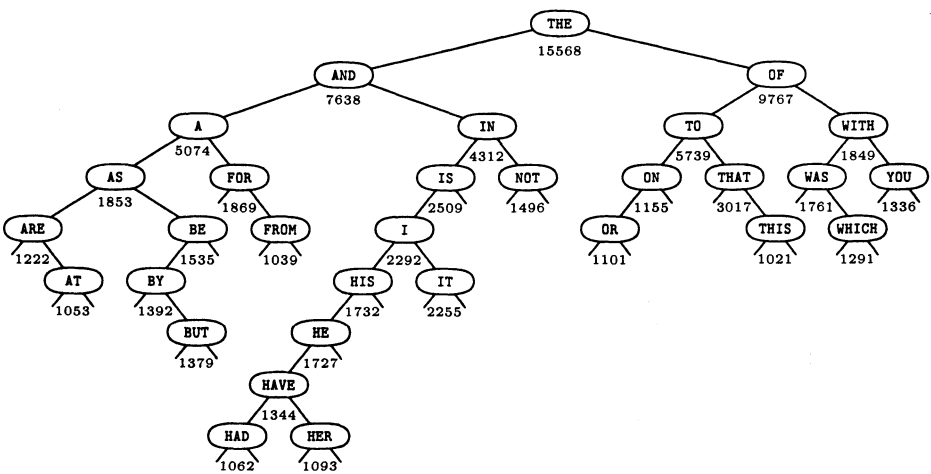


Рис. 32. Дерево цифрового поиска для 31 наиболее употребительного английского слова, вставленного в порядке уменьшения частот.

Алгоритм D (*Цифровой поиск со вставкой по дереву*). Дана таблица записей, представляющих бинарное дерево, которое описано выше. Алгоритм предназначен для поиска некоторого аргумента K. Если K отсутствует в таблице, новый узел, содержащий K, вставляется в дерево в соответствующем месте.

Алгоритм предполагает, что дерево не пусто и что его узлы имеют поля KEY, LLINK и RLINK, такие же, как в алгоритме 6.2.2T. В действительности, как вы можете убедиться сами, алгоритмы практически идентичны.

D1. [Инициализация.] Установить P $\leftarrow$ ROOT и $K' \leftarrow K$.

D2. [Сравнение.] Если K = KEY(P), поиск успешно завершается. В противном случае установить b равным лидирующему биту K' и сдвинуть K' влево на

один бит (тем самым удалив этот бит и вставив 0 справа). Если $b = 0$, перейти к шагу D3; в противном случае перейти к шагу D4.

D3. [Перемещение влево.] Если LLINK(P) $\neq \Lambda$, установить P $\leftarrow$ LLINK(P) и перейти к шагу D2. В противном случае перейти к шагу D5.

D4. [Перемещение вправо.] Если RLINK(P) $\neq \Lambda$, установить P $\leftarrow$ RLINK(P) и перейти к шагу D2.

D5. [Вставка в дерево.] Установить Q $\Leftarrow$ AVAIL, KEY(Q) $\leftarrow K$, LLINK(Q) $\leftarrow$ RLINK(Q) $\leftarrow \Lambda$. Если $b = 0$, присвоить LLINK(P) $\leftarrow$ Q; в противном случае присвоить RLINK(P) $\leftarrow$ Q. ▮

Хотя алгоритм поиска по дереву 6.2.2T по сути своей бинарный, не сложно увидеть, что он может быть распространен на M-арный цифровой поиск для любого $M \geq 2$ (см. упр. 13).

Дональд Р. Моррисон (Donald R. Morrison) [*JACM* **15** (1968), 514–534] открыл весьма привлекательный способ построения N-узловых деревьев поиска, основанных на бинарном представлении ключей *без* необходимости их хранения в узлах. Этот метод, названный "Патриция" (Patricia — **P**ractical **A**lgorithm **T**o **R**etrieve **I**nformation **C**oded **I**n **A**lphanumeric), очень хорошо подходит для работы с большими ключами переменной длины, например с заголовками или фразами, хранящимися в файле большого объема. Похожий алгоритм был одновременно опубликован в Германии (G. Gwehenberger, *Elektronische Rechenanlagen* **10** (1968), 223–226).

Основная суть метода "Патриция" состоит в построении бинарного дерева без однопутевых ветвей посредством включения в каждый узел количества битов, которые можно пропустить, прежде чем приступить к следующему тесту. Существует несколько способов реализации этой идеи; возможно, простейший из них представлен на рис. 33. Имеется массив битов TEXT (обычно довольно длинный), который может храниться во внешнем файле с произвольным доступом, поскольку при каждом поиске обращение к TEXT осуществляется только один раз. Каждый ключ, который должен храниться в нашей таблице, определяется местом его начала в тексте; можно считать, что он идет от места своего начала и до конца текста. (Метод "Патриция" не ищет точного соответствия между ключом и аргументом; вместо этого определяется, существует ли ключ, *начинающийся* с аргумента).

В ситуации, показанной на рис. 33, представлено семь ключей, которые начинаются с каждого входящего в текст слова, а именно — с "THIS IS THE HOUSE THAT JACK BUILT?", "IS THE HOUSE THAT JACK BUILT?", ..., "BUILT?". Имеется одно важное ограничение: *ни один ключ не может быть началом другого.* Оно выполняется, если текст завершается специальным символом конца текста (в нашем случае это "?"), который не встречается нигде в тексте. То же ограничение неявно применяется и в схеме луча алгоритма T, в которой признаком конца слова служит "␣". Дерево, используемое методом "Патриция" для поиска, должно целиком размещаться в оперативной памяти с произвольным доступом (или должно быть организовано по страничной схеме, описанной в разделе 6.2.4). Оно состоит из заголовка и $N - 1$ узла; узлы имеют несколько полей.

KEY, указатель на текст. Это поле должно иметь длину как минимум $\lg C$ бит, если текст содержит C символов. На рис. 33 слова, показанные внутри

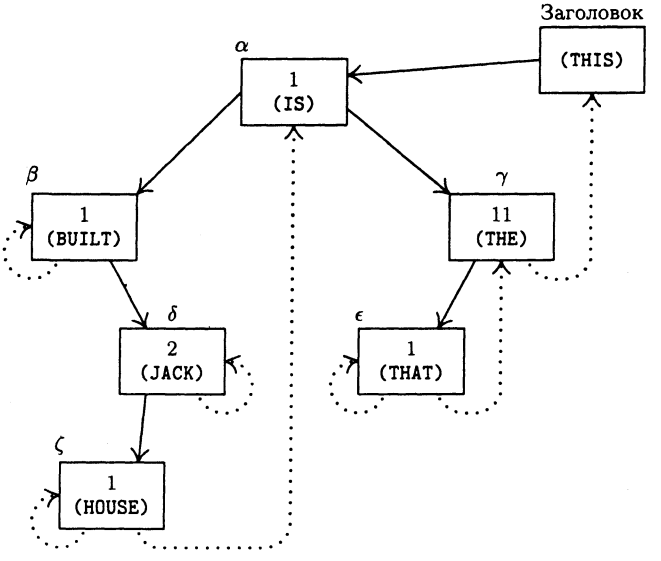


Рис. 33. Пример текста и дерева метода "Патриция".

каждого узла, на самом деле представлены указателями на текст, например вместо (JACK) узел содержит число 24, которое указывает начальное положение JACK BUILT? в текстовой строке.

LLINK и RLINK, ссылки внутри дерева. Длина этих полей должна составлять не менее $\lg N$ бит.

LTAG и RTAG, однобитовые поля, которые указывают, являются ли LLINK и RLINK ссылками на дочерние или родительские узлы данного узла соответственно. Пунктирные линии на рис. 33 соответствуют указателям, биты TAG которых равны 1.

SKIP, число, которое указывает, сколько битов должно быть пропущено при поиске (как объяснялось ранее). Это поле должно быть достаточно велико для хранения наибольшего числа k, для которого в двух различных ключах найдутся совпадающие подцепочки из k бит. Обычно на практике можно считать, что k не слишком велико, и сообщать об ошибке при превышении размеров поля SKIP. На рис. 33 поля SKIP показаны как числа внутри узлов.

Заголовок содержит только поля KEY, LLINK и LTAG.

Поиск в дереве метода "Патриция" выполняется следующим образом. Предположим, необходимо найти слово THE (его битовое представление — 10111 01000 00101). Начинаем просмотр с поля SKIP в корневом узле α, которое указывает, что следует проверить первый бит аргумента. Этот бит равен 1, и потому мы должны двигаться вправо. Поле SKIP следующего узла, γ, указывает, что теперь нужно обратить внимание на $1 + 11 = 12$-й бит аргумента. Он равен 0, поэтому мы движемся влево. Поле SKIP следующего узла, ϵ, заставляет нас взглянуть на $(12 + 1)$-й бит, равный 1.

Находим, что RTAG = 1, так что следует вернуться к узлу γ, который отсылает нас к массиву TEXT. Тот же путь поиска будет получен для любого аргумента, битовая маска которого равна 1xxxx xxxxx x01..., и необходимо проверить, не соответствует ли аргумент уникальному ключу, начинающемуся с этой маски, а именно — с THE.

Предположим, с другой стороны, что нужно найти некоторый ключ, начинающийся с TH (или все такие ключи). Процесс поиска начинается так же, как описывалось выше, но в конечном счете мы попытаемся обратиться к несуществующему двенадцатому биту десятибитового аргумента. Теперь необходимо сравнить аргумент с фрагментом массива TEXT, определяемым в текущем узле (в нашем случае — узле γ). Если совпадения не произошло, значит, аргумент не начинается ни с одного ключа; если же совпадение имело место, то аргумент служит началом любого ключа, на который указывают пунктирные линии, выходящие из узла γ и его потомков (а именно — THIS, THAT, THE).

Более точно процесс можно описать следующим образом.

Алгоритм P (*"Патриция"*). Дан массив TEXT и дерево с описанными выше полями KEY, LLINK, RLINK, LTAG, RTAG и SKIP. Алгоритм определяет, имеется ли в массиве TEXT ключ, начинающийся с некоторого аргумента K. (Если имеется $r \geq 1$ таких ключей, за $O(r)$ шагов можно последовательно установить расположение каждого из них; см. упр. 14.) Предполагается, что имеется, по меньшей мере, один ключ.

P1. [Инициализация.] Установить P $\leftarrow$ HEAD и $j \leftarrow 0$. (Переменная P представляет собой указатель, который будет перемещаться вниз по дереву, а j — счетчик, определяющий позиции битов аргумента.) Установить n равным количеству битов в аргументе K.

P2. [Движение влево.] Присвоить Q $\leftarrow$ P и P $\leftarrow$ LLINK(Q). Если LTAG(Q) = 1, перейти к шагу P6.

P3. [Пропуск битов.] (В этот момент известно, что если первые j бит K соответствуют некоторому ключу, то они соответствуют ключу, начинающемуся в KEY(P)). Установить $j \leftarrow j +$ SKIP(P). Если $j > n$, перейти к шагу P6.

P4. [Проверка бита.] (В этот момент известно, что если первые $j-1$ бит аргумента соответствуют некоторому ключу, то они соответствуют ключу, начинающемуся в KEY(P)). Если j-й бит аргумента равен 0, перейти к шагу P2; в противном случае — к шагу P5.

P5. [Перемещение вправо.] Присвоить Q $\leftarrow$ P и P $\leftarrow$ RLINK(Q). Если RTAG(Q) = 0, перейти к шагу P3.

P6. [Сравнение.] (В этот момент известно, что если аргумент соответствует некоторому ключу, то он соответствует ключу, начинающемуся в KEY(P)). Сравнить K с ключом, начинающимся в позиции KEY(P) в массиве TEXT. Если они эквивалентны (до n-го бита (длины K)), то алгоритм успешно завершается; в случае неравенства он завершается неудачей. ▮

В упр. 15 показано, каким образом может быть построено дерево метода "Патриция". Можно также вносить дополнения к тексту и вставлять новые ключи, если новый текстовый материал всегда заканчивается уникальным разделителем (например, символом конца текста с последующим серийным номером).

Алгоритм "Патриция" несколько причудлив, и требуется внимание для выявления всех его достоинств.

Анализ алгоритмов. Завершается этот раздел математическим анализом лучей, деревьев цифрового поиска и метода "Патриция". Важнейшие выводы будут приведены в самом конце раздела.

Сначала рассмотрим бинарные лучи, т. е. лучи с $M = 2$. На рис. 34 показан бинарный луч, который был получен при рассмотрении шестнадцати ключей из примеров сортировки (глава 5) в качестве 10-битовых двоичных чисел. (Ключи показаны в *восьмеричной записи*, например *1144* представляет собой десятибитовое число $612 = (1001100100)_2$.) Как и в алгоритме T, для хранения информации о ведущих битах ключей (до тех пор, пока ключ не идентифицируется однозначно) используется луч, в котором ключ записывается полностью.

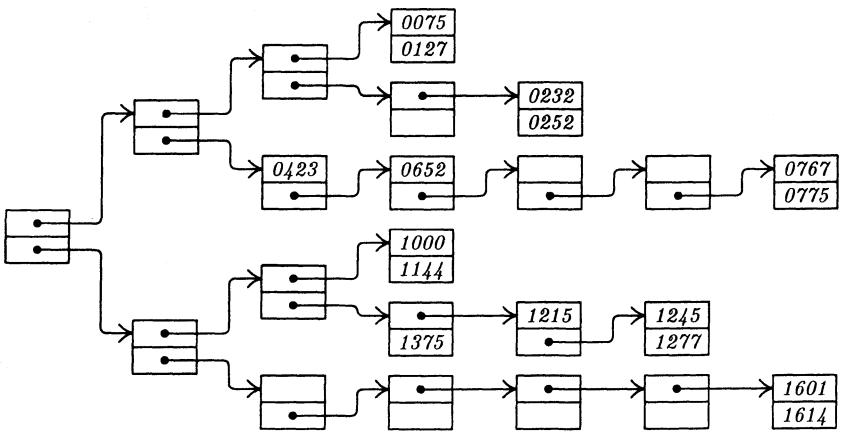


Рис. 34. Пример случайного бинарного луча.

Если сравнить рис. 34 с табл. 5.2.2–3, обнаружится удивительная взаимосвязь между "лучевой" памятью и обменной поразрядной сортировкой. (Возможно, эта связь очевидна.) 22 узла на рис. 34 точно совпадают с 22 стадиями из табл. 5.2.2–3 с соответствием p-го узла в предварительном порядке p-й стадии. Количество проверяемых на стадиях разбиения битов равно числу ключей в соответствующих узлах и их подлучах; значит, можно сформулировать следующий результат.

Теорема T. *Если N различных двоичных чисел помещены в описанный выше бинарный луч, то* (i) *количество узлов луча равно количеству стадий разбиения при обменной поразрядной сортировке и* (ii) *среднее количество проверок битов, требующееся для выборки ключа с помощью алгоритма T, равно $1/N$ от количества проверок битов при обменной поразрядной сортировке.* ▮

Благодаря этой теореме можно использовать весь математический аппарат, разработанный для поразрядной сортировки в разделе 5.2.2. Например, если предположить, что ключами являются случайные равномерно распределенные между 0 и 1 числа, заданные с бесконечной точностью, то количество проверок битов, необходимое для выборки, равно $\lg N + \gamma/\ln 2 + 1/2 + \delta(N) + O(N^{-1})$, а число узлов

луча — $N/\ln 2 + N\bar{\delta}(N) + O(1)$. Здесь $\delta(N)$ и $\bar{\delta}(N)$ — сложные функции, которыми можно пренебречь, поскольку их абсолютные значения никогда не превышают 10^{-6} (см. упр. 5.2.2–38 и 5.2.2–48).

Конечно же, перед нами стоит более трудная задача: обобщить результаты, полученные для бинарных лучей, на случай M-арных лучей. Мы опишем лишь стартовую точку исследований, помещая детальные инструкции в упражнения.

Пусть A_N — среднее число внутренних узлов в случайном M-арном луче поиска, который содержит N ключей. Тогда $A_0 = A_1 = 0$ и для $N \geq 2$ имеем

$$A_N = 1 + \sum_{k_1+\cdots+k_M=N} \left(\frac{N!}{k_1! \ldots k_M!} M^{-N} \right) (A_{k_1} + \cdots + A_{k_M}), \tag{3}$$

поскольку $N!\, M^{-N}/k_1! \ldots k_M!$ представляет собой вероятность того, что k_1 ключей находятся в первом подлуче, ..., k_M — в M-м. Это соотношение может быть переписано как

$$\begin{aligned} A_N &= 1 + M^{1-N} \sum_{k_1+\cdots+k_M=N} \left(\frac{N!}{k_1! \ldots k_M!} \right) A_{k_1} \\ &= 1 + M^{1-N} \sum_k \binom{N}{k} (M-1)^{N-k} A_k \qquad \text{при } N \geq 2 \end{aligned} \tag{4}$$

с использованием симметрии и суммирования по $k_2, \ldots, k_M$. Аналогично, если через C_N обозначить общее среднее количество проверок битов, необходимое для поиска всех N ключей в луче, то $C_0 = C_1 = 0$ и

$$C_N = N + M^{1-N} \sum_k \binom{N}{k} (M-1)^{N-k} C_k \qquad \text{при } N \geq 2. \tag{5}$$

В упр. 17 показано, как работать с такого рода рекуррентными соотношениями, а в упр. 18–25 приводится соответствующая теория случайных лучей [с другой точки зрения анализ A_N был впервые проведен в работе L. R. Johnson, M. H. McAndrew, *IBM J. Res. and Devel.* **8** (1964), 189–193, в связи с эквивалентным аппаратно-ориентированным алгоритмом сортировки].

Если теперь перейти к изучению деревьев цифрового поиска, то обнаружится, что формулы похожи, однако не настолько, чтобы можно было легко определить асимптотическое поведение. Например, если через $\bar{C}_N$ обозначить среднее суммарное количество проверок битов при поиске всех N ключей в M-арном дереве цифрового поиска, нетрудно вывести, как и ранее, что $\bar{C}_0 = \bar{C}_1 = 0$ и

$$\bar{C}_{N+1} = N + M^{1-N} \sum_k \binom{N}{k} (M-1)^{N-k} \bar{C}_k \qquad \text{при } N \geq 0. \tag{6}$$

Это выражение практически идентично выражению (5), однако появления $N+1$ вместо N в левой части уравнения достаточно для того, чтобы коренным образом изменить характер рекуррентности, а потому использовавшиеся при изучении (5) методы становятся неприемлемыми.

Рассмотрим сначала бинарный цифровой поиск. На рис. 35 показано дерево цифрового поиска, соответствующее шестнадцати ключам из рис. 34, если они

вставляются в порядке, который использовался в примерах главы 5. Если будет необходимо определить среднее количество проверок битов при случайном успешном поиске, окажется, что это просто длина внутреннего пути дерева, деленная на N, так как для поиска узла на уровне l потребуется l проверок. Заметим, однако, что среднее число проверок битов при случайном *неудачном* поиске *не так* просто связано с длиной внешнего пути дерева, поскольку неудачный поиск с большей вероятностью оканчивается во внешнем узле недалеко от корня. Так, вероятность достижения левой ветви узла *0075* на рис. 35 равна $\frac{1}{8}$ (при рассмотрении ключей с бесконечной точностью), а вероятность попадания в левую ветвь узла *0232* составляет лишь $\frac{1}{32}$. Из-за этого деревья цифрового поиска при равномерном распределении ключей в целом лучше сбалансированы, чем бинарные деревья поиска из алгоритма 6.2.2T.

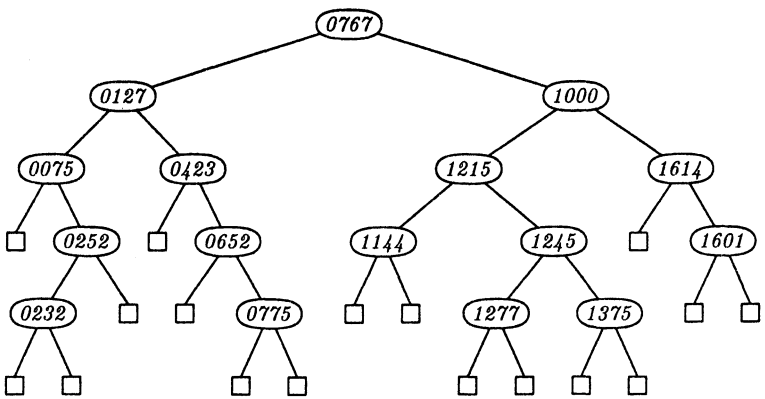


Рис. 35. Дерево случайного цифрового поиска, построенное по алгоритму D.

Для описания соответствующих характеристик деревьев цифрового поиска можно воспользоваться производящей функцией. Пусть на уровне l находится a_l внутренних узлов. Рассмотрим производящую функцию $a(z) = \sum_l a_l z^l$. Допустим, что производящая функция, соответствующая рис. 35, — $a(z) = 1 + 2z + 4z^2 + 5z^3 + 4z^4$. Если на уровне l находится b_l внешних узлов и $b(z) = \sum_l b_l z^l$, то согласно упр. 6.2.1–25 получаем

$$b(z) = 1 + (2z - 1)a(z). \tag{7}$$

Например, $1 + (2z - 1)(1 + 2z + 4z^2 + 5z^3 + 4z^4) = 3z^3 + 6z^4 + 8z^5$. Среднее количество проверок битов при случайном *успешном* поиске равно $a'(1)/a(1)$, так как $a'(1)$ — длина внутреннего пути дерева, а $a(1)$ — количество внутренних узлов. Среднее количество проверок битов при случайном *неудачном* поиске составляет $\sum_l l b_l 2^{-l} = \frac{1}{2}b'(\frac{1}{2}) = a(\frac{1}{2})$, поскольку мы достигаем данного внешнего узла уровня l с вероятностью 2^{-l}. Количество сравнений совпадает с количеством проверок битов (плюс 1 при успешном завершении поиска). Например, на рис. 35 при успешном поиске будет выполнено в среднем $2\frac{9}{16}$ проверок битов и $3\frac{9}{16}$ сравнений, в то время как в случае неудачного поиска обе эти величины будут равны $3\frac{7}{8}$.

Обозначим через $g_N(z)$ усредненную по всем деревьям с N узлами функцию $a(z)$; другими словами, $g_N(z)$ представляет собой сумму $\sum p_T a_T(z)$ по всем бинар-

ным деревьям цифрового поиска T с N внутренними узлами, где $a_T(z)$ — производящая функция для внутренних узлов дерева T, а p_T — вероятность того, что T образуется при вставке N случайных чисел согласно алгоритму D. Тогда среднее количество проверок битов будет равно $g'_N(1)/N$ в случае успешного и $g_N(\frac{1}{2})$ — в случае неудачного поиска.

Можно вычислить $g_N(z)$ при помощи имитации процесса построения дерева следующим образом. Если $a(z)$ представляет собой производящую функцию для дерева с N узлами, можно сформировать из него $N+1$ дерево методом вставки в позицию любого внешнего узла. Эта вставка производится в данный внешний узел уровня l с вероятностью 2^{-l}; следовательно, сумма производящих функций для $N+1$ нового дерева, умноженная на вероятность ее появления, равна $a(z)+b(\frac{1}{2}z) = a(z)+1+(z-1)a(\frac{1}{2}z)$. Усреднение по всем деревьям с N узлами дает

$$g_{N+1}(z) = g_N(z) + 1 + (z-1)g_N(\tfrac{1}{2}z); \qquad g_0(z) = 0. \tag{8}$$

С соответствующей производящей функцией для внешних узлов,

$$h_N(z) = 1 + (2z-1)g_N(z),$$

работать несколько легче в связи с тем, что (8) эквивалентно формуле

$$h_{N+1}(z) = h_N(z) + (2z-1)h_N(\tfrac{1}{2}z); \qquad h_0(z) = 1. \tag{9}$$

Многократно применяя это правило, находим, что

$$\begin{aligned} h_{N+1}(z) &= h_{N-1}(z) + 2(2z-1)h_{N-1}(\tfrac{1}{2}z) + (2z-1)(z-1)h_{N-1}(\tfrac{1}{4}z) \\ &= h_{N-2}(z) + 3(2z-1)h_{N-2}(\tfrac{1}{2}z) + 3(2z-1)(z-1)h_{N-2}(\tfrac{1}{4}z) \\ &\quad + (2z-1)(z-1)(\tfrac{1}{2}z-1)h_{N-2}(\tfrac{1}{8}z) \end{aligned}$$

и т. д. В результате имеем следующее:

$$h_N(z) = \sum_k \binom{N}{k} \prod_{j=0}^{k-1} (2^{1-j}z - 1); \tag{10}$$

$$g_N(z) = \sum_{k\ge 0} \binom{N}{k+1} \prod_{j=0}^{k-1} (2^{-j}z - 1). \tag{11}$$

Например, $g_4(z) = 4 + 6(z-1) + 4(z-1)(\frac{1}{2}z-1) + (z-1)(\frac{1}{2}z-1)(\frac{1}{4}z-1)$. Эти формулы позволяют выразить искомые величины в виде сумм произведений:

$$\bar{C}_N = g'_N(1) = \sum_{k\ge 0} \binom{N}{k+2} \prod_{j=1}^{k} (2^{-j} - 1); \tag{12}$$

$$g_N(\tfrac{1}{2}) = \sum_{k\ge 0} \binom{N}{k+1} \prod_{j=1}^{k} (2^{-j} - 1) = \bar{C}_{N+1} - \bar{C}_N. \tag{13}$$

Совершенно неочевидно, что эти формулы для $\bar{C}_N$ удовлетворяют (6)!

К сожалению, данные выражения непригодны для вычислений или поиска асимптотических зависимостей, поскольку $2^{-j} - 1$ отрицательно; мы получим большие

члены, многие из которых сократятся. Более полезная формула для $\bar{C}_N$ может быть получена в результате применения тождеств из упр. 5.1.1–16:

$$\begin{aligned}
\bar{C}_N &= \Big(\prod_{j\ge1}(1-2^{-j})\Big)\sum_{k\ge0}\binom{N}{k+2}(-1)^k\prod_{l\ge0}(1-2^{-l-k-1})^{-1}\\
&= \Big(\prod_{j\ge1}(1-2^{-j})\Big)\sum_{k\ge0}\binom{N}{k+2}(-1)^k\sum_{m\ge0}(2^{-k-1})^m\prod_{r=1}^{m}(1-2^{-r})^{-1}\\
&= \sum_{m\ge0}2^m\Big(\sum_k\binom{N}{k}(-2^{-m})^k-1+2^{-m}N\Big)\prod_{j\ge0}(1-2^{-j-m-1})\\
&= \sum_{m\ge0}2^m\big((1-2^{-m})^N-1+2^{-m}N\big)\sum_{n\ge0}(-2^{-m-1})^n\frac{2^{-n(n-1)/2}}{\prod_{r=1}^{n}(1-2^{-r})}. \qquad (14)
\end{aligned}$$

На первый взгляд, никакого улучшения по сравнению с (12) не произошло, однако очень большое достоинство полученного результата заключается в том, что сумма по m довольно быстро сходится для любого фиксированного n. Аналогичная ситуация возникает и в случае луча в формулах 5.2.2–(38) и 5.2.2–(39); в самом деле, при рассмотрении только членов с $n = 0$ из (14) получается в точности $N - 1$ плюс количество проверок битов в бинарном луче. Теперь можно получить асимптотическое значение так же, как и ранее (см. упр. 27). [Приведенный вывод базируется, в основном, на подходе, предложенном в работе A. J. Konheim, D. J. Newman, *Discrete Mathematics* **4** (1973), 57–63].

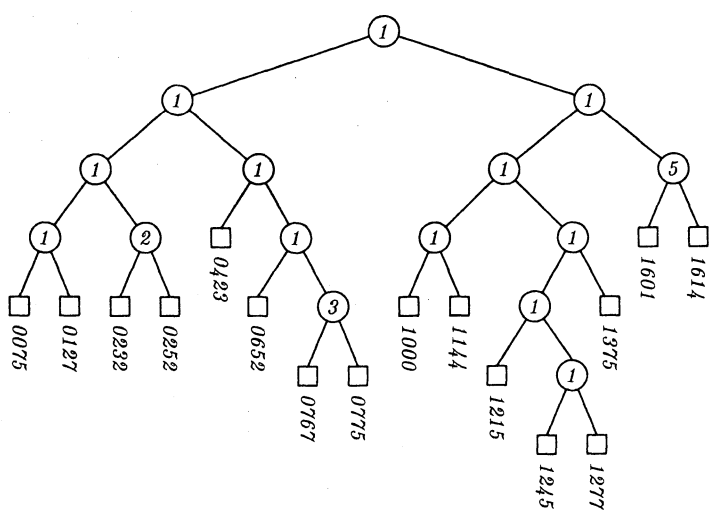


Рис. 36. Дерево, которое строится алгоритмом "Патриция" вместо луча, приведенного на рис. 34.

И, наконец, бросим на метод "Патриция" "математический" взгляд. В таком случае бинарное дерево похоже на соответствующий луч с теми же ключами, но

сжатыми вместе (поля SKIP позволяют устранить однопутевые ветвления), так что всегда имеется ровно $N-1$ внутренних узлов и N внешних узлов. На рис. 36 показано дерево метода "Патриция", соответствующее шестнадцати ключам луча, представленного на рис. 34. Число, показанное в каждом узле ветви, представляет собой значение SKIP; ключи расположены у внешних узлов, хотя они и не присутствуют в дереве явно (в действительности на месте каждого внешнего узла имеется ссылка на внутренний узел, который, в свою очередь, ссылается на массив TEXT). Впрочем, для целей анализа будем считать, что внешние узлы имеют именно такой вид, как показано на рисунке.

Поскольку успешный поиск с использованием метода "Патриция" завершается во внешних узлах, среднее количество проверок битов, выполненных при случайном успешном поиске, будет равно длине внешнего пути, деленной на N. Если сформировать, как и ранее, производящую функцию $b(z)$ для внешних узлов, то эта величина может быть представлена как $b'(1)/b(1)$. *Неудачный* поиск с использованием метода "Патриция" также заканчивается во внешнем узле уровня l с вероятностью 2^{-l}, так что среднее количество проверок битов составляет $\frac{1}{2}b'(\frac{1}{2})$. Например, для случая, представленного на рис. 36, мы имеем $b(z) = 3z^3 + 8z^4 + 3z^5 + 2z^6$; таким образом, среднее число проверок битов при успешном поиске равно $4\frac{1}{4}$ и при неудачном — $3\frac{25}{32}$.

Обозначим через $h_n(z)$ "среднюю" производящую функцию $b(z)$, т. е. функцию, усредненную по всем деревьям метода "Патриция" с n внешними узлами и равномерно распределенными ключами. Рекуррентное соотношение

$$h_n(z) = 2^{1-n}\sum_k \binom{n}{k} h_k(z)\big(z + \delta_{kn}(1-z)\big), \quad h_0(z) = 0, \quad h_1(z) = 1, \tag{15}$$

по-видимому, не имеет простого решения; но, к счастью, существует простое рекуррентное соотношение для средней длины внешнего пути $h_n'(1)$, поскольку

$$\begin{aligned} h_n'(1) &= 2^{1-n}\sum_k \binom{n}{k} h_k'(1) + 2^{1-n}\sum_k \binom{n}{k} k(1-\delta_{kn}) \\ &= n - 2^{1-n}n + 2^{1-n}\sum_k \binom{n}{k} h_k'(1). \end{aligned} \tag{16}$$

Так как оно имеет вид (6), для поиска $h_n'(1)$ можно использовать ранее разработанные методы. Оказывается, $h_n'(1)$ ровно на n меньше соответствующего числа проверок битов в случайном бинарном луче. Следовательно, поле SKIP позволяет сэкономить около одной проверки бита при удачном поиске случайных данных (см. упр. 31). Избыточность же типичных реальных данных приведет к еще большей экономии.

Если попытаться найти среднее количество проверок битов в случае *неудачного* поиска методом "Патриция", получится рекуррентное соотношение

$$a_n = 1 + \frac{1}{2^n - 2}\sum_{k<n} \binom{n}{k} a_k \quad \text{для } n \geq 2; \qquad a_0 = a_1 = 0 \tag{17}$$

(здесь $a_n = \frac{1}{2}h_n'(\frac{1}{2})$). Это соотношение *не* похоже ни на одно из рассмотренных выше рекуррентных соотношений и не сводится к ним каким-либо более или

менее простым способом. Однако теория преобразований Меллина, приведенная в разделе 5.2.2, обеспечивает возможность работы с такого рода рекуррентными соотношениями. Оказывается, решение (17) содержит числа Бернулли:

$$\frac{na_{n-1}}{2} - n + 2 = \sum_{k=2}^{n-1} \binom{n}{k} \frac{B_k}{2^{k-1}-1} \quad \text{для } n \geq 2. \tag{18}$$

Эта формула, вероятно, представляет собой самый твердый асимптотический орешек, который был разгрызен в данной книге. Решение, представленное в упр. 34, является поучительным обзором многих ранее встречавшихся задач (с некоторыми нюансами).

Итоги анализа. Следующие результаты сложных математических выкладок этого раздела заслуживают особого внимания.

a) Количество узлов, которые необходимы для хранения N случайных ключей в M-арном луче с разветвлениями, прекращающимися по достижении подфайлов из $\leq s$ ключей, составляет примерно $N/(s \ln M)$. Это приближение справедливо при больших N, малых s и малых M. Поскольку узлы луча содержат M полей ссылок, всего при $s = M$ потребуется около $N/\ln M$ полей ссылок.

b) Количество цифр или символов, проверяемых в ходе случайного поиска, составляет для всех рассмотренных методов примерно $\log_M N$. При $M = 2$ различные методы анализа дают более точные приближения для количества проверок битов.

	Успешный	Неудачный
Поиск по лучу	$\lg N + 1.33275$	$\lg N - 0.10995$
Цифровой поиск по дереву	$\lg N - 1.71665$	$\lg N - 0.27395$
Поиск методом "Патриция"	$\lg N + 0.33275$	$\lg N - 0.31875$

(Эти приближения могут быть выражены фундаментальными математическими постоянными; например, 0.31875 на самом деле означает $(\ln \pi - \gamma)/\ln 2 - 1/2$).

c) "Случайные данные" в нашем случае означают, что M-ичные цифры равномерно распределены, как если бы ключи были действительными числами между 0 и 1, записанными в M-ичной системе счисления. Методы цифрового поиска не зависят от порядка, в котором ключи введены в файл (за исключением алгоритма D, который слабо чувствителен к порядку); однако они весьма чувствительны к распределению цифр. Например, если нулевые биты встречаются гораздо чаще единичных, деревья будут существенно более асимметричными по сравнению с деревьями, полученными для случайных данных (в упр. 5.2.2–53 приведен пример того, что может случиться при таком смещении данных).

УПРАЖНЕНИЯ

1. [*00*] Если дерево имеет листья, то что имеет луч?

2. [*20*] Разработайте алгоритм вставки нового ключа в M-арный луч с использованием соглашений алгоритма T.

3. [*21*] Разработайте алгоритм удаления ключа из M-арного луча с использованием соглашений алгоритма T.

▶ **4.** [*21*] Большинство из 360 элементов, приведенных в табл. 1, представляет собой пустые ссылки. Однако можно сжать таблицу до 49 элементов, перекрыв непустые и пустые

элементы, как показано ниже.

Позиция	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Элемент		(10)		WAS	THAT	(11)	OF	BE	THE	HIS	WHICH	WITH	THIS		(12)	ON	I	HE	A	OR	(2)	(3)	TO	HAD	

Позиция	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
Элемент	(4)	BUT	(5)	(6)	FOR	BY	IN	FROM	AND	NOT	(7)	HER	ARE	IS	IT	AS	AT	(8)		HAVE	(9)		YOU	

(Узлы (1), (2), ..., (12) в табл. 1 начинаются соответственно в позициях 20, 19, 3, 14, 1, 17, 1, 7, 3, 20, 18, 4 этой сжатой таблицы.)

Покажите, что, если сжатой таблицей заменить табл. 1, программа T останется работоспособной, хотя и не столь быстрой.

▸ **5.** [*M26*] (Я. Н. Патт (Y. N. Patt).) В деревьях на рис. 31 содержатся буквы каждого семейства, упорядоченные по алфавиту. Такое упорядочение не является необходимым, и, если переупорядочить узлы внутри каждого семейства перед построением дерева типа (2), поиск может ускориться. Какое переупорядочение представленного на рис. 31 дерева приведет к оптимальному с этой точки зрения результату? (Используйте частоты, приведенные на рис. 32, и найдите "лес", который позволит минимизировать время успешного поиска, когда он представлен в виде бинарного дерева.)

6. [*15*] Какое дерево цифрового поиска получится при вставке 15 четырехбитовых бинарных ключей 0001, 0010, 0011, ..., 1111 в порядке возрастания согласно алгоритму D? (Начните с ключа 0001 в корневом узле и выполните 14 вставок).

▸ **7.** [*M26*] Если 15 ключей из упр. 6 вставлены в другом порядке, может быть построено другое дерево. Какая перестановка этих ключей из всех 15! возможных будет *наихудшей* в том смысле, что она породит дерево с наибольшей длиной внутреннего пути?

8. [*20*] Рассмотрим следующие изменения в алгоритме D, ведущие к исключению из него переменной K': заменим "K'" на "K" на шаге D2 и удалим операцию "$K' \leftarrow K$" на шаге D1. Будет ли полученный в результате этих действий алгоритм корректным, и можно ли с его помощью выполнять поиск и вставку?

9. [*21*] Напишите MIX-программу для алгоритма D и сравните ее с программой 6.2.2T. Можете использовать бинарные операции, такие как SLB (бинарный сдвиг AX влево), JAE (переход при четном A) и др.; возможно, вам поможет упр. 8.

10. [*23*] Дан файл, все ключи которого представляют собой n-битовые двоичные числа, и дан аргумент поиска $K = b_1\, b_2 \ldots b_n$. Предположим, что необходимо найти максимальное значение k, при котором в файле будет содержаться ключ, начинающийся с $b_1\, b_2 \ldots b_k$. Как эффективно решить поставленную задачу, если файл представлен в виде

a) бинарного дерева поиска (алгоритм 6.2.2T);

b) бинарного луча (алгоритм T);

c) бинарного дерева цифрового поиска (алгоритм D)?

11. [*21*] Можно ли использовать неизмененный алгоритм 6.2.2D для удаления узла из дерева цифрового поиска?

12. [*25*] Будет ли случайным дерево, полученное путем удаления случайного элемента из случайного дерева цифрового поиска, которое построено с помощью алгоритма D? (См. упр. 11 и теорему·6.2.2H.)

13. [*20*] (*M-арный цифровой поиск.*) Объясните, как можно объединить алгоритмы T и D в один обобщенный алгоритм, при $M = 2$ представляющий собой алгоритм D. Какие изменения следует внести в табл. 1 для использования алгоритма при $M = 30$?

▸ **14.** [*25*] Напишите эффективный алгоритм, который может быть выполнен сразу после успешного окончания алгоритма P для нахождения *всех* мест, в которых TEXT содержит K.

15. [*28*] Разработайте эффективный алгоритм, который может применяться для построения дерева, используемого методом "Патриция", или для вставки новой ссылки на TEXT в существующее дерево. Ваш алгоритм вставки должен обращаться к массиву TEXT не более двух раз.

16. [*22*] Почему в алгоритме "Патриция" требуется, чтобы один ключ не служил началом другого?

17. [*M25*] Как выразить решение рекуррентного соотношения

$$x_0 = x_1 = 0, \qquad x_n = a_n + m^{1-n} \sum_k \binom{m}{k} (m-1)^{n-k} x_k, \qquad n \geq 2,$$

с помощью биномиальных преобразований, обобщая метод упр. 5.2.2–36?

18. [*M21*] Используя результат упр. 17, выразите решения уравнений (4) и (5) через функции U_n и V_n, аналогичные определенным в упр. 5.2.2–38.

19. [*HM23*] Найдите асимптотическое значение функции

$$K(n, s, m) = \sum_{k \geq 2} \binom{n}{k} \binom{k}{s} \frac{(-1)^k}{m^{k-1} - 1}$$

с точностью $O(1)$ при $n \to \infty$ для фиксированных значений $s \geq 0$ и $m > 1$. [Случай для $s = 0$ рассматривался в упр. 5.2.2–50, а случай для $s = 1$, $m = 2$ — в упр. 5.2.2–48.]

▸ **20.** [*M30*] Рассмотрим M-арную "лучевую" память, в которой используется последовательный поиск по достижении поддерева из s или меньшего количества лучей (алгоритм T представляет собой частный случай при $s = 1$). Примените результаты предыдущих упражнений для анализа

a) среднего количества узлов луча;

b) среднего количества проверок цифр или символов при успешном поиске;

c) среднего количества сравнений, выполняемых при успешном поиске.

Сформулируйте ответы на вопросы в виде асимптотических формул при $N \to \infty$ для фиксированных M и s; ответ для (a) должен быть дан с точностью до $O(1)$, а для (b) и (c) — до $O(N^{-1})$. [При $M = 2$ этот анализ применим к модифицированному методу обменной поразрядной сортировки, в котором подфайлы размером $\leq s$ сортируются посредством вставок.]

21. [*M25*] Сколько узлов случайного M-арного луча с N ключами имеют в качестве нулевой компоненты пустую ссылку? (Например, 9 из 12 узлов в табл. 1 имеют пустую ссылку в позиции "␣". "Случайность" в данном упражнении, как обычно, означает, что цифры ключей равномерно распределены между 0 и $M - 1$.)

22. [*M25*] Сколько узлов находится в случайном M-арном луче с N ключами на уровне l ($l = 0, 1, 2, \ldots$)?

23. [*M26*] Сколько проверок цифр выполняется в среднем в процессе неуспешного поиска в M-арном луче, содержащем N случайных ключей?

24. [*M30*] Рассмотрите M-арный луч, представленный в виде леса (см. рис. 31). Найдите точные и асимптотические выражения для

a) среднего количества узлов в лесу;

b) среднего количества присвоений "P ← RLINK(P)" в процессе случайного успешного поиска.

▸ **25.** [*M24*] Математический вывод асимптотических значений в этом разделе весьма сложен (использовалась даже теория функций комплексного переменного). Дело в том, что мы не хотели ограничиться одним членом асимптотической формулы, а вывод второго члена действительно сложен. Назначение данного упражнения — показать, что элементарных методов достаточно для вывода некоторых (пусть и ослабленных) результатов.

a) Докажите по индукции, что решение (4) удовлетворяет неравенству $A_N \le M(N-1)/(M-1)$.

b) Пусть $D_N = C_N - NH_{N-1}/\ln M$, где C_N определяется формулой (5). Докажите, что $D_N = O(N)$; следовательно, $C_N = N\log_M N + O(N)$. [*Указание.* Используйте (a) и теорему 1.2.7A.]

26. [*23*] Определите значение бесконечного произведения

$$(1-\tfrac{1}{2})(1-\tfrac{1}{4})(1-\tfrac{1}{8})(1-\tfrac{1}{16})\ldots$$

с точностью до пяти значащих цифр непосредственным вычислением. [*Указание.* См. упр. 5.1.1–16.]

27. [*HM31*] Чему с точностью до $O(1)$ равно асимптотическое значение $\bar{C}_N$ из (14)?

28. [*HM26*] Найдите асимптотическое среднее количество проверок цифр при выполнении в случайном M-арном дереве цифрового поиска при $M \ge 2$. Рассмотрите случай как успешного, так и неудачного поиска; дайте ответ с точностью до $O(N^{-1})$.

29. [*HM40*] Чему равно асимптотическое среднее количество узлов в M-арном дереве цифрового поиска, все ссылки которых пусты? (Можно было бы сэкономить память, исключая такие узлы; см. упр. 13.)

30. [*M24*] Покажите, что производящая функция алгоритма "Патриция" $h_n(z)$, определенная в (15), может быть выражена в совершенно ужасном виде:

$$n\sum_{m\ge 1} z^m \left(\sum_{\substack{a_1+\cdots+a_m=n-1 \\ a_1,\ldots,a_m\ge 1}} \binom{n-1}{a_1,\ldots,a_m} \frac{1}{(2^{a_1}-1)(2^{a_1+a_2}-1)\ldots(2^{a_1+\cdots+a_m}-1)} \right).$$

(Таким образом, если бы имелась простая формула для $h_n(z)$, можно было бы упростить это громоздкое выражение.)

31. [*M21*] Решите рекуррентное соотношение (16).

32. [*M21*] Чему равно среднее значение суммы всех полей SKIP в случайном дереве алгоритма "Патриция" с $N-1$ внутренним узлом?

33. [*M30*] Докажите, что (18) представляет собой решение рекуррентного соотношения (17). [*Указание.* Рассмотрите производящую функцию $A(z) = \sum_{n\ge 0} a_n z^n/n!$.]

34. [*HM40*] Назначение данного упражнения — поиск асимптотического поведения формулы (18).

a) Докажите, что при $n \ge 2$

$$\frac{1}{n}\sum_{2\le k<n}\binom{n}{k}\frac{B_k}{2^{k-1}-1} = \sum_{j\ge 1}\left(\frac{1^{n-1}+2^{n-1}+\cdots+(2^j-1)^{n-1}}{2^{j(n-1)}} - \frac{2^j}{n}+\frac{1}{2}\right).$$

b) Покажите, что слагаемые в правой части (a) приближенно равны $1/(e^x-1)-1/x+1/2$, где $x = n/2^j$; получающаяся при этом сумма отличается от первоначальной на $O(n^{-1})$.

c) Покажите, что

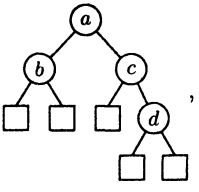
$$\frac{1}{e^x-1}-\frac{1}{x}+\frac{1}{2}=\frac{1}{2\pi i}\int_{-\frac{1}{2}-i\infty}^{-\frac{1}{2}+i\infty}\zeta(z)\Gamma(z)x^{-z}dz \qquad \text{для действительного } x>0.$$

d) Следовательно, рассматриваемая сумма равна

$$\frac{1}{2\pi i}\int_{-\frac{1}{2}-i\infty}^{-\frac{1}{2}+i\infty}\frac{\zeta(z)\Gamma(z)n^{-z}}{2^{-z}-1}\,dz+O(n^{-1}).$$

Оцените этот интеграл.

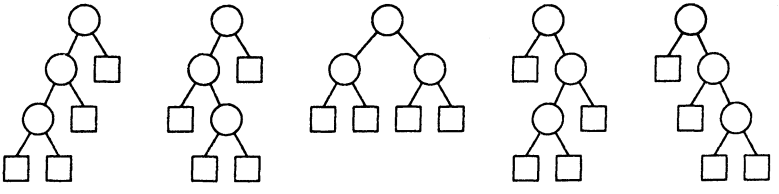
▸ **35.** [*M20*] Какова вероятность того, что дерево алгоритма "Патриция" с пятью ключами имеет вид



,

причем в полях SKIP содержатся a, b, c, d, как показано на рисунке? (Считаем, что ключи имеют независимые случайные биты; ответ должен иметь вид функции от a, b, c и d.)

36. [*M25*] Имеется пять бинарных деревьев с тремя внутренними узлами в каждом. Если рассмотреть, как часто каждое из них служит деревом поиска в различных алгоритмах при случайных данных, то получатся следующие различные вероятности.



Поиск по дереву (алгоритм 6.2.2T)	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{6}$	$\frac{1}{6}$
Цифровой поиск по дереву (алгоритм D)	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{2}$	$\frac{1}{8}$	$\frac{1}{8}$
Метод "Патриция" (алгоритм P)	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{3}{7}$	$\frac{1}{7}$	$\frac{1}{7}$

(Заметим, что дерево цифрового поиска будет сбалансировано чаще других.) В упр. 6.2.2–5 приведена формула для вероятности в случае поиска по дереву: $\prod(1/s(x))$, где произведение берется по всем внутренним узлам x, а $s(x)$ — число внутренних узлов в поддереве, корнем которого является x. Найдите аналогичные формулы для вероятности в случае (a) алгоритма D и (b) алгоритма P.

▸ **37.** [*M22*] Рассмотрите бинарное дерево, на уровне l которого имеется b_l внешних узлов. В тексте указывалось, что время неудачного поиска в деревьях цифрового поиска не связано с длиной внешнего пути $\sum lb_l$ непосредственно, а пропорционально *модифицированной длине внешнего пути* $\sum lb_l 2^{-l}$. Докажите или опровергните следующее утверждение: среди всех деревьев с N внешними узлами наименьшую модифицированную длину внешнего пути имеет то дерево, все внешние узлы которого расположены не более чем на двух смежных уровнях (см. упр. 5.3.1–20).

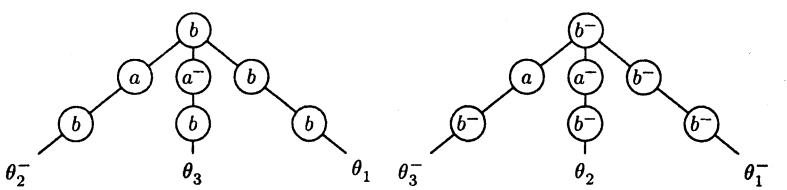
38. [*M40*] Разработайте алгоритм поиска дерева с n узлами, имеющего минимальное значение величины $\alpha\cdot$(длина внутреннего пути)$+\beta\cdot$(модифицированная длина внешнего пути), где α и β — некоторые числа (см. упр. 37).

39. [*M43*] Разработайте алгоритм поиска оптимальных деревьев цифрового поиска, аналогичных оптимальным бинарным деревьям поиска, которые рассматривались в разделе 6.2.2.

▸ **40.** [*25*] Пусть $a_0\,a_1\,a_2\ldots$ — это периодическая бинарная последовательность, в которой $a_{N+k} = a_k$ для всех $k \geq 0$. Покажите, что любая фиксированная последовательность такого типа может быть представлена в $O(N)$ ячейках памяти так, что следующая операция может быть выполнена за $O(N)$ шагов. Определите для данной цепочки битов $b_0\,b_1\ldots b_{n-1}$, сколько раз она встречается в периоде (т. е. найдите, сколько существует таких значений p, при которых $0 \leq p < N$ и $b_k = a_{p+k}$ для $0 \leq k < n$). Предполагается, что каждая ячейка памяти достаточно велика, чтобы содержать любое целое число от 0 до N. [*Указание.* См. упр. 14.]

41. [*HM28*] (Приложение к теории групп.) Пусть G — свободная группа над символами $\{a_1,\ldots,a_n\}$, т. е. множество всех строк $\alpha = b_1\ldots b_r$, где каждый b_i представляет собой либо a_j, либо a_j^- и не имеется смежных пар $a_j a_j^-$ или $a_j^- a_j$. Обратным к α элементом является $b_r^-\ldots b_1^-$. Перемножим две такие строки путем их конкатенации и сокращения смежных обратных пар. Пусть H — подгруппа группы G, порожденная строками $\{\beta_1,\ldots,\beta_p\}$, т. е. множество всех элементов G, которые могут быть записаны в виде произведений β и обратных им элементов. Согласно широкоизвестной теореме Якоба Нильсена (Jakob Nielsen) [см. Marshall Hall, *The Theory of Groups* (New York: Macmillan, 1959), Ch. 7] всегда можно найти образующие $\theta_1,\ldots,\theta_m$ для H, $m \leq p$, обладающие тем свойством, что средний символ θ_i (или, по крайней мере, один из центральных символов θ_i, если его длина четна) никогда не сокращается в выражениях $\theta_i\theta_j^e$ или $\theta_j^e\theta_i$, $e = \pm 1$, за исключением случая $j = i$ и $e = -1$. Из этого свойства следует, что существует простой алгоритм проверки принадлежности некоторого элемента G подгруппе H: запишем $2m$ ключей $\theta_1,\ldots,\theta_m$, $\theta_1^-,\ldots,\theta_m^-$ в дерево, ориентированное на поиск по символам, с помощью $2n$ букв $a_1,\ldots,a_n$, $a_1^-,\ldots,a_n^-$. Пусть $\alpha = b_1\ldots b_r$ — данный элемент из G; если $r = 0$, то α, разумеется, принадлежит H. В противном случае ищем α, определяя самый длинный префикс $b_1\ldots b_k$, который соответствует ключу. Если имеется несколько ключей, начинающихся с $b_1\ldots b_k$, то α не принадлежит H. Иначе обозначим этот единственный ключ через $b_1\ldots b_k c_1\ldots c_l = \theta_i^e$ и заменим α ключом $\theta_i^{-e}\alpha = c_l^-\ldots c_1^- b_{k+1}\ldots b_r$. Если новое значение α длиннее старого (т. е. если $l > k$), α не принадлежит H; в противном случае повторяем процесс с новым значением α. Из свойства Нильсена вытекает конечность этого алгоритма. Если α сводится к пустой строке, можно восстановить исходное представление α в виде произведений θ.

Например, пусть $\{\theta_1,\theta_2,\theta_3\} = \{bbb,\ b^-a^-b^-,\ ba^-b\}$ и $\alpha = bbabaab$. Лес



вместе с описанным алгоритмом позволяет вывести, что $\alpha = \theta_1\theta_3^-\theta_1\theta_3^-\theta_2^-$. Реализуйте этот алгоритм, используя в качестве входных данных вашей программы θ_i.

42. [*23*] (*Сжатие спереди и сзади.*) Если множество бинарных ключей используется в качестве индекса для разделения большого файла, можно не хранить полные ключи. Например, 16 ключей, показанных на рис. 34, могут быть урезаны справа так, чтобы

оставалось достаточное количество цифр для их однозначной идентификации: 0000, 0001, 00100, 00101, 010, ..., 1110001. Такие урезанные ключи могут использоваться для разделения файла на 17 частей, и, например, в пятой части могут содержаться все ключи, начинающиеся с 0011 или 010, а в последней части — ключи, начинающиеся с 111001, 11101 или 1111. Урезанные ключи можно представить в более компактной форме, если опустить все лидирующие цифры, совпадающие с цифрами предыдущего ключа: 0000, ◇◇◇1, ◇◇100, ◇◇◇◇1, ◇10, ..., ◇◇◇◇◇◇1. Бит, следующий за символом "◇", всегда равен 1, поэтому он также может быть опущен. В большом файле будет содержаться много символов "◇", и мы должны хранить только их количество и значения последующих битов.

Покажите, что общее количество битов в сжатом файле с исключенными символами "◇" и последующим одним битом, всегда равно количеству узлов в бинарном луче с этими ключами.

(Следовательно, среднее общее количество таких битов во всем индексе составляет около $N/\ln 2$, 1.44 бит на ключ. Этот способ сжатия был показан автору Э. Геллером (A. Heller) и Р. Л. Джонсеном (R. L. Johnsen). Возможно и дальнейшее сжатие, поскольку нам необходимо только представление структуры луча; см. теорему 2.3.1A.)

43. [*HM42*] Проанализируйте высоту случайного M-арного луча, имеющего N ключей и параметр обрезки s, как в упр. 20 (когда $s = 1$, эта величина представляет собой длину самого длинного общего префикса случайных слов длиной N в M-арном алфавите).

▸ **44.** [*30*] (Дж. Л. Бентли (J. L. Bentley) и Р. Седгевик (R. Sedgewick).) Исследуйте тернарное представление лучей, при котором левая и правая ссылки соответствуют горизонтальным ветвям (2), в то время как средняя ссылка соответствует ветви, идущей вниз.

▸ **45.** [*M25*] Если семь ключей, которые показаны на рис. 33, вставлены в случайном порядке с помощью алгоритма, описанного в упр. 15, то чему равна вероятность получения показанного дерева?