Глава 6. Пирамидальная сортировка

В этой главе описывается еще один алгоритм сортировки, а именно — пирамидальная сортировка. Время работы этого алгоритма, как и время работы сортировки слиянием (и в отличие от времени работы сортировки вставкой), равно $O(n \lg n)$. Как и сортировка методом вставок, и в отличие от сортировки слиянием, пирамидальная сортировка выполняется без привлечения дополнительной памяти: в любой момент времени требуется память для хранения вне массива только некоторого постоянного количества элементов. Таким образом, в пирамидальной сортировке сочетаются наилучшие особенности двух рассмотренных ранее алгоритмов сортировки.

В ходе рассмотрения пирамидальной сортировки мы познакомимся с еще одним методом разработки алгоритмов, а именно — с использованием специализированных структур данных для управления информацией в ходе выполнения алгоритма. В рассматриваемом случае такая структура данных называется *пирамидой* (heap) и может оказаться полезной не только при пирамидальной сортировке, но и при создании эффективной очереди с приоритетами. В последующих главах эта структура данных появится снова.

Изначально термин "heap" использовался в контексте пирамидальной сортировки (heapsort), но позже его основной смысл изменился, и он стал обозначать память со сборкой мусора, в частности в языках программирования Lisp и Java (и переводиться как "куча"). Однако в данной книге термину "heap" (который здесь переводится как "пирамида") возвращен его первоначальный смысл.[1]

6.1. Пирамиды

Структура данных ***(бинарная) пирамида*** представляет собой объект-массив, который можно рассматривать как почти полное бинарное дерево (см. раздел Б.5.3), как показано на рис. 6.1. Каждый узел этого дерева соответствует

[1] Впрочем, поскольку перевод на русский язык термина *heap* в контексте структур данных и в контексте управления памятью разный, никаких проблем неоднозначности у русскоязычного читателя возникнуть не должно. — *Примеч. пер.*

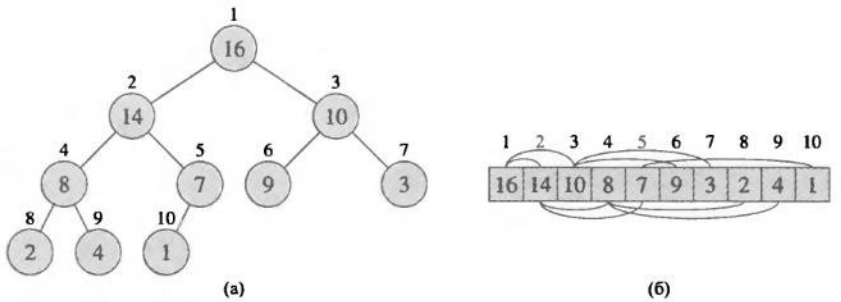

Рис. 6.1. Невозрастающая пирамида, представленная в виде **(а)** бинарного дерева и в виде **(б)** массива. Числа внутри кружков в каждом узле показывают значение, хранящееся в этом узле. Числа над узлами соответствуют индексам в массиве. Линии над и под элементами массива показывают отношения между родительскими и дочерними узлами; родительские узлы всегда находятся левее дочерних. Высота дерева равна трем; узел с индексом 4 (и значением 8) имеет высоту 1.

элементу массива. Дерево полностью заполнено на всех уровнях, за исключением, возможно, наинизшего, который заполняется слева направо. Массив A, представляющий пирамиду, является объектом с двумя атрибутами: $A.length$, который, как обычно, дает количество элементов в массиве, и $A.heap\text{-}size$, который указывает, сколько элементов пирамиды содержится в массиве A. Т.е. хотя $A[1..A.length]$ может содержать некоторые числа, только элементы подмассива $A[1..A.heap\text{-}size]$, где $0 \leq A.heap\text{-}size \leq A.length$, являются корректными элементами пирамиды. Корнем дерева является $A[1]$, а для заданного индекса i узла можно легко вычислить индексы его родительского, левого и правого дочерних узлов.

```
PARENT(i)
1  return ⌊i/2⌋

LEFT(i)
1  return 2i

RIGHT(i)
1  return 2i + 1
```

На большинстве компьютеров операция $2i$ в процедуре LEFT выполняется с помощью одной команды процессора путем битового сдвига числа i на один бит влево. Аналогично операция $2i + 1$ в процедуре RIGHT также выполняется очень быстро, путем сдвига бинарного представления числа i на одну позицию влево, а затем младший бит устанавливается равным 1. Процедура PARENT выполняется путем сдвига числа i на один бит вправо. Эффективные программы пирамидальной сортировки часто реализуют эти процедуры как макросы или встраиваемые процедуры.

Различают два вида бинарных пирамид: неубывающие и невозрастающие. В пирамидах обоих видов значения, расположенные в узлах, удовлетворяют ***свойству пирамиды*** (heap property), являющемуся отличительной чертой пирамиды

того или иного вида. ***Свойство невозрастающих пирамид*** (max-heap property) заключается в том, что для каждого отличного от корневого узла с индексом i выполняется неравенство

$$A[\text{PARENT}(i)] \geq A[i] ,$$

т.е. значение узла не превышает значение родительского по отношению к нему узла. Таким образом, в невозрастающей пирамиде самый большой элемент находится в корне дерева, а значения узлов поддерева, берущего начало в каком-то элементе, не превышают значения самого этого элемента. Принцип организации ***неубывающей пирамиды*** (min-heap) прямо противоположный. ***Свойство неубывающих пирамид*** (min-heap property) заключается в том, что для всех отличных от корневого узлов с индексом i выполняется неравенство

$$A[\text{PARENT}(i)] \leq A[i] .$$

Таким образом, наименьший элемент такой пирамиды находится в ее корне.

В алгоритме пирамидальной сортировки используются невозрастающие пирамиды. Неубывающие пирамиды часто реализуют очереди с приоритетами (этот вопрос обсуждается в разделе 6.5). Для каждого приложения будет указано, с пирамидами какого вида мы будем иметь дело — с неубывающими или невозрастающими. При описании свойств, общих как для неубывающих, так и для невозрастающих пирамид, будет использоваться общий термин "пирамида".

Рассматривая пирамиду как дерево, определим ***высоту*** ее узла как число ребер в самом длинном простом нисходящем пути от этого узла к какому-то из листьев дерева. Высота пирамиды определяется как высота ее корня. Поскольку n-элементная пирамида строится по принципу полного бинарного дерева, ее высота равна $\Theta(\lg n)$ (см. упр. 6.1.2). Мы увидим, что время выполнения основных операций в пирамиде приблизительно пропорционально высоте дерева, и, таким образом, эти операции требуют для работы время $O(\lg n)$. В остальных разделах этой главы представлено несколько базовых процедур и продемонстрировано их использование в алгоритме сортировки и в структуре данных очереди с приоритетами.

- Процедура MAX-HEAPIFY выполняется за время $O(\lg n)$ и служит для поддержки свойства невозрастания пирамиды.
- Время выполнения процедуры BUILD-MAX-HEAP увеличивается с ростом количества элементов линейно. Эта процедура предназначена для создания невозрастающей пирамиды из неупорядоченного входного массива.
- Процедура HEAPSORT выполняется за время $O(n \lg n)$ и сортирует массив без привлечения дополнительной памяти.
- Процедуры MAX-HEAP-INSERT, HEAP-EXTRACT-MAX, HEAP-INCREASE-KEY и HEAP-MAXIMUM выполняются за время $O(\lg n)$ и позволяют использовать пирамиду для реализации очереди с приоритетами.

Упражнения

6.1.1

Чему равно минимальное и максимальное количества элементов в пирамиде высотой h?

6.1.2

Покажите, что n-элементная пирамида имеет высоту $\lfloor \lg n \rfloor$.

6.1.3

Покажите, что в любом поддереве невозрастающей пирамиды корень этого поддерева содержит наибольшее значение среди узлов поддерева.

6.1.4

Где в невозрастающей пирамиде может находиться наименьший ее элемент, если все элементы различаются по величине?

6.1.5

Является ли массив с отсортированными элементами неубывающей пирамидой?

6.1.6

Является ли последовательность значений $\langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12 \rangle$ невозрастающей пирамидой?

6.1.7

Покажите, что если n-элементную пирамиду представить в виде массива, то ее листьями будут элементы с индексами $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$.

6.2. Поддержка свойства пирамиды

Для поддержки свойства невозрастающей пирамиды мы вызываем процедуру MAX-HEAPIFY. Ее входными данными являются массив A и индекс i в этом массиве. При вызове процедуры MAX-HEAPIFY предполагается, что бинарные деревья с корнями LEFT(i) и RIGHT(i) представляют собой невозрастающие пирамиды, но $A[i]$ может быть меньше, чем значения в дочерних узлах (таким образом, нарушая свойство невозрастающей пирамиды). Процедура MAX-HEAPIFY "сплавляет" значение $A[i]$ вниз по невозрастающей пирамиде, так, что поддерево с корневым элементом с индексом i подчиняется свойству невозрастающей пирамиды.

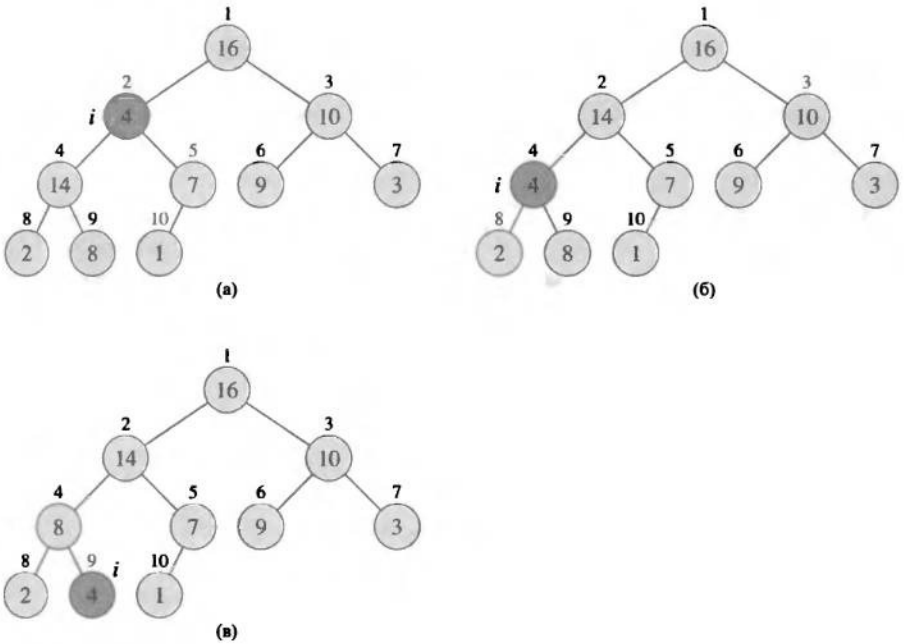

Рис. 6.2. Работа процедуры MAX-HEAPIFY$(A, 2)$, где $A.heap\text{-}size = 10$. **(а)** Исходная конфигурация, в которой значение $A[2]$ в узле $i = 2$ нарушает свойство невозрастающей пирамиды, поскольку оно меньше, чем каждое из дочерних значений. Свойство невозрастающей пирамиды восстанавливается для узла 2 в части **(б)** путем обмена $A[2]$ с $A[4]$, который при этом нарушает свойство невозрастающей пирамиды для узла 4. В рекурсивном вызове MAX-HEAPIFY$(A, 4)$ значение $i = 4$. После обмена $A[4]$ с $A[9]$, как показано в части **(в)**, ситуация в узле 4 исправляется, и рекурсивный вызов MAX-HEAPIFY$(A, 9)$ не вносит никаких изменений в полученную структуру данных.

```
MAX-HEAPIFY(A, i)
 1  l = LEFT(i)
 2  r = RIGHT(i)
 3  if l ≤ A.heap-size и A[l] > A[i]
 4       largest = l
 5  else largest = i
 6  if r ≤ A.heap-size и A[r] > A[largest]
 7       largest = r
 8  if largest ≠ i
 9       Обменять A[i] и A[largest]
10       MAX-HEAPIFY(A, largest)
```

На рис. 6.2 показана работа процедуры MAX-HEAPIFY. На каждом шаге определяется больший из элементов $A[i]$, $A[\text{LEFT}(i)]$ и $A[\text{RIGHT}(i)]$, и его индекс сохраняется в переменной *largest*. Если наибольшим оказывается $A[i]$, то поддерево с корнем i уже представляет собой корректную невозрастающую пирамиду и процедура завершает работу. В противном случае наибольшим оказывается один из двух дочерних элементов, и процедура выполняет обмен $A[i]$ с $A[largest]$, что

приводит к тому, что для узла i и его дочерних узлов выполняется свойство невозрастающей пирамиды. Однако теперь исходное значение $A[i]$ оказывается в узле с индексом *largest*, так что поддерево с корнем *largest* может нарушать свойство невозрастающей пирамиды. Следовательно, необходимо рекурсивно вызвать процедуру MAX-HEAPIFY уже для этого дерева.

Для работы процедуры MAX-HEAPIFY на поддереве размером n с корнем в заданном узле i требуется время $\Theta(1)$, необходимое для исправления отношений между элементами $A[i]$, $A[\text{LEFT}(i)]$ и $A[\text{RIGHT}(i)]$, плюс время работы этой процедуры с поддеревом, корень которого находится в одном из дочерних узлов узла i. Размер каждого из таких дочерних поддеревьев не превышает величину $2n/3$, причем наихудший случай осуществляется, когда последний уровень заполнен наполовину. Таким образом, время работы процедуры MAX-HEAPIFY описывается рекуррентным соотношением

$$T(n) \leq T(2n/3) + \Theta(1) .$$

Решение этого рекуррентного соотношения, согласно случаю 2 основной теоремы (теорема 4.1), имеет вид $T(n) = O(\lg n)$. По-другому время работы процедуры MAX-HEAPIFY с узлом, который находится на высоте h, можно выразить как $O(h)$.

Упражнения

6.2.1

Воспользовавшись рис. 6.2 в качестве образца, проиллюстрируйте работу процедуры MAX-HEAPIFY$(A, 3)$ с массивом $A = \langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$.

6.2.2

Используя в качестве отправной точки процедуру MAX-HEAPIFY, напишите псевдокод процедуры MIN-HEAPIFY(A, i), которая выполняет соответствующие действия над неубывающей пирамидой. Каково время работы процедуры MIN-HEAPIFY по сравнению со временем работы процедуры MAX-HEAPIFY?

6.2.3

Как влияет на вызов процедуры MAX-HEAPIFY(A, i) ситуация, когда элемент $A[i]$ больше, чем его дочерние элементы?

6.2.4

К чему приведет вызов процедуры MAX-HEAPIFY(A, i) в случае $i > A.\mathit{heap\text{-}size}/2$?

6.2.5

Код процедуры MAX-HEAPIFY достаточно рационален, если не считать рекурсивного вызова в строке 10, из-за которого некоторые компиляторы могут сгенерировать неэффективный код. Напишите эффективную процедуру MAX-HEAPIFY, в которой вместо рекурсивного вызова использовалась бы итеративная управляющая конструкция (цикл).

6.2.6

Покажите, что в наихудшем случае время работы процедуры MAX-HEAPIFY на пирамиде размером n равно $\Omega(\lg n)$. (*Указание:* в пирамиде с n узлами присвойте узлам такие значения, чтобы процедура MAX-HEAPIFY рекурсивно вызывалась в каждом узле, расположенном на простом пути от корня до листа.)

6.3. Построение пирамиды

Процедуру MAX-HEAPIFY можно использовать в восходящем направлении для того, чтобы преобразовать массив $A[1..n]$, где $n = A.length$, в невозрастающую пирамиду. В соответствии с упр. 6.1.7 элементы в подмассиве $A[(\lfloor n/2 \rfloor + 1)..n]$ представляют собой листья дерева, поэтому каждый из них можно считать одноэлементной пирамидой, с которой можно начать процесс построения. Процедура BUILD-MAX-HEAP проходит по остальным узлам и для каждого из них выполняет процедуру MAX-HEAPIFY.

```
BUILD-MAX-HEAP(A)
1  A.heap-size = A.length
2  for i = ⌊A.length/2⌋ downto 1
3      MAX-HEAPIFY(A, i)
```

На рис. 6.3 показан пример работы процедуры BUILD-MAX-HEAP.

Чтобы показать корректность работы процедуры BUILD-MAX-HEAP, воспользуемся следующим инвариантом цикла.

> В начале каждой итерации цикла **for** в строках 2 и 3 каждый узел node $i+1, i+2, \ldots, n$ является корнем невозрастающей пирамиды.

Необходимо показать, что этот инвариант справедлив перед первой итерацией цикла, сохраняется при каждой итерации и позволяет продемонстрировать корректность алгоритма после его завершения.

Инициализация. Перед первой итерацией цикла $i = \lfloor n/2 \rfloor$. Все узлы с индексами $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$ являются листьями, поэтому каждый из них является корнем тривиальной невозрастающей пирамиды.

Сохранение. Чтобы убедиться, что каждая итерация сохраняет инвариант цикла, заметим, что узлы, дочерние по отношению к узлу i, имеют номера, которые больше i. В соответствии с инвариантом цикла оба эти узла являются корнями невозрастающих пирамид. Это именно то условие, которое требуется для вызова процедуры MAX-HEAPIFY(A, i), чтобы преобразовать узел с индексом i в корень невозрастающей пирамиды. Кроме того, при вызове процедуры MAX-HEAPIFY сохраняется свойство пирамиды, заключающееся в том, что все узлы с индексами $i+1, i+2, \ldots, n$ являются корнями невозрастающих пирамид. Уменьшение индекса i в цикле **for** обеспечивает выполнение инварианта цикла для следующей итерации.

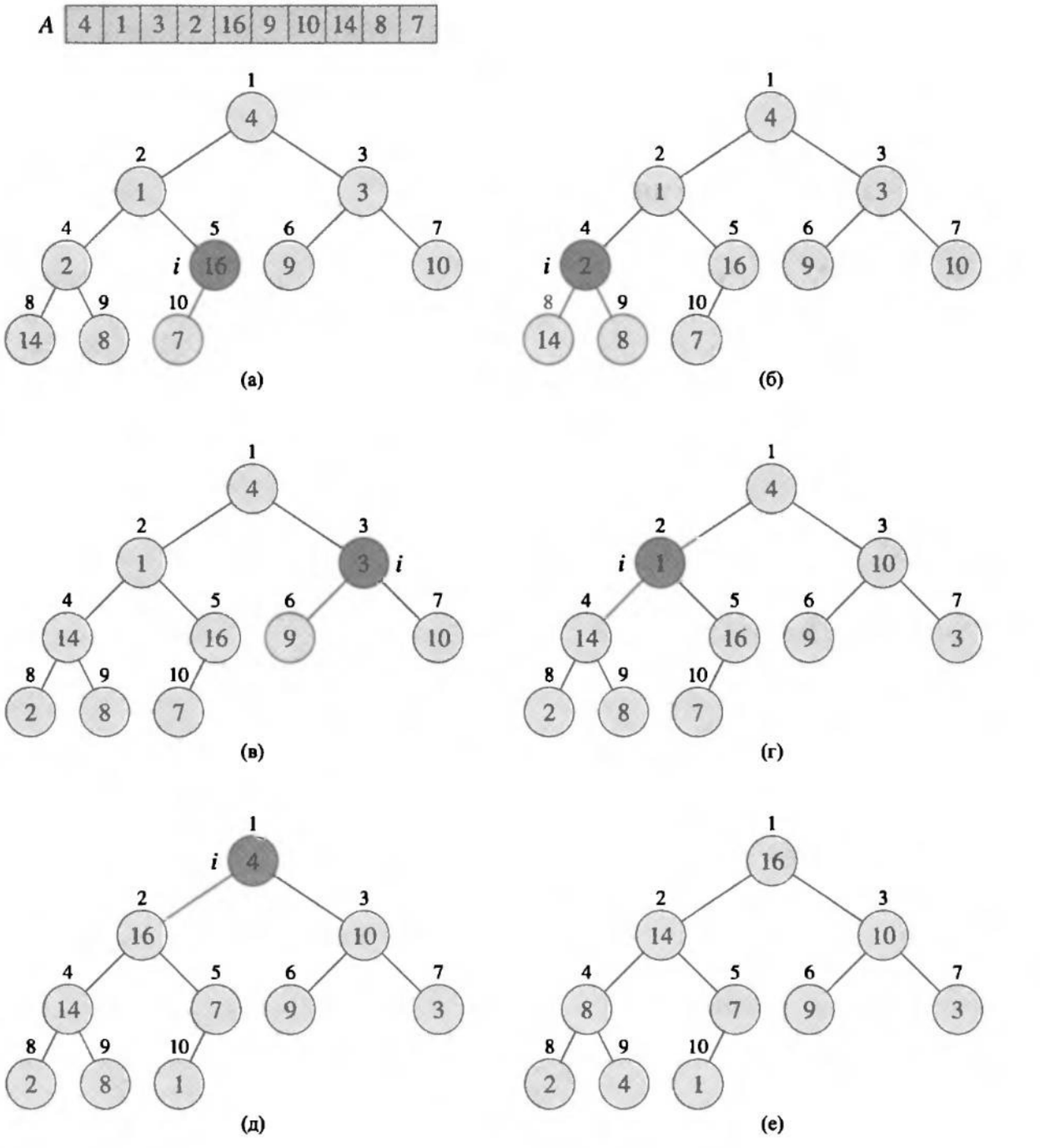

Рис. 6.3. Работа процедуры BUILD-MAX-HEAP. Показаны структуры данных перед вызовом MAX-HEAPIFY в строке 3 процедуры BUILD-MAX-HEAP. **(а)** 10-элементный входной массив A и бинарное дерево, которое он представляет. На рисунке показано, что индекс цикла i перед вызовом MAX-HEAPIFY(A, i) указывает на узел 5. **(б)** Полученная в результате структура данных. Индекс цикла i в следующей итерации указывает на узел 4. **(в)–(д)** Последовательные итерации цикла **for** в BUILD-MAX-HEAP. Обратите внимание, что, когда для некоторого узла вызывается процедура MAX-HEAPIFY, два поддерева этого узла представляют собой невозрастающие пирамиды. **(е)** Невозрастающая пирамида по завершении работы BUILD-MAX-HEAP.

Завершение. После завершения цикла $i = 0$. В соответствии с инвариантом цикла все узлы с индексами 1, 2, ..., n являются корнями невозрастающих пирамид. В частности, таким корнем является узел 1.

Простую верхнюю оценку времени работы процедуры BUILD-MAX-HEAP можно получить следующим образом. Каждый вызов процедуры MAX-HEAPIFY имеет стоимость $O(\lg n)$, а всего имеется $O(n)$ таких вызовов. Таким образом, время работы алгоритма равно $O(n \lg n)$. Эта верхняя граница, будучи корректной, не является асимптотически точной.

Чтобы получить более точную оценку, заметим, что время работы MAX-HEAPIFY в том или ином узле зависит от высоты этого узла, и при этом большинство узлов расположено на малой высоте. При более тщательном анализе принимается во внимание тот факт, что высота n-элементной пирамиды равна $\lfloor \lg n \rfloor$ (упр. 6.1.2) и что на любом уровне на высоте h содержится не более $\lceil n/2^{h+1} \rceil$ узлов (упр. 6.3.3).

Время работы процедуры MAX-HEAPIFY при ее вызове для работы с узлом, который находится на высоте h, равно $O(h)$, так что общая стоимость процедуры BUILD-MAX-HEAP ограничена сверху значением

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right) .$$

Последняя сумма вычисляется путем подстановки $x = 1/2$ в формулу (A.8), что дает

$$\begin{aligned} \sum_{h=0}^{\infty} \frac{h}{2^h} &= \frac{1/2}{(1 - 1/2)^2} \\ &= 2 . \end{aligned}$$

Таким образом, время работы процедуры BUILD-MAX-HEAP имеет верхнюю границу

$$\begin{aligned} O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right) &= O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h} \right) \\ &= O(n) . \end{aligned}$$

Следовательно, построить невозрастающую пирамиду из неупорядоченного массива можно за линейное время.

Неубывающая пирамида строится с помощью процедуры BUILD-MIN-HEAP, идентичной процедуре BUILD-MAX-HEAP, лишь вызов MAX-HEAPIFY в строке 3 заменяется вызовом MIN-HEAPIFY (упр. 6.2.2). Процедура BUILD-MIN-HEAP строит неубывающую пирамиду из неупорядоченного массива за линейное время.

Упражнения

6.3.1

Воспользовавшись в качестве образца рис. 6.3, проиллюстрируйте работу процедуры BUILD-MAX-HEAP над входным массивом $A = \langle 5, 3, 17, 10, 84, 19, 6, 22, 9 \rangle$.

6.3.2

Почему индекс цикла i в строке 2 процедуры BUILD-MAX-HEAP убывает от $\lfloor A.length/2 \rfloor$ до 1, а не возрастает от 1 до $\lfloor A.length/2 \rfloor$?

6.3.3

Покажите, что в любой n-элементной пирамиде на высоте h находится не более $\lceil n/2^{h+1} \rceil$ узлов.

6.4. Алгоритм пирамидальной сортировки

Работа алгоритма пирамидальной сортировки начинается с вызова процедуры BUILD-MAX-HEAP для построения невозрастающей пирамиды из входного массива $A[1..n]$, где $n = A.length$. Поскольку наибольший элемент массива находится в корне $A[1]$, его можно поместить в корректную окончательную позицию в отсортированном массиве, поменяв его местами с элементом $A[n]$. Выбросив из пирамиды узел n (путем уменьшения на единицу величины $A.heap\text{-}size$, мы обнаружим, что дочерние поддеревья корня остаются корректными невозрастающими пирамидами, и только корень может нарушать свойство невозрастающей пирамиды. Для восстановления этого свойства достаточно вызвать процедуру MAX-HEAPIFY$(A, 1)$, после чего подмассив $A[1..n-1]$ превратится в невозрастающую пирамиду. Затем алгоритм пирамидальной сортировки повторяет описанный процесс для невозрастающих пирамид размером от $n-1$ до 2. (См. упр. 6.4.2, посвященное точной формулировке инварианта цикла данного алгоритма.)

```
HEAPSORT(A)
1  BUILD-MAX-HEAP(A)
2  for i = A.length downto 2
3      Обменять A[1] с A[i]
4      A.heap-size = A.heap-size − 1
5      MAX-HEAPIFY(A, 1)
```

На рис. 6.4 приведен пример работы процедуры HEAPSORT, после того как в строке 1 была построена начальная невозрастающая пирамида. На рисунке показана невозрастающая пирамида перед первой итерацией цикла **for** в строках 2–5 и после каждой из итераций.

Время работы процедуры HEAPSORT равно $O(n \lg n)$, поскольку вызов процедуры BUILD-MAX-HEAP требует времени $O(n)$, а каждый из $n - 1$ вызовов процедуры MAX-HEAPIFY — времени $O(\lg n)$.

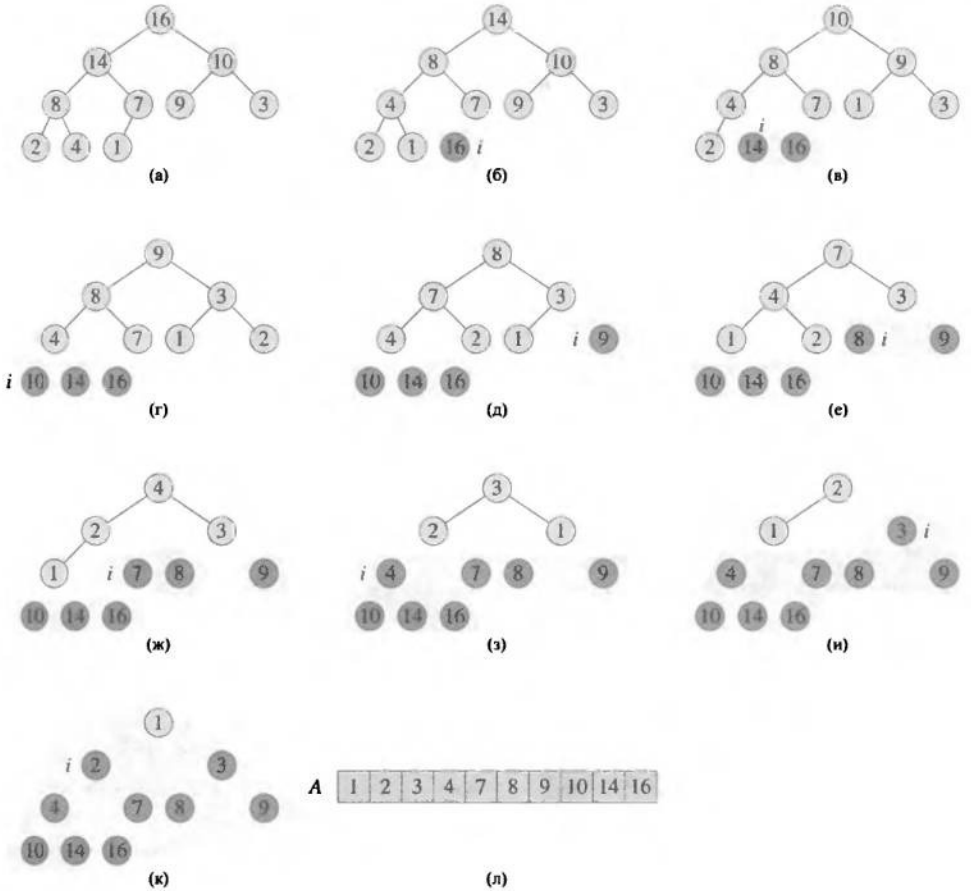

Рис. 6.4. Работа процедуры HEAPSORT. **(а)** Невозрастающая пирамида сразу после построения процедурой BUILD-MAX-HEAP в строке 1. **(б)–(к)** Невозрастающая пирамида после каждого вызова MAX-HEAPIFY в строке 5; указано также значение i в этот момент. В невозрастающей пирамиде остаются только светлые узлы. **(л)** Выходной отсортированный массив A.

Упражнения

6.4.1

Воспользовавшись в качестве образца рис. 6.4, проиллюстрируйте работу процедуры HEAPSORT над входным массивом $A = \langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$.

6.4.2

Докажите корректность процедуры HEAPSORT с помощью следующего инварианта цикла.

> В начале каждой итерации цикла **for** в строках 2–5 подмассив $A[1\,..\,i]$ представляет собой невозрастающую пирамиду, содержащую i наименьших элементов массива $A[1\,..\,n]$, а в подмассиве $A[i{+}1\,..\,n]$ содержатся $n{-}i$ наибольших элементов массива $A[1\,..\,n]$ в отсортированном состоянии.

6.4.3

Чему равно время работы процедуры HEAPSORT с массивом A длиной n, в котором элементы отсортированы и расположены в порядке возрастания? В порядке убывания?

6.4.4

Покажите, что время работы процедуры HEAPSORT в наихудшем случае составляет $\Omega(n \lg n)$.

6.4.5 ⋆

Покажите, что, когда все элементы различны, время работы процедуры HEAPSORT в наилучшем случае равно $\Omega(n \lg n)$.

6.5. Очереди с приоритетами

Пирамидальная сортировка — превосходный алгоритм, однако качественная реализация алгоритма быстрой сортировки, представленного в главе 7, на практике обычно превосходит по производительности пирамидальную сортировку. Тем не менее структура данных, использующаяся при пирамидальной сортировке, сама по себе имеет множество применений. В этом разделе представлено одно из наиболее популярных применений пирамид — в качестве эффективных очередей с приоритетами. Как и пирамиды, очереди с приоритетами бывают двух видов: невозрастающие и неубывающие. Мы рассмотрим процесс реализации невозрастающих очередей с приоритетами, которые основаны на невозрастающих пирамидах; в упр. 6.5.3 требуется написать процедуры для неубывающих очередей с приоритетами.

Очередь с приоритетами (priority queue) представляет собой структуру данных, предназначенную для обслуживания множества S, с каждым элементом которого связано определенное значение, называющееся ***ключом*** (key). В ***невозрастающей очереди с приоритетами*** поддерживаются следующие операции.

INSERT(S, x) вставляет элемент x в множество S, что эквивалентно операции $S = S \cup \{x\}$.

MAXIMUM(S) возвращает элемент множества S с наибольшим ключом.

EXTRACT-MAX(S) удаляет и возвращает элемент множества S с наибольшим ключом.

INCREASE-KEY(S, x, k) увеличивает значение ключа элемента x до нового значения k, которое предполагается не меньшим значения текущего ключа элемента x.

Среди прочих областей применения невозрастающих очередей — планирование заданий на совместно используемом компьютере. Очередь позволяет следить за заданиями, которые подлежат выполнению, и за их относительными приоритетами. Если задание прервано или завершило свою работу, планировщик выбирает

из очереди с помощью операции EXTRACT-MAX следующее задание с наибольшим приоритетом. В очередь в любое время можно добавить новое задание, воспользовавшись операцией INSERT.

Аналогично в ***неубывающей очереди с приоритетами*** поддерживаются операции INSERT, MINIMUM, EXTRACT-MIN и DECREASE-KEY. Очереди этого вида могут использоваться в моделировании систем, управляемых событиями. В роли элементов очереди в таком случае выступают моделируемые события, для каждого из которых сопоставляется время осуществления, играющее роль ключа. События должны моделироваться последовательно, согласно времени их наступления, поскольку процесс моделирования может вызвать генерацию других событий, которые нужно будет моделировать позже. Моделирующая программа выбирает очередное моделируемое событие с помощью операции EXTRACT-MIN. Когда инициируются новые события, они помещаются в очередь с помощью процедуры INSERT. В главах 23 и 24 нам предстоит познакомиться и с другими случаями применения неубывающих очередей с приоритетами, когда особо важной становится роль операции DECREASE-KEY.

Не удивительно, что приоритетную очередь можно реализовать с помощью пирамиды. В каждом отдельно взятом приложении, например в планировщике заданий, или при моделировании событий элементы очереди с приоритетами соответствуют объектам, с которыми работает это приложение. Часто возникает необходимость определить, какой из объектов приложения отвечает тому или иному элементу очереди, или наоборот. Если очередь с приоритетами реализуется с помощью пирамиды, то в каждом элементе пирамиды приходится хранить ***идентификатор*** (handle) соответствующего объекта приложения. То, каким будет конкретный вид этого идентификатора (таким, как указатель или целочисленный индекс), зависит от приложения. В каждом объекте приложения точно так же необходимо хранить идентификатор соответствующего элемента пирамиды. В данной книге таким идентификатором, как правило, будет индекс массива. Поскольку в ходе операций над пирамидой ее элементы изменяют свое расположение в массиве, при перемещении элемента пирамиды необходимо также обновлять значение индекса в соответствующем объекте приложения. Так как детали доступа к объектам приложения сильно зависят от самого приложения и его реализации, мы не станем останавливаться на этом вопросе. Ограничимся лишь замечанием, что на практике необходима организация надлежащей обработки идентификаторов.

Теперь рассмотрим, как реализовать операции в невозрастающей очереди с приоритетами. Процедура HEAP-MAXIMUM реализует операцию MAXIMUM за время $\Theta(1)$.

```
HEAP-MAXIMUM(A)
1  return A[1]
```

Процедура HEAP-EXTRACT-MAX реализует операцию EXTRACT-MAX. Она похожа на тело цикла **for** (строки 3–5) процедуры HEAPSORT.

```
HEAP-EXTRACT-MAX(A)
1  if A.heap-size < 1
2      error "Очередь пуста"
3  max = A[1]
4  A[1] = A[A.heap-size]
5  A.heap-size = A.heap-size - 1
6  MAX-HEAPIFY(A, 1)
7  return max
```

Время работы процедуры HEAP-EXTRACT-MAX равно $O(\lg n)$, поскольку она выполняет только константное количество работы перед вызовом процедуры MAX-HEAPIFY, время работы которой — $O(\lg n)$.

Процедура HEAP-INCREASE-KEY реализует операцию INCREASE-KEY. Элемент очереди с приоритетами, ключ которого подлежит увеличению, идентифицируется в массиве с помощью индекса i. Сначала процедура обновляет ключ элемента $A[i]$. Поскольку это действие может нарушить свойство невозрастающей пирамиды, после этого процедура проходит путь от измененного узла к корню в поисках надлежащего места для нового ключа. Эта операция напоминает операцию, реализованную в цикле процедуры INSERTION-SORT из раздела 2.1 (строки 5–7). В процессе прохода выполняется сравнение текущего элемента с родительским. Если оказывается, что ключ текущего элемента превышает значение ключа родительского элемента, то происходит обмен ключами элементов и процедура продолжает свою работу на более высоком уровне. В противном случае процедура прекращает работу, поскольку ей удалось восстановить свойство невозрастающих пирамид. (Точная формулировка соответствующего инварианта цикла приведена в упр. 6.5.5.)

```
HEAP-INCREASE-KEY(A, i, key)
1  if key < A[i]
2      error "Новый ключ меньше текущего"
3  A[i] = key
4  while i > 1 и A[PARENT(i)] < A[i]
5      Обменять A[i] и A[PARENT(i)]
6      i = PARENT(i)
```

На рис. 6.5 показан пример выполнения операции HEAP-INCREASE-KEY. Время работы процедуры HEAP-INCREASE-KEY над n-элементной пирамидой равно $O(\lg n)$, поскольку путь от обновляемого в строке 3 узла до корня имеет длину $O(\lg n)$.

Процедура MAX-HEAP-INSERT реализует операцию INSERT. В качестве параметра этой процедуре передается ключ нового элемента, вставляемого в невозрастающую пирамиду A. Сначала процедура добавляет в пирамиду новый лист и присваивает ему ключ со значением $-\infty$. Затем вызывается процедура HEAP-INCREASE-KEY, которая присваивает корректное значение ключу нового узла и помещает его в надлежащее место, чтобы не нарушалось свойство невозрастающей пирамиды.

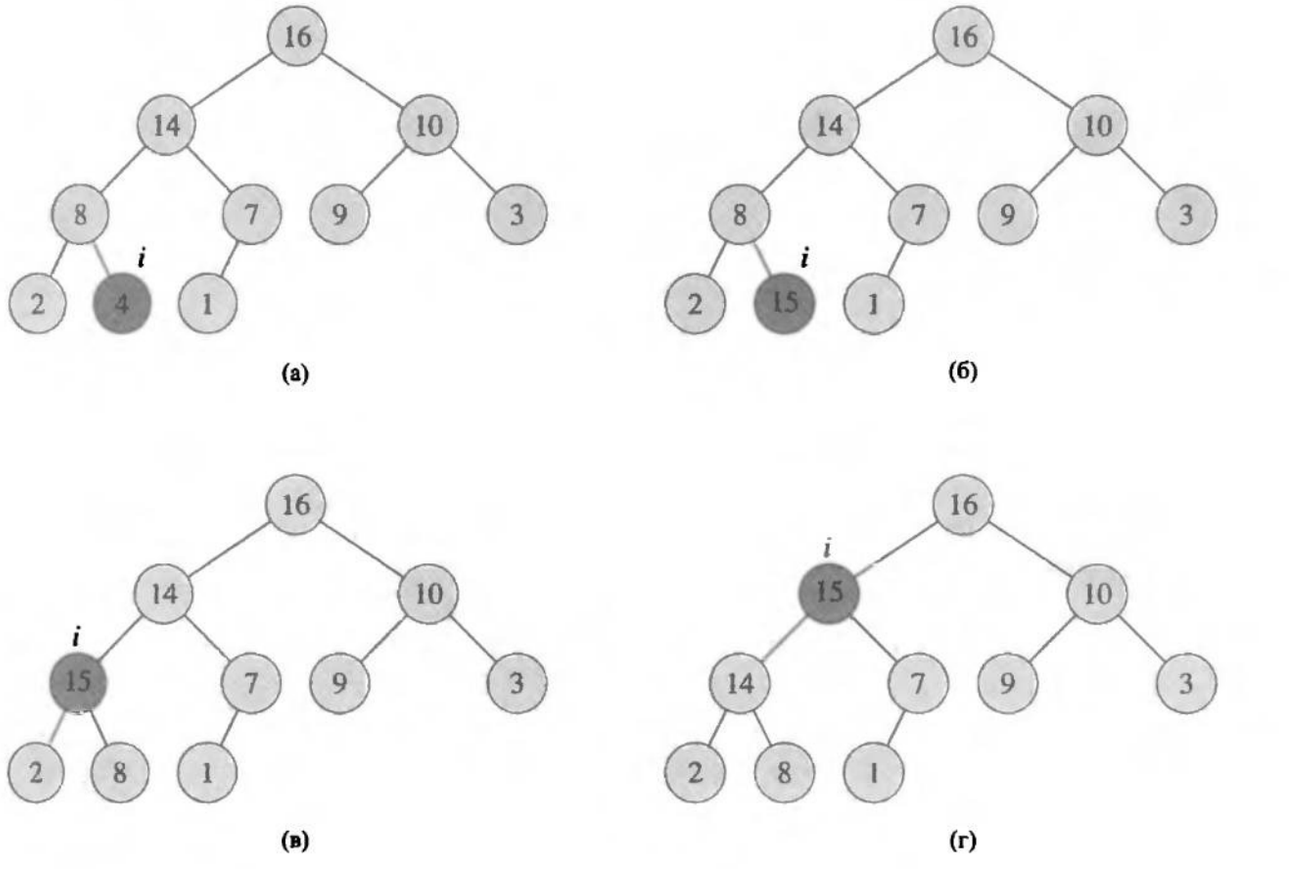

Рис. 6.5. Операция HEAP-INCREASE-KEY. **(а)** Невозрастающая пирамида с рис. 6.4, (а), с заштрихованным узлом с индексом i. **(б)** Ключ этого узла увеличивается до 15. **(в)** После одной итерации цикла **while** в строках 4–6 узел и его родитель обмениваются ключами, и индекс i перемещается в родительский узел. **(г)** Невозрастающая пирамида после еще одной итерации цикла **while**. В этот момент $A[\text{PARENT}(i)] \geq A[i]$. Теперь свойство невозрастающей пирамиды выполняется, и процедура завершает работу.

MAX-HEAP-INSERT(A, key)

1 $A.heap\text{-}size = A.heap\text{-}size + 1$
2 $A[A.heap\text{-}size] = -\infty$
3 HEAP-INCREASE-KEY$(A, A.heap\text{-}size, key)$

Время работы процедуры MAX-HEAP-INSERT с n-элементной пирамидой составляет $O(\lg n)$.

Подводя итог, заметим, что время выполнения всех операций по обслуживанию очереди с приоритетами в пирамиде равно $O(\lg n)$.

Упражнения

6.5.1

Проиллюстрируйте работу процедуры HEAP-EXTRACT-MAX над пирамидой $A = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$.

6.5.2

Проиллюстрируйте операцию MAX-HEAP-INSERT$(A, 10)$ над пирамидой $A = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$.

6.5.3

Напишите псевдокоды процедур HEAP-MINIMUM, HEAP-EXTRACT-MIN, HEAP-DECREASE-KEY и MIN-HEAP-INSERT, реализующих неубывающую очередь с приоритетами на базе неубывающей пирамиды.

6.5.4

Зачем нужна такая мера предосторожности, как присвоение ключу добавляемого в строке 2 процедуры MAX-HEAP-INSERT в пирамиду узла значения $-\infty$, если уже на следующем шаге значение этого ключа увеличивается до требуемой величины?

6.5.5

Докажите корректность процедуры HEAP-INCREASE-KEY с помощью следующего инварианта цикла.

> В начале каждой итерации цикла **while** в строках 4–6 $A[\text{PARENT}(i)] \geq A[\text{LEFT}(i)]$ и $A[\text{PARENT}(i)] \geq A[\text{RIGHT}(i)]$, если эти узлы существуют, а подмассив $A[1\,..\,A.heap\text{-}size]$ удовлетворяет свойству невозрастающей пирамиды, за исключением, возможно, одного нарушения: $A[i]$ может быть больше, чем $A[\text{PARENT}(i)]$.

Можно считать, что в момент вызова процедуры HEAP-INCREASE-KEY подмассив $A[1\,..\,A.heap\text{-}size]$ удовлетворяет свойству невозрастающей пирамиды.

6.5.6

Каждая операция обмена в строке 5 процедуры HEAP-INCREASE-KEY обычно требует трех присваиваний. Покажите, как воспользоваться идеей внутреннего цикла процедуры INSERTION-SORT, чтобы вместо трех присваиваний обойтись только одним.

6.5.7

Покажите, как с помощью очереди с приоритетами реализовать очередь "первым вошел — первым вышел". Продемонстрируйте, как с помощью очереди с приоритетами реализовать стек. (Очереди и стеки определены в разделе 10.1.)

6.5.8

Процедура HEAP-DELETE(A, i) удаляет из пирамиды A узел i. Разработайте реализацию этой процедуры, которой требуется время $O(\lg n)$ для удаления узла из n-элементной невозрастающей пирамиды.

6.5.9

Разработайте алгоритм, объединяющий k отсортированных списков в один список за время $O(n \lg k)$, где n — общее количество элементов во всех входных списках. (*Указание:* для слияния списков воспользуйтесь неубывающей пирамидой.)

Задачи

6.1. Построение пирамиды вставками

Пирамиду можно построить с помощью многократного вызова процедуры MAX-HEAP-INSERT для вставки элементов в пирамиду. Рассмотрим следующий вариант процедуры BUILD-MAX-HEAP.

```
BUILD-MAX-HEAP'(A)
1  A.heap-size = 1
2  for i = 2 to A.length
3      MAX-HEAP-INSERT(A, A[i])
```

а. Всегда ли процедуры BUILD-MAX-HEAP и BUILD-MAX-HEAP′ для одного и того же входного массива создают одну и ту же пирамиду? Докажите, что это так, или приведите контрпример.

б. Покажите, что в наихудшем случае для создания n-элементной пирамиды процедуре BUILD-MAX-HEAP′ потребуется время $\Theta(n \lg n)$.

6.2. Анализ d-арных пирамид

d-арные пирамиды подобны бинарным, но отличаются тем, что все внутренние узлы (с одним возможным исключением) имеют вместо двух d дочерних узлов.

а. Как бы вы представили d-арную пирамиду в виде массива?

б. Как выражается высота d-арной n-элементной пирамиды через n и d?

в. Разработайте эффективную реализацию процедуры EXTRACT-MAX, предназначенную для работы с d-арной невозрастающей пирамидой. Проанализируйте время работы этой процедуры и выразите его через d и n.

г. Разработайте эффективную реализацию процедуры INSERT, предназначенную для работы с d-арной невозрастающей пирамидой. Проанализируйте время работы этой процедуры и выразите его через d и n.

д. Разработайте эффективную реализацию процедуры INCREASE-KEY(A, i, k), которая при $k < A[i]$ сообщает об ошибке, а в противном случае выполняет присваивание $A[i] = k$ и соответствующим образом обновляет структуру d-арной невозрастающей пирамиды. Проанализируйте время работы этой процедуры и выразите его через d и n.

6.3. Таблицы Юнга

Таблица Юнга (Young tableau) $m \times n$ представляет собой матрицу размером $m \times n$, элементы которой в каждой строке отсортированы слева направо, а в каж-

дом столбце — сверху вниз. Некоторые элементы таблицы Юнга могут быть равны ∞, что трактуется как отсутствие элемента. Таким образом, таблицу Юнга можно использовать для хранения $r \leq mn$ конечных чисел.

а. Начертите таблицу Юнга 4×4, в которой содержатся элементы $\{9, 16, 3, 2, 4, 8, 5, 14, 12\}$.

б. Докажите, что таблица Юнга Y размером $m \times n$ пуста, если $Y[1,1] = \infty$. Докажите, что таблица Y полностью заполнена (т.е. содержит mn элементов), если $Y[m,n] < \infty$.

в. Разработайте алгоритм, реализующий процедуру EXTRACT-MIN для непустой таблицы Юнга $m \times n$ за время $O(m+n)$. В алгоритме следует использовать рекурсивную подпрограмму, которая решает задачу размером $m \times n$ путем рекурсивного сведéния к задачам $(m-1) \times n$ или $m \times (n-1)$. (*Указание:* вспомните о процедуре MAX-HEAPIFY.) Обозначим максимальное время обработки произвольной таблицы Юнга $m \times n$ с помощью процедуры EXTRACT-MIN как $T(p)$, где $p = m + n$. Запишите и решите рекуррентное соотношение, которое дает для $T(p)$ границу $O(m+n)$.

г. Покажите, как вставить новый элемент в незаполненную таблицу Юнга размером $m \times n$ за время $O(m+n)$.

д. Покажите, как с помощью таблицы Юнга $n \times n$ выполнить сортировку n^2 чисел за время $O(n^3)$, не используя при этом никаких других подпрограмм сортировки.

е. Разработайте алгоритм, позволяющий за время $O(m+n)$ определить, содержится ли в таблице Юнга размером $m \times n$ заданное число.

Заключительные замечания

Алгоритм пирамидальной сортировки был разработан Вильямсом (Williams) [355], который также описал, каким образом с помощью пирамиды можно реализовать очередь с приоритетами. Процедура BUILD-MAX-HEAP предложена Флойдом (Floyd) [105].

В главах 16, 23 и 24 неубывающие пирамиды будут использованы для реализации неубывающих очередей с приоритетами. В главе 19 будет представлена реализация с улучшенными временны́ми границами, а в главе 20 — усовершенствованная реализация для случая, когда ключи выбираются из ограниченного множества неотрицательных целых чисел.

Для случая, когда данные представляют собой b-битовые целые числа, а память компьютера состоит из адресуемых b-битовых слов, Фредман (Fredman) и Уиллард (Willard) [114] показали, как реализовать процедуру MINIMUM со временем работы $O(1)$ и процедуры INSERT и EXTRACT-MIN со временем работы

$O(\sqrt{\lg n})$. Торуп (Thorup) [335] улучшил границу $O(\sqrt{\lg n})$ до $O(\lg\lg n)$. При этом используемая память не ограничена величиной n, однако такого линейного ограничения используемой памяти можно достичь с помощью рандомизированного хеширования.

Важный частный случай очередей с приоритетами имеет место, когда последовательность операций EXTRACT-MIN является ***монотонной***, т.е. возвращаемые последовательными операциями EXTRACT-MIN значения образуют монотонно неубывающую последовательность. Такая ситуация встречается во многих важных приложениях, например в алгоритме поиска кратчайшего пути Дейкстры (Dijkstra), который рассматривается в главе 24, или при моделировании дискретных событий. Для алгоритма Дейкстры особенно важна эффективность реализации операции DECREASE-KEY. Для частного случая монотонности для целочисленных данных из диапазона $1, 2, \ldots, C$ Ахуйя (Ahuja), Мельхорн (Mehlhorn), Орлин (Orlin) и Таржан (Tarjan) [8] описали, как с помощью структуры данных под названием "позиционная пирамида" (radix heap) реализовать операции EXTRACT-MIN и INSERT с амортизированным временем работы $O(\lg C)$ (более подробные сведения на эту тему можно найти в главе 17) и операцию DECREASE-KEY со временем работы $O(1)$. Граница $O(\lg C)$ может быть улучшена до $O(\sqrt{\lg C})$ путем совместного использования пирамид Фибоначчи (см. главу 19) и позиционных пирамид. Дальнейшее улучшение этой границы до $O(\lg^{1/3+\epsilon} C)$ было осуществлено Черкасски (Cherkassky), Гольдбергом (Goldberg) и Сильверстайном (Silverstein) [64], которые объединили многоуровневую группирующую структуру (multi-level bucketing structure) Денардо (Denardo) и Фокса (Fox) [84] с уже упоминавшейся пирамидой Торупа. Раману (Raman) [289] удалось еще больше улучшить эти результаты и получить границу, которая равна $O(\min(\lg^{1/4+\epsilon} C, \lg^{1/3+\epsilon} n))$ для произвольной фиксированной величины $\epsilon > 0$.