3.4. СОРТДЕРЕВОМ — УПОРЯДОЧЕНИЕ С ПОМОЩЬЮ $O(n \log n)$ СРАВНЕНИЙ

Так как любой сортирующий алгоритм, упорядочивающий с помощью сравнений, затрачивает по необходимости $n \log n$ сравнений для упорядочения хотя бы одной последовательности длины n, естественно спросить, существуют ли сортирующие алгоритмы, затрачивающие не более $O(n \log n)$ сравнений для упорядочения любой последовательности длины n. Один такой алгоритм мы уже видели — это сортировка слиянием из разд. 2.7. Другой алгоритм — Сортдеревом. Помимо того, что это полезный алгоритм упорядочения, в нем используется интересная структура данных, которая находит и другие приложения.

Сортдеревом лучше всего понять в терминах двоичного дерева вроде изображенного на рис. 3.5, у которого каждый лист имеет глубину d или d—1. Узлы дерева помечаются элементами последовательности, которую хотят упорядочить. Затем Сортдеревом меняет размещение этих элементов на дереве до тех пор, пока элемент, соответствующий произвольному узлу, станет не меньше элементов, соответствующих его сыновьям. Такое помеченное дерево мы будем называть *сортирующим*.

Пример 3.3. На рис. 3.5 изображено сортирующее дерево. Заметим, что последовательность элементов, лежащих на пути из любого листа в корень, линейно упорядочена и наибольший элемент в поддереве всегда соответствует его корню. □

На следующем шаге алгоритма Сортдеревом из сортирующего дерева удаляется наибольший элемент — он соответствует корню дерева. Метка некоторого листа переносится в корень, а сам лист удаляется. Затем полученное дерево переделывается в сортирующее, и процесс повторяется. Последовательность элементов, удаленных из сортирующего дерева, упорядочена по невозрастанию.

Удобной структурой данных для сортирующего дерева служит такой массив A, что $A[1]$ — элемент в корне, а $A[2i]$ и $A[2i+1]$ —

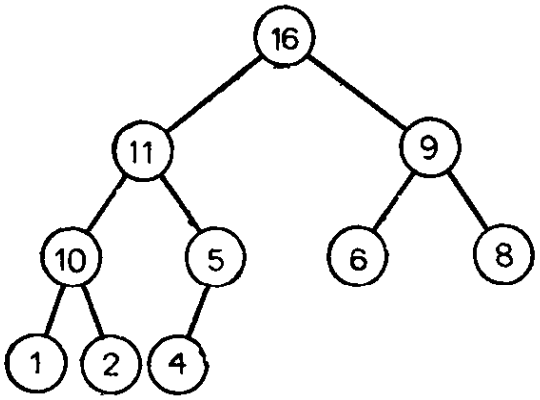


Рис. 3.5. Сортирующее дерево.

элементы в левом и правом сыновьях (если они существуют) того узла, в котором хранится $A[i]$.

Например, сортирующее дерево на рис. 3.5 можно представить массивом

16	11	9	10	5	6	8	1	2	4

Заметим, что узлы наименьшей глубины стоят в этом массиве первыми, а узлы равной глубины стоят в порядке слева направо.

Не каждое сортирующее дерево можно представить таким способом. На языке представления деревьев можно сказать, что для образования такого массива требуется, чтобы листья самого низкого уровня стояли как можно левее (как, например, на рис. 3.5).

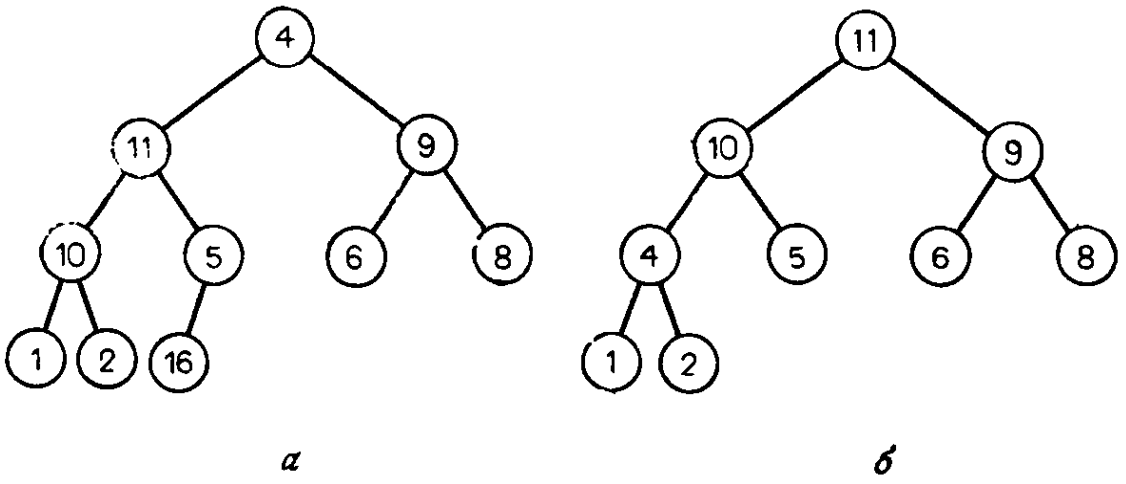
Если сортирующее дерево представляется описанным массивом, то некоторые операции из алгоритма Сортдеревом легко выполнить. Например, согласно алгоритму, нужно удалить элемент из корня, где-то запомнить его, переделать оставшееся дерево в сортирующее и удалить непомеченный лист. Можно удалить наибольший элемент из сортирующего дерева и запомнить его, поменяв местами $A[1]$ и $A[n]$, и затем не считать более ячейку n нашего массива частью сортирующего дерева. Ячейка n рассматривается как лист, удаленный из этого дерева. Для того чтобы переделать дерево, хранящееся в ячейках $1,2, \ldots, n-1$, в сортирующее, надо взять новый элемент $A[1]$ и провести его вдоль подходящего пути в дереве. Затем можно повторить процесс, меняя местами $A[1]$ и $A[n-1]$ и считая, что дерево занимает ячейки $1, 2, \ldots, n-2$ и т. д.

Пример 3.4. Рассмотрим на примере сортирующего дерева рис. 3.5, что происходит, когда мы поменяем местами первый и последний элементы массива, представляющего это дерево. Новый массив

4	11	9	10	5	6	8	1	2	16

соответствует помеченному дереву на рис. 3.6,*а*. Элемент 16 исключается из дальнейшего рассмотрения. Чтобы превратить полученное дерево в сортирующее, надо поменять местами элемент 4 с бóльшим из его сыновей, т. е. с элементом 11.

В своем новом положении элемент 4 обладает сыновьями 10 и 5. Так как они больше 4, то 4 переставляется с 10, бóльшим сыном. После этого сыновьями элемента 4 в новом положении становятся 1 и 2. Поскольку 4 превосходит их обоих, дальнейшие перестановки не нужны.



а *б*

Рис. 3.6. *а* — результат перестановки элементов 4 и 16 в сортирующем дереве рис. 3.5; *б* — результат перестройки сортирующего дерева и удаления элемента 16.

Полученное в результате сортирующее дерево показано на рис. 3.6,*б*. Заметим, что хотя элемент 16 был удален из сортирующего дерева, он все же присутствует в конце массива A. □

Теперь перейдем к формальному описанию алгоритма Сортдеревом.

Пусть $a_1, a_2, \ldots, a_n$ — последовательность сортируемых элементов. Предположим, что вначале они помещаются в массив A именно в этом порядке, т. е. $A[i]=a_i$, $1 \leqslant i \leqslant n$. Первый шаг состоит в построении сортирующего дерева, т. е. элементы в A перераспределяются так, чтобы удовлетворялось *свойство сортирующего дерева*: $A[i] \geqslant A[2i]$ при $1 \leqslant i \leqslant n/2$ и $A[i] \geqslant A[2i+1]$ при $1 \leqslant i < n/2$. Это делают, строя, начиная с листьев, все большие и большие сортирующие деревья. Всякое поддерево, состоящее из листа, уже является сортирующим. Чтобы сделать поддерево высоты h сортирующим, надо переставить элемент в корне с наибольшим из элементов, соответствующих его сыновьям, если, конечно, он меньше какого-то из них. Такое действие может испортить сортирующее дерево высоты $h-1$, и тогда его надо снова перестроить в сортирующее. Приведем точное описание этого алгоритма.

Алгоритм 3.3. Построение сортирующего дерева

Вход. Массив элементов $A[i]$, $1 \leqslant i \leqslant n$.

Выход. Элементы массива A, организованные в виде сортирующего дерева, т. е. $A[i] \leqslant A[\lfloor n/2 \rfloor]$ для $1 < i \leqslant n$.

Метод. В основе алгоритма лежит рекурсивная процедура ПЕРЕСЫПКА. Ее параметры i и j задают область ячеек массива A, обладающую свойством сортирующего дерева; корень строящегося дерева помещается в i.

procedure ПЕРЕСЫПКА (i, j):
1. **if** i — не лист и какой-то его сын содержит элемент, превосходящий элемент в i **then**
   begin
2. пусть k — тот сын узла i, в котором хранится наибольший элемент;
3. переставить $A[i]$ и $A[k]$;
4. ПЕРЕСЫПКА (k, j)
   end

Параметр j используется, чтобы определить, является ли i листом и имеет он одного или двух сыновей. Если $i>j/2$, то i — лист, и процедуре ПЕРЕСЫПКА (i, j) ничего не нужно делать, поскольку $A[i]$ — уже сортирующее дерево.

Алгоритм, превращающий весь массив A в сортирующее дерево, выглядит просто:

procedure ПОСТРСОРТДЕРЕВА:
for $i \leftarrow n$[1]) **step** -1 **until** 1 **do** ПЕРЕСЫПКА (i, n) □

Покажем, что алгоритм 3.3 преобразует A в сортирующее дерево за линейное время.

Лемма 3.2. *Если узлы* $i+1, i+2, \ldots, n$ *являются корнями сортирующих деревьев, то после вызова процедуры* ПЕРЕСЫПКА (i, n) *все узлы* $i, i+1, \ldots, n$ *будут корнями сортирующих деревьев.*

Доказательство. Доказательство проводится возвратной индукцией по i.

Базис, т. е. случай $i=n$, тривиален, так как узел n должен быть листом и условие, проверяемое в строке 1, гарантирует, что ПЕРЕСЫПКА (n, n) ничего не делает.

Для шага индукции заметим, что если i — лист или у него нет сына с бóльшим элементом, то доказывать нечего (как и при обосновании базиса). Но если у узла i есть один сын (т. е. если $2i=n$) и $A[i]<A[2i]$, то строка 3 процедуры ПЕРЕСЫПКА переставляет $A[i]$ и $A[2i]$. В строке 4 вызывается ПЕРЕСЫПКА ($2i$, n); поэтому из предположения индукции вытекает, что дерево с корнем $2i$ будет переделано в сортирующее. Что касается узлов $i+1, i+2, \ldots, 2i-1$, то они и не переставали быть корнями сортирующих деревьев. Так как после этой новой перестановки в массиве A выполняется неравенство $A[i]>A[2i]$, то дерево с корнем i также оказывается сортирующим.

Аналогично, если узел i имеет двух сыновей (т. е. если $2i+1\leqslant n$) и наибольший из элементов в $A[2i]$ и в $A[2i+1]$ больше элемента

[1]) На практике мы бы начали с $\lfloor n/2 \rfloor$.

в $A[i]$, то, рассуждая, как и выше, можно показать, что после вызова процедуры ПЕРЕСЫПКА(i, n) все узлы i, $i+1$, . . ., n будут корнями сортирующих деревьев. □

Теорема 3.5. *Алгоритм* 3.3 *преобразует* A *в сортирующее дерево за линейное время.*

Доказательство. Применяя лемму 3.2, можно с помощью простой возвратной индукции по i показать, что узел i становится корнем какого-нибудь сортирующего дерева для всех i, $1 \leqslant i \leqslant n$.

Пусть $T(h)$ — время выполнения процедуры ПЕРЕСЫПКА на узле высоты h. Тогда $T(h) \leqslant T(h-1)+c$ для некоторой постоянной c. Отсюда вытекает, что $T(h)$ есть $O(h)$.

Алгоритм 3.3 вызывает процедуру ПЕРЕСЫПКА, если не считать рекурсивных вызовов, один раз для каждого узла. Поэтому время, затрачиваемое на ПОСТРСОРТДЕРЕВА, имеет тот же порядок, что и сумма высот всех узлов. Но узлов высоты i не больше, чем $\lceil n/2^{i+1} \rceil$. Следовательно, общее время, затрачиваемое процедурой ПОСТРСОРТДЕРЕВА, имеет порядок $\sum_{i=1}^{n} in/2^i$, т. е. $O(n)$. □

Теперь можно завершить детальное описание алгоритма СОРТДЕРЕВОМ. Коль скоро элементы массива A преобразованы в сортирующее дерево, некоторые элементы удаляются из корня по одному за раз. Это делается перестановкой $A[1]$ и $A[n]$ и последующим преобразованием $A[1]$, $A[2]$, . . ., $A[n-1]$ в сортирующее дерево. Затем переставляются $A[1]$ и $A[n-1]$, а $A[1]$, $A[2]$, ..., $A[n-2]$ преобразуется в сортирующее дерево. Процесс продолжается до тех пор, пока в сортирующем дереве не останется один элемент. Тогда $A[1]$, $A[2]$, . . ., $A[n]$ — упорядоченная последовательность.

Алгоритм 3.4. Сортдеревом

Вход. Массив элементов $A[i]$, $1 \leqslant i \leqslant n$.

Выход. Элементы массива A, расположенные в порядке неубывания.

Метод. Применяется процедура ПОСТРСОРТДЕРЕВА, т. е. алгоритм 3.3. Наш алгоритм выглядит так:

```
begin
    ПОСТРСОРТДЕРЕВА;
    for i ← n step −1 until 2 do
        begin
            переставить A[1] и A[i];
            ПЕРЕСЫПКА(1, i−1)
        end
end
```

□

Теорема 3.6. *Алгоритм* 3.4 *упорядочивает последовательность из* n *элементов за время* $O(n \log n)$.

Доказательство. Корректность алгоритма доказывается индукцией по числу выполнений основного цикла, которое обозначим через m. Предположение индукции состоит в том, что после m итераций список $A[n-m+1], \ldots, A[n]$ содержит m наибольших элементов, расположенных в порядке неубывания, а список $A[1], \ldots, A[n-m]$ образует сортирующее дерево. Детали доказательства оставляем в качестве упражнения. Время выполнения процедуры ПЕРЕСЫПКА$(1, i)$ есть $O(\log i)$. Следовательно, алгоритм 3.4 имеет сложность $O(n \log n)$. □

Следствие. *Алгоритм Сортдеревом имеет временную сложность* $O_C(n \log n)$.

3.5. БЫСТРСОРТ — УПОРЯДОЧЕНИЕ ЗА СРЕДНЕЕ ВРЕМЯ $O(n \log n)$

До сих пор мы рассматривали поведение сортирующих алгоритмов только в худшем случае. Для многих приложений более реалистичной мерой временной сложности является усредненное время работы. В случае сортировки с помощью деревьев решений мы видим, что никакой сортирующий алгоритм не может иметь среднюю временную сложность, существенно меньшую $n \log n$. Однако известны алгоритмы сортировки, которые работают в худшем случае cn^2 времени, где c — некоторая постоянная, но среднее время работы которых относит их к лучшим алгоритмам сортировки. Примером такого алгоритма служит алгоритм Быстрсорт, рассматриваемый в этом разделе.

Чтобы можно было рассуждать о среднем времени работы алгоритма, мы должны договориться о вероятностном распределении входов. Для сортировки естественно допустить, что мы и сделаем, что любая перестановка упорядочиваемой последовательности равновероятна в качестве входа. При таком допущении уже можно оценить снизу среднее число сравнений, необходимых для упорядочения последовательности из n элементов.

Общий метод состоит в том, чтобы поставить в соответствие каждому листу v дерева решений вероятность быть достигнутым при данном входе. Зная распределение вероятностей на входах, можно найти вероятности, поставленные в соответствие листьям. Таким образом, можно определить среднее число сравнений, производимых данным алгоритмом сортировки, если вычислить сумму $\sum_i p_i d_i$, взятую по всем листьям дерева решений данного алгоритма, в которой p_i — вероятность достижения i-го листа, а d_i — его глубина. Это число называется *средней глубиной* дерева решений. Итак, мы пришли к следующему обобщению теоремы 3.4.