Теперь должно быть ясно, как доказать по индукции, что процедура ПОСТРДЕРЕВА (i, j) правильно строит оптимальное дерево для $\{a_{i+1}, a_{i+2}, \ldots, a_j\}$. □

В алгоритме 4.2 можно ограничить поиск m в строке 8 рис. 4.9 областью между положениями корней деревьев $T_{i,j-1}$ и $T_{i+1,j}$, при этом гарантируется нахождение минимума. Тогда алгоритм 4.2 сможет находить оптимальное дерево за время $O(n^2)$.

4.6. ПРОСТОЙ АЛГОРИТМ ДЛЯ НАХОЖДЕНИЯ ОБЪЕДИНЕНИЯ НЕПЕРЕСЕКАЮЩИХСЯ МНОЖЕСТВ

Рассмотрим обработку узлов в алгоритме для нахождения остовного дерева (пример 4.1). Возникающая здесь задача преобразования множеств обладает следующими тремя особенностями.

1. Всякий раз сливаются только непересекающиеся множества.
2. Элементы множеств можно считать целыми числами от 1 до n.
3. Операциями над множествами являются ОБЪЕДИНИТЬ и НАЙТИ

В этом и следующем разделах мы изучим структуры данных для задач такого типа. Пусть даны n элементов, которые мы будем считать целыми числами 1, 2, . . ., n. Предположим, что вначале каждый элемент образует одноэлементное множество. Пусть надо выполнить последовательность операций ОБЪЕДИНИТЬ и НАЙТИ. Напомним, что операция ОБЪЕДИНИТЬ имеет вид ОБЪЕДИНИТЬ(A, B, C), указывающий, что два непересекающихся множества с именами A и B надо заменить их объединением и назвать это объединение C. В приложениях часто неважно, что выбирается в качестве имени множества, так что мы будем предполагать, что множества можно именовать целыми числами от 1 до n. Кроме того, мы будем предполагать, что никакие два множества ни в один момент не названы одинаково.

Эту задачу позволяют решить несколько интересных структур данных. Здесь мы познакомимся со структурой данных, благодаря которой можно выполнить за время $O(n \log n)$ последовательность, содержащую до n—1 операций ОБЪЕДИНИТЬ и до $O(n \log n)$ операций НАЙТИ. В следующем разделе опишем структуру данных, позволяющую обрабатывать последовательность из $O(n)$ операций ОБЪЕДИНИТЬ и НАЙТИ в худшем случае за время, почти линейное по n. Эти структуры данных могут обрабатывать последовательности операций ВСТАВИТЬ, УДАЛИТЬ и ПРИНАДЛЕЖАТЬ с той же вычислительной сложностью.

Заметим, что в алгоритмах поиска, изложенных в разд. 4.2—4.5, предполагалось, что элементы выбираются из универсального

множества, много большего, чем множество выполняемых операций. В этом разделе универсальное множество будет приблизительно того же размера, что и длина последовательности выполняемых операций.

По-видимому, простейшей структурой данных для задачи типа ОБЪЕДИНИТЬ — НАЙТИ служит массив, представляющий набор множеств в данный момент. Пусть R — массив размера n, а $R[i]$ — имя множества содержащего элемент i. Так как вид имен множеств не существен, можно вначале взять $R[i]=i$, $1 \leqslant i \leqslant n$, и выразить тем самым факт, что перед началом работы набором множеств является $\{\{1\}, \{2\}, \ldots, \{n\}\}$ и множество $\{i\}$ имеет имя i.

Операция НАЙТИ(i) выполняется путем печати значения $R[i]$ в данный момент. Поэтому сложность выполнения операции НАЙТИ постоянна, а это лучшее, на что можно надеяться.

Чтобы выполнить операцию ОБЪЕДИНИТЬ(A, B, C), надо последовательно просмотреть массив R и заменить каждую его компоненту, равную A или B, на C. Поэтому сложность выполнения такой операции есть $O(n)$. Последовательность из n операций ОБЪЕДИНИТЬ могла бы потребовать $O(n^2)$ времени, что нежелательно.

Этот безыскусный алгоритм можно улучшить несколькими способами. Для одного улучшения можно воспользоваться преимуществом связанных списков. Для другого — понять, что всегда эффективнее влить меньшее множество в большее. Чтобы сделать это, надо отличать "внутренние имена", используемые для идентификации множеств в массиве R, от "внешних имен", упоминаемых в операциях ОБЪЕДИНИТЬ. И те, и другие предполагаются числами от 1 до n, но не обязательно одинаковыми.

Рассмотрим следующую структуру данных для этой задачи. Как и ранее, возьмем такой массив R, что $R[i]$ содержит "внутреннее" имя множества, которому принадлежит элемент i. Но теперь для каждого множества A мы построим связанный список СПИСОК[A], содержащий его элементы. Для реализации этого связанного списка применяются два массива СПИСОК и СЛЕДУЮЩИЙ. СПИСОК[A] представляет собой целое число j, указывающее, что j — первый элемент в множестве с внутренним именем A. СЛЕДУЮЩИЙ[j] дает следующий элемент в A, СЛЕДУЮЩИЙ[СЛЕДУЮЩИЙ[j]] — следующий за ним элемент, и т. д.

Кроме того, возьмем еще массив, называемый РАЗМЕР, такой, что РАЗМЕР[A] — число элементов в множестве A. Множества будут переименовываться по внутренним именам, а два массива ВНУТР_ИМЯ и ВНЕШ_ИМЯ будут устанавливать соответствие между внутренними и внешними именами. Иными словами, ВНЕШ_ИМЯ[A] — это настоящее имя (диктуемое операциями ОБЪЕДИНИТЬ) множества с внутренним именем A. ВНУТР_ИМЯ[j] — это внутреннее имя множества с внешним именем j. Внутренние имена — это имена, используемые в массиве R.

	R	СЛЕДУЮЩИЙ
1	2	3
2	3	4
3	2	5
4	3	8
5	2	7
6	1	0
7	2	0
8	3	0

	СПИСОК	РАЗМЕР	ВНЕШ_ИМЯ
1	6	1	3
2	1	4	1
3	2	3	2

Множества (с внешними именами)

1 = {1, 3, 5, 7}

2 = {2, 4, 8}

3 = {6}

	ВНУТР_ИМЯ
1	2
2	3
3	1

Рис. 4.13. Структуры данных для алгоритма ОБЪЕДИНИТЬ — НАЙТИ.

```
    procedure ОБЪЕДИНИТЬ(I, J, K):
    begin
 1.   A ← ВНУТР_ИМЯ[I];
 2.   B ← ВНУТР_ИМЯ[J];
 3.   wlg положить РАЗМЕР[A] ⩽ РАЗМЕР[B]
 4.     otherwise поменять ролями A и B in
        begin
 5.       ЭЛЕМЕНТ ← СПИСОК[A];
 6.       while ЭЛЕМЕНТ ≠ 0 do
            begin
 7.           R[ЭЛЕМЕНТ] ← B;
 8.           ПОСЛЕДНИЙ ← ЭЛЕМЕНТ;
 9.           ЭЛЕМЕНТ ← СЛЕДУЮЩИЙ[ЭЛЕМЕНТ]
            end;
10.       СЛЕДУЮЩИЙ[ПОСЛЕДНИЙ] ← СПИСОК[B];
11.       СПИСОК[B] ← СПИСОК[A];
12.       РАЗМЕР[B] ← РАЗМЕР[A] + РАЗМЕР[B];
13.       ВНУТР_ИМЯ [K] ← B;
14.       ВНЕШ_ИМЯ[B] ← K
        end
    end
```

Рис. 4.14. Реализация операции ОБЪЕДИНИТЬ.

Пример 4.6. Пусть $n=8$ и у нас есть набор из трех множеств $\{1, 3, 5, 7\}$, $\{2, 4, 8\}$ и $\{6\}$ с внешними именами 1, 2 и 3 соответственно. Структуры данных для этих трех множеств показаны на рис. 4.13, где 2,3 и 1 — внутренние имена для 1,2 и 3 соответственно. □

Операция НАЙТИ(i) выполняется, как и раньше, обращением к $R[i]$ для установления внутреннего имени множества, содержащего элемент i в данный момент. Затем ВНЕШ_ИМЯ[$R[i]$] дает настоящее имя множества, которому принадлежит i.

Операцию объединения вида ОБЪЕДИНИТЬ(I, J, K) выполняем следующим образом. (Номера строк относятся к рис. 4.14.)

1. Определяем внутренние имена для множеств I и J (строки 1, 2).
2. Сравниваем относительные размеры множеств I и J, справляясь в массиве РАЗМЕР (строки 3, 4).
3. Проходим список элементов меньшего множества и изменяем соответствующие компоненты в массиве R на внутреннее имя большего множества (строки 5—9).
4. Вливаем меньшее множество в большее, добавляя список элементов меньшего множества к началу списка для большего множества (строки 10—12).
5. Присваиваем полученному множеству внешнее имя K (строки 13, 14).

Вливая меньшее множество в большее, мы делаем время выполнения операции ОБЪЕДИНИТЬ пропорциональным мощности меньшего множества. Все детали приведены в процедуре на рис.4.14.

Пример 4.7. После выполнения операции ОБЪЕДИНИТЬ (1, 2, 4) структура данных рис. 4.13 превратится в структуру данных, изображенную на рис. 4.15. □

Теорема 4.3. *С помощью алгоритма рис.* 4.14 *можно выполнить* $n-1$ *(максимально возможное число) операций* ОБЪЕДИНИТЬ *за* $O(n \log n)$ *шагов.*

Доказательство. За каждое выполнение операции ОБЪЕДИНИТЬ будем налагать на перемещаемые элементы одинаковые штрафы, в сумме равные сложности этого выполнения. Так как сложность пропорциональна числу перемещаемых элементов, то величина штрафа, налагаемого на один элемент, будет всегда одна и та же. Основное здесь — это заметить, что всякий раз, когда элемент перемещается из списка, он оказывается в списке, по крайней мере в два раза длиннее прежнего. Поэтому никакой элемент нельзя переместить более чем $\log n$ раз и, значит, суммарный штраф, налагаемый на один элемент, составляет $O(\log n)$.

	R	СЛЕДУЮЩИЙ
1	2	3
2	2	4
3	2	5
4	2	8
5	2	7
6	1	0
7	2	0
8	2	1

	СПИСОК	РАЗМЕР	ВНЕШ_ИМЯ
1	6	1	3
2	2	7	4
3	—	—	—
4	—	—	—

Множества (с внешними именами)

$3 = \{6\}$

$4 = \{1, 2, 3, 4, 5, 7, 8\}$

	ВНУТР_ИМЯ
1	—
2	—
3	1
4	2

Рис. 4.15. Структура данных после выполнения операции ОБЪЕДИНИТЬ.

Общая сложность получается суммированием штрафов, наложенных на отдельные элементы. Таким образом, общая сложность есть $O(n \log n)$. □

Из теоремы 4.3 вытекает, что если выполняется m операций НАЙТИ и до $n-1$ операций ОБЪЕДИНИТЬ, то тратится время $O(\text{MAX}(m, n \log n))$. Если m имеет порядок $n \log n$ или больше, то этот алгоритм действительно оптимален с точностью до постоянного множителя. Однако во многих ситуациях m есть $O(n)$, а в таком случае, как мы увидим в следующем разделе, можно добиться лучшего времени, нежели $O(\text{MAX}(m, n \log n))$.

4.7. ДРЕВОВИДНЫЕ СТРУКТУРЫ ДЛЯ ЗАДАЧИ ОБЪЕДИНИТЬ — НАЙТИ

В предыдущем разделе мы познакомились со структурой данных для задачи ОБЪЕДИНИТЬ — НАЙТИ, позволяющей выполнить $n-1$ операций ОБЪЕДИНИТЬ и $O(n \log n)$ операций НАЙТИ за время $O(n \log n)$. В данном разделе будет описана структура данных, которая состоит из деревьев, образующих лес, и предназначена для представления набора множеств. Эта структура данных позволит выполнить $O(n)$ операций ОБЪЕДИНИТЬ и НАЙТИ за почти линейное время.