```
begin
    LIST ← (s1, s2);
    COLLECTION ← ∅;
    for для каждого состояния s ∈ S1 ∪ S2 do добавить {s}
                                в COLLECTION;
    comment мы инициализировали исходные множества
        для каждого состояния из S1 ∪ S2;
    while существует пара (s, s'), входящая в LIST do
        begin
            удалить (s, s') из LIST;
            пусть A и A' обозначают FIND(s) и FIND(s') соответственно;
                if A ≠ A' then
                    begin
                        UNION(A, A', A);
                        for все a ∈ I  do
                            добавить (δ(s, a), δ(s', a)) в LIST
                    end
        end
end
```

Рис. 4.25. Алгоритм для распознавания множеств эквивалентных состояний в предположении, что состояния s_1 и s_2 эквивалентны

4.9. Схемы сбалансированных деревьев

При решении ряда важных классов задач, аналогичных задаче UNION–FIND, приходится вернуться к способам, при которых последовательность из n операций выполняется за время $O(n\log n)$ в худшем случае. Один из таких классов состоит из задач, в которых последовательность, состоящая из операций MEMBER, INSERT и DELETE, выполняется над элементами, мощность базового множества которых намного больше количества реально используемых элементов. В этом случае нельзя выбирать элемент, непосредственно индексируя массив указателей. Необходимо применять хеширование или дерево бинарного поиска.

Если n элементов уже вставлены, то время выполнения одной операции выбора в методе хеширования является постоянным в среднем и $O(n)$ в худшем случае. В дереве бинарного поиска среднее время одной операции выбора имеет порядок $O(\log n)$, но, если множество имен не является статичным, временная сложность в худшем случае тоже может оказаться высокой. Если просто добавлять имена в дерево, не поддерживая его сбалансированность, то можно в результате получить дерево с n вершинами, глубина которого близка к n. Тогда в худшем случае временная сложность операции в дереве бинарного поиска будет составлять $O(n)$. Методом, изложенным в этом разделе, можно уменьшить временную сложность в худшем случае до $O(\log n)$ шагов на операцию.

Другой класс задач, требующих $O(n \log n)$ времени, состоит из задач выполнения последовательности. состоящей из n операций INSERT, DELETE и MIN, в онлайн-режиме. Еще один, третий, класс задач возникает, когда нужно представлять упорядоченные списки и уметь конкатенировать и расщеплять их.

В этом разделе описываются методы выполнения в онлайн-режиме последовательностей, содержащих важные подмножества семи основных операций на множествах, введенных в разделе 4.1. Структурой данных, лежащей в основе метода, является *сбалансированное дерево*, под которым мы понимаем дерево с высотой, приблизительно равной логарифму числа его вершин.

Вначале сбалансированное дерево строится легко. Однако в процессе выполнения последовательности операций INSERT и DELETE трудно предотвратить нарушение баланса. Например, если периодически не осуществлять балансировку, то при выполнении последовательности операций DELETE, которые удаляют вершины только из левой части дерева, получится дерево, смещенное вправо.

Существует много способов балансировки дерева. Некоторые из них оставляют структуру дерева довольно гибкой, так что число вершин в дереве высоты h может изменяться от 2^h до 2^{h+1} или 3^h. Такие методы позволяют, по меньшей мере, удвоить число вершин в поддереве, прежде чем что-то менять выше его корня.

Мы обсудим два метода такого вида, называемые методами 2-3-деревьев и AVL-деревьев. Алгоритмы, работающие с 2-3-деревьями, интуитивно понятнее, поэтому мы обсудим их в первую очередь, а похожие на них алгоритмы работы с AVL-деревьями вынесем в упражнения.

Определение. *2-3-деревом* называется дерево, в котором каждая вершина, не являющаяся листом, имеет двух или трех сыновей, а длины всех путей из корня в листья одинаковы. Заметим, что дерево, состоящее из единственной вершины, является 2-3-деревом. На рис. 4.26 приведены два 2-3-дерева с шестью листьями.

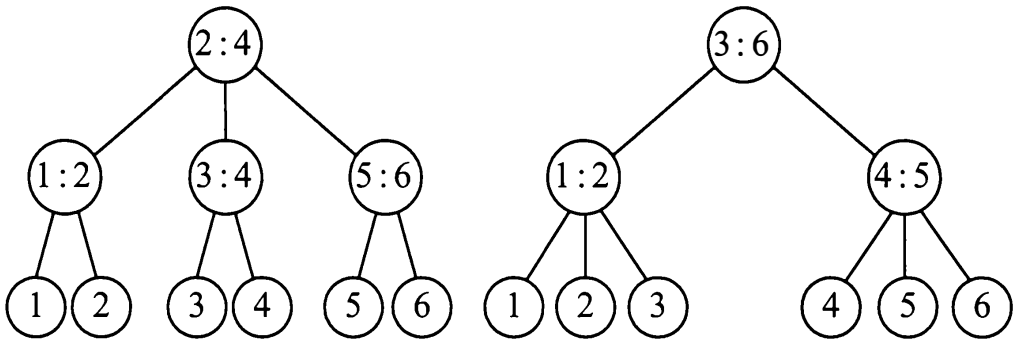

Рис. 4.26. 2-3-деревья

Следующая лемма устанавливает связь между количеством вершин и листьев в 2-3-дереве и его высотой.

Лемма 4.6. Пусть T — 2-3-дерево высоты h. Количество вершин дерева T лежит в диапазоне от $2^{h+1} - 1$ до $(3^{h+1} - 1)/2$, а число листьев — от 2^h и 3^h.

Доказательство. Простая индукция по h. □

Линейно упорядоченное множество S можно представить в виде 2-3-дерева, присвоив листьям дерева его элементы. Обозначим элемент, присвоенный листу l, через $E[l]$. Существуют два основных метода присваивания элементов листьям, применение которых зависит от рассматриваемой задачи.

Если базовое множество элементов гораздо больше, чем количество реально используемых элементов, и дерево будет использоваться как словарь, то, вероятно, лучше присваивать элементы в порядке возрастания слева направо. В каждой вершине v, не являющейся листом, нам потребуется еще два элемента: $L[v]$ — наибольший элемент множества S в поддереве, корнем которого служит самый левый сын вершины v, и $M[v]$ — наибольший элемент множества S в поддереве, корнем которого служит второй сын вершины v (см. рис. 4.26). Значения L и M, присвоенные вершинам, позволяют искать элемент, начиная с корня, способом, напоминающим бинарный поиск. Время обнаружения произвольного элемента пропорционально высоте дерева, поэтому операцию MEMBER на множестве из n элементов можно выполнить за время $O(\log n)$, если представить его в виде 2-3-дерева такого вида.

Во втором методе присваивания элементов листьям порядок выполнения этих операций ничем не ограничен. Этот метод особенно полезен для реализации операции UNION. Однако для выполнения операций типа DELETE необходим механизм для определения положения листа, представляющего данный элемент. Если элементами множества являются целые числа из фиксированного диапазона, например, от 1 до n, то лист, представляющий элемент i, можно найти с помощью i-й ячейки некоторого массива. Если же элементы рассматриваемых множеств принадлежат некоторому большому базовому множеству, то лист, представляющий элемент i, можно найти с помощью вспомогательного словаря.

Рассмотрим следующие наборы операций.

1) INSERT, DELETE, MEMBER.

2) INSERT, DELETE, MIN.

3) INSERT, DELETE, UNION, MIN.

4) INSERT, DELETE, FIND, CONCATENATE, SPLIT.

Структуру данных, обеспечивающую выполнение операций из множества 1, будем называть *словарем* (dictionary), из множества 2 — *очередью с приоритетами* (priority queue), из множества 3 — *сливаемой кучей* (mergeable heap), из множества 4 — *сцепляемой очередью* (concatenable queue).

Покажем, что 2-3-деревья могут служить для реализации словарей, очередей с приоритетами, сцепляемых очередей и сливаемых деревьев, применение которых обеспечивает выполнение n операций за время $O(n \log n)$. Описанные здесь методы достаточно мощные, чтобы выполнять последовательности, составленные из любого совместимого подмножества семи операций, перечисленных в начале главы. Единственная несовместимость состоит в том, что операция `UNION` предполагает неупорядоченное множество, а `SPLIT` и `CONCATENATE` предполагают наличие порядка.

4.10. Словари и очереди с приоритетами

В этом разделе мы изучим основные операции, необходимые для реализации словарей и очередей с приоритетами. На протяжении всего раздела будем предполагать, что элементы присвоены листьям 2-3-дерева в порядке слева направо и в каждой внутренней вершине v определены функции $L[v]$ и $M[v]$, введенные в предыдущем разделе.

Чтобы вставить новый элемент a в 2-3-дерево, необходимо найти позицию для нового листа l, который будет содержать элемент a. Для этого необходимо выполнить поиск элемента a в дереве. Если дерево содержит более одного элемента, то поиск элемента a закончится в вершине f, имеющей двух или трех сыновей, которые являются листьями.

Если вершина f имеет только два листа, l_1 и l_2, то l делаем сыном вершины f. Если $a < E[l_1]$,[6] то l делаем самым левым сыном вершины f и полагаем $L[f] = a$ и $M[f] = E[l_1]$; если $E[l_1] < a < E[l_2]$, то l делаем средним сыном вершины f и полагаем $M[f] = a$; если $E[l_2] < a$, то l делаем третьим сыном вершины f. Значения L и M на некоторых прямых предках вершины f, возможно, придется изменить.

Пример 4.9. Если в 2-3-дерево, показанное на рис. 4.27, *а*, вставляется элемент 2, то получается 2-3-дерево, изображенное на рис. 4.27, *б*. □

[6] $E[v]$ — элемент, хранящийся в листе v.