

## Лекции 7-8. Структуры данных для операций с геометрическими объектами

### 1. Операции над множествами геометрических объектов

Рассмотрим основные операции над множествами геометрических объектов и структуры данных, пригодные для представления этих множеств и операций.

Пусть  $S$  - множество каких-то элементов и  $a$  - какой-то объект. Будем рассматривать следующие операции:

FUNCTION  $Belong(a, S)$  - устанавливает принадлежность  $a$  множеству  $S$ ;

PROCEDURE  $Insert(a, S)$  - заменяет  $S$  на  $S \cup \{a\}$ ;

PROCEDURE  $Remove(a, S)$  - заменяет  $S$  на  $S \setminus \{a\}$ ;

FUNCTION  $Min(S)$  - возвращает наименьший элемент множества  $S$  относительно принятого правила сравнения элементов ( $\leq$ ).

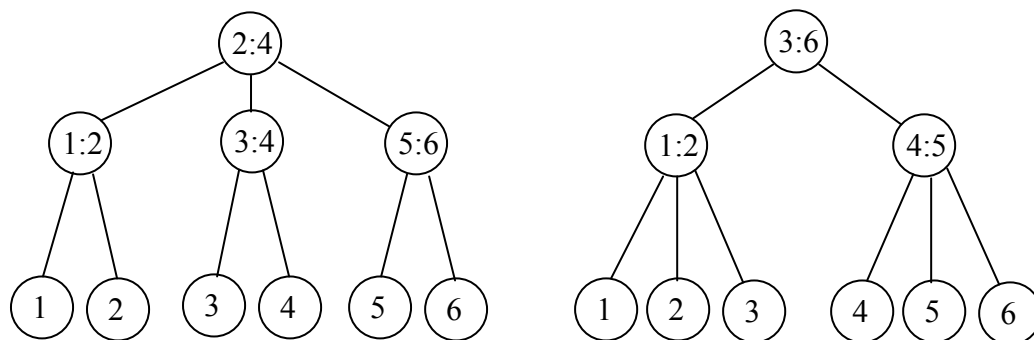
Многие задачи можно свести к выполнению последовательности этих операций на некотором множестве элементов. Структуры данных, обеспечивающие выполнение этих операций, имеют специальные названия. Структура, поддерживающая цепочки операций  $Belong$ ,  $Insert$  и  $Remove$ , называется *словарем*, а структура, поддерживающая операции  $Insert$ ,  $Remove$  и  $Min$  называется *очередью с приоритетами*.

В принципе в качестве структуры, реализующей указанные возможности можно выбрать самые простые, например, обычный массив. Ясно, однако, что при этом выполнение каждой операции потребует времени  $O(n)$  при количестве элементов в  $S$  равном  $n$ . Существуют структуры данных, позволяющие более эффективно выполнять требуемые операции. Например, двоичное сбалансированное дерево позволяет выполнить операции  $Belong$  и  $Min$  за время  $O(\log n)$ . Однако, при многократном выполнении операций  $Insert$  и  $Remove$  может сильно нарушиться сбалансированность двоичного дерева, что приведет к снижению эффективности этих операций вплоть до величины  $O(n)$ .

Рассмотрим структуру данных, обеспечивающую поддержку сбалансированности дерева, так называемое *2-3-дерево*.

### 2. Определение 2-3-дерева

*2-3-деревом* называется дерево, в котором каждый узел, не являющийся листом, имеет двух или трех сыновей, а длины всех путей из корня в листья одинаковы. Заметим, что дерево, состоящее из одного листа, является 2-3-деревом. На рисунке приведены примеры 2-3-деревьев с шестью листьями.



Число узлов в 2-3-дереве связано с его высотой.

**Лемма.** Пусть  $T$  - 2-3-дерево высоты  $h$ . Число узлов дерева  $T$  заключено между  $2^{h+1}-1$  и  $(3^{h+1}-1)/2$ , а число листьев - между  $2^h$  и  $3^h$ .

**Доказательство** леммы проводится методом математической индукции по высоте дерева  $h$ .

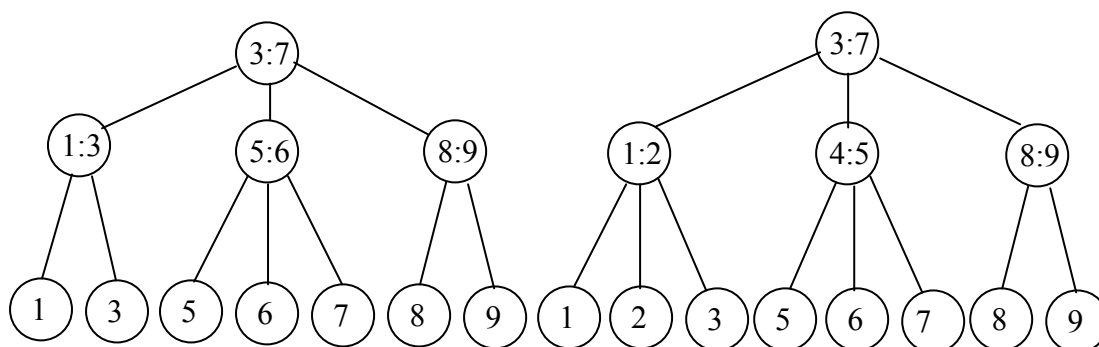
Пусть теперь  $S$  - множество, между элементами которого существует отношение порядка, обозначаемое  $\leq$ . Поставим в соответствие множеству  $S$  2-3-дерево, у которого листьям приписаны элементы  $S$  слева направо в порядке возрастания (как в приведенных примерах). Обозначим элемент множества  $S$ , приписанный листу  $l$ , через  $E[l]$ . В каждом узле  $v$ , не являющемся листом, запоминаются два элемента:  $L[v]$  и  $M[v]$ .  $L[v]$  - это наибольший элемент из  $S$  в поддереве, корнем которого служит самый левый сын узла  $v$ , а  $M[v]$  - это наибольший элемент из  $S$  в поддереве, корнем которого служит второй сын узла  $v$ . Значения  $L$  и  $M$ , приписываемые узлам, позволяют искать элемент, начиная с корня, способом, аналогичным двоичному поиску, причем время поиска пропорционально высоте дерева.

### 3. Вставка элемента в 2-3-дерево

Чтобы вставить в 2-3-дерево новый элемент  $a$ , надо найти место для нового листа  $l$ , который будет содержать  $a$ . Для этого ищут элемент  $a$  в дереве. Если дерево содержит более одного элемента, то поиск  $a$  окончится в узле  $f$ , имеющем двух или трех сыновей, которые являются листьями.

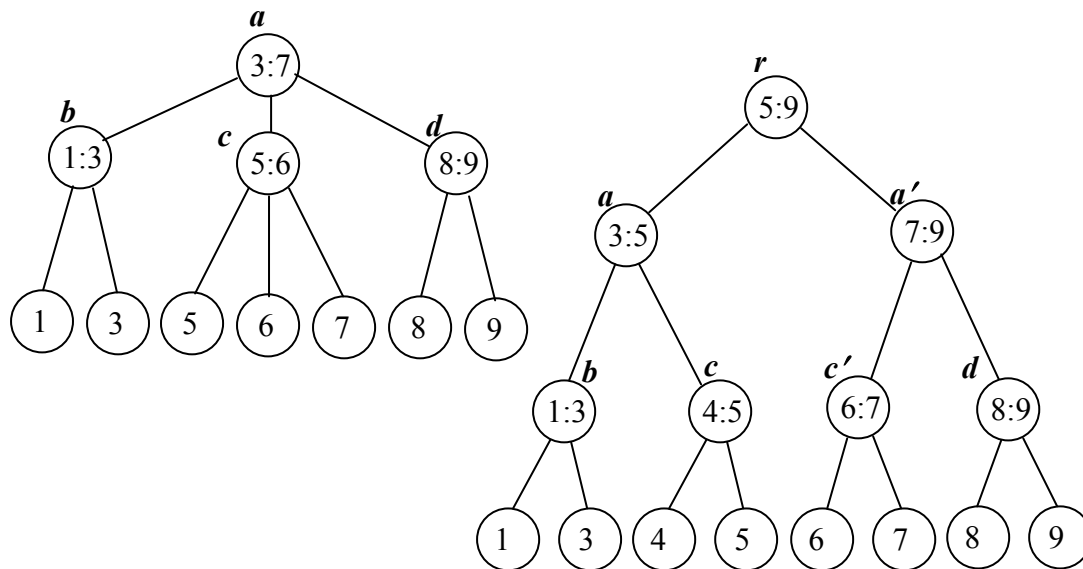
Если из узла  $f$  выходит только два листа  $l_1$  и  $l_2$ , то  $l$  делаем сыном узла  $f$ . Если  $a < E[l_1]$ , то  $l$  делаем самым левым сыном узла  $f$  и полагаем  $L[f] := a$  и  $M[f] := E[l_1]$ . Если  $E[l_1] < a < E[l_2]$ , то  $l$  делаем средним сыном узла  $f$  и полагаем  $M[f] := a$ . Наконец, если  $E[l_2] < a$ , то  $l$  делаем третьим сыном узла  $f$ . В этом последнем случае, возможно, надо будет изменить значения  $L$  и  $M$  на некоторых подлинных предках узла  $f$ .

В примере на рисунке элемент 2 вставляется в 2-3-дерево. Слева изображено дерево перед вставкой, а справа - после вставки.



Теперь предположим, что у узла  $f$  уже есть три листа  $l_1, l_2, l_3$ . Сделаем  $l$  соответствующим сыном узла  $f$ . Теперь  $f$  имеет четырех сыновей. Чтобы сохранить свойства 2-3-дерева, образуем новый узел  $g$ . Два левых сына оставим сыновьями узла  $f$ , а два правых сделаем сыновьями узла  $g$ . Затем сделаем  $g$  братом узла  $f$ , сделав его сыном отца узла  $f$ . Если отец узла  $f$  имел двух сыновей, то на этом мы остановимся. Если же трех, то надо рекурсивно повторять эту процедуру до тех пор, пока у всех узлов в дереве останется не более трех сыновей. Если мы дойдем до корня и у него окажется 4 сына, образуем новый корень с двумя новыми сыновьями, каждый из которых будет иметь в качестве двух своих сыновей двух из четырех сыновей старого корня.

В примере на рисунке элемент 4 вставляется в 2-3-дерево, изображенное слева. Сначала этот элемент надо сделать самым левым сыном узла  $c$ .



Поскольку у  $c$  уже есть три сына, строим новый узел  $c'$ . Затем делаем листья 4 и 5 сыновьями узла  $c$ , а листья 6 и 7 - сыновьями узла  $c'$ . Теперь делаем  $c'$  сыном узла  $a$ . Но поскольку у  $a$  уже есть три сына, строим новый узел  $a'$ . Делаем узлы  $b$  и  $c$  сыновьями старого узла  $a$ , а узлы  $c'$  и  $d$  - сыновьями нового узла  $a'$ . Наконец, образуем новый корень  $r$  и делаем  $a$  и  $a'$  его сыновьями. Полученное дерево изображено справа.

### Алгоритм вставки нового элемента в 2-3-дерево

```

PROCEDURE Insert(A: Element; T: Tree);
(* Эта процедура вставляет элемент A в 2-3-дерево T,
имеющее корень T.r Предполагается, что сначала A ∉ T *)
BEGIN
  IF (T состоит из единственного узла) THEN
  BEGIN Пусть этот узел l с элементом b.
    образуем новый корень r'.
    образуем новый узел v с элементом a.
    Делаем l и v сыновьями корня r':
      IF (b < a) THEN l будет левым сыном, v - правым
      ELSE v будет левым сыном, l - правым
    END
  ELSE (* T содержит более одного узла *)
  BEGIN
    f := Find(a, r);
    образуем новый лист l с элементом a.
    IF (f имеет два сына) THEN
    BEGIN
      Пусть эти сыновья связаны с элементами b1 и b2.
      Делаем l надлежащим сыном узла f:
        IF (a < b1) THEN левым
        ELSE IF (b1 < a < b2) - средним
    
```

```

        ELSE (* b2 < a *) - правым
    END
    ELSE (* у f три сына *)
    BEGIN
        AddSon(f, l); (* Включает в T узел l к отцу f,
                        имеющему трех сыновей *)
        Далее корректируем значения L и M вдоль пути из
        а в корень.
    END;
END;
END;

FUNCTION Find(a: Element; r: Knot): Knot;
(* поиск элемента a в поддереве, начиная с корня r *)
BEGIN
    IF (любой сын узла r является листом) THEN
        Result := r
    ELSE
        BEGIN
            (* r.s1-левый, r.s2-средний, r.s3-правый сыновья *)
            IF (a ≤ L[r]) THEN
                Result := Find(a, r.s1)
            ELSE IF (у r два сына) OR (a ≤ M[r]) THEN
                Result := Find(a, r.s2)
            ELSE Result := Find(a, r.s3)
        END;
    END;
END;

PROCEDURE AddSon(v, l: Knot);
(* присоединяет узел l к отцу v, у которого уже есть
три сына *)
BEGIN
    Делаем l надлежащим сыном узла f - четвертым.
    Образовать новый узел v'.
    Сделать двух самых правых сыновей узла v левым
    и правым сыновьями узла v'.
    IF (у узла v нет отца) THEN
        BEGIN
            Образовать новый корень r.
            Сделать v левым, а v' - правым сыновьями корня r.
        END
    ELSE
        BEGIN
            Пусть f - отец узла v.
            IF (у f всего два сына) THEN
                Сделать v' сыном узла f, расположенным
                непосредственно справа от v
            ELSE (* у f три сына *)
                AddSon(f, v');
        END;
    END;
END;

```

END;

В приведенном алгоритме не отражены коррекции значений  $L$  и  $M$ . Их предлагается написать в качестве самостоятельного упражнения.

Нетрудно видеть, что алгоритм обеспечивает вставку элемента в 2-3-дерево за время  $O(\log n)$ , где  $n$  - число листьев.

#### 4. Удаление элемента из 2-3-дерева

Элемент  $a$  можно удалить из 2-3-дерева методом, обратным вставке. Пусть  $l$  лист, содержащий элемент  $a$ . Рассмотрим отдельно 3 случая.

*Случай 1.* Если  $l$  корень - удаляем его. (В этом случае  $a$  был единственным элементом в дереве).

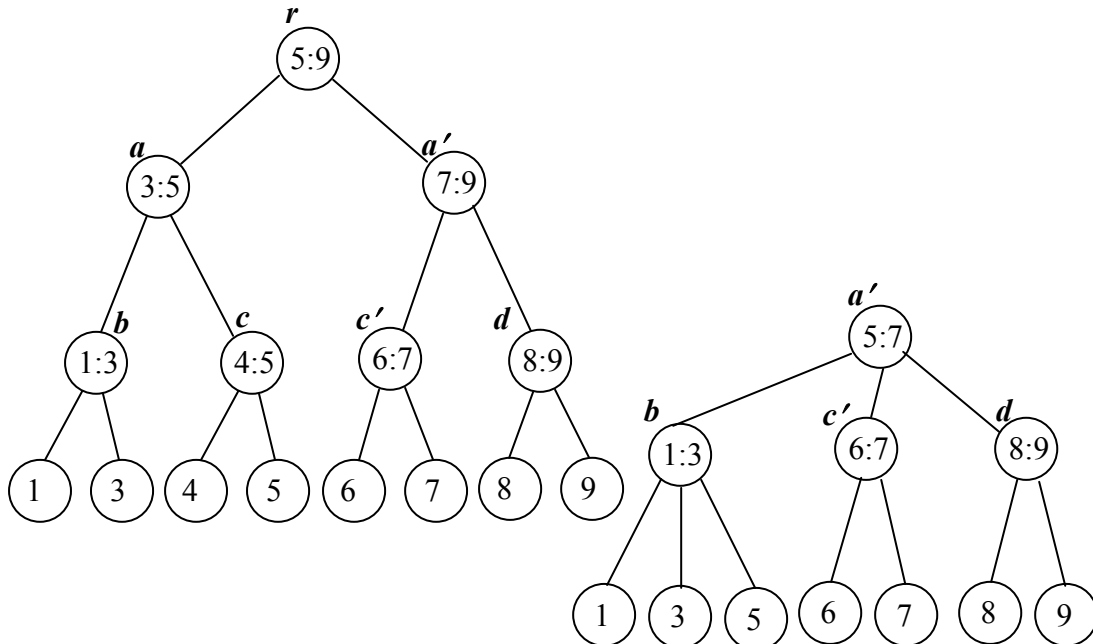
*Случай 2.* Если  $l$  сын узла, имеющего трех сыновей, удаляем его.

*Случай 3.* Если  $l$  сын узла, имеющего двух сыновей  $s$  и  $t$ , то может быть одно из двух:

А)  $f$  - корень. Удаляем  $l$  и  $f$  и делаем корнем второго сына  $s$ ;

Б)  $f$  - не корень. Допустим, что  $f$  имеет брата  $g$  слева от себя (если брат справа, то рассмотрение аналогичное). Если у  $g$  только два сына, то делаем узел  $s$  самым правым сыном узла  $g$ , удаляем  $l$  и рекурсивно вызываем процедуру удаления, чтобы удалить  $f$ . Если же у  $g$  три сына, то самого правого сына делаем левым сыном узла  $f$  и удаляем  $l$ .

В примере, представленном на рисунке, нужно из 2-3-дерева, изображенного слева, удалить элемент 4. Лист с элементом 4 является сыном узла  $c$ , у которого два сына. Поэтому делаем лист с меткой 5 самым правым



сыном узла  $b$ , удаляем лист с элементом 4 и затем рекурсивно удаляем узел  $c$ .

Узел  $c$  - сын узла  $a$ , у которого два сына. Узел  $a'$  - правый брат узла  $a$ . Поэтому делаем  $b$  самым левым сыном узла  $a'$ , удаляем  $c$  и затем рекурсивно удаляем  $a$ .

Узел  $a$  - сын корня. Применяя случай 3а, делаем  $a'$  корнем остающегося дерева. Полученное в результате дерево представлено на рисунке справа.

Нетрудно показать, что и операция удаления элемента из 2-3-дерева требует времени  $O(\log n)$ .

Таким образом, мы построили структуру данных, которая позволяет реализовать операции проверки принадлежности элемента множеству, вставки и удаления элемента за время  $O(\log n)$ .

Рассмотрим операцию поиска минимального элемента в множестве. Минимальный элемент в 2-3-дереве расположен в самом левом листе, который легко находится за  $O(\log n)$  шагов.