

Тема 2. СТРУКТУРЫ ДАННЫХ

2.5. Деревья

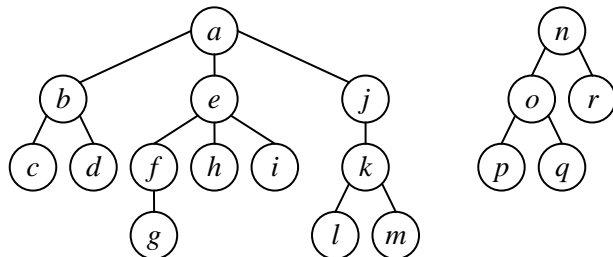
2.5.3. Прохождения деревьев

Во многих приложениях необходимо пройти лес, обрабатывая вершины каким-либо способом (в простейшем случае – печать содержимого узла) в некотором систематическом порядке. Предполагается, что при посещении вершин структура леса не меняется. Рассмотрим четыре основных способа прохождения: в прямом, обратном, горизонтальном и симметричном (обычно определен только для бинарных деревьев) порядках.

При прохождении в *прямом* порядке (известном также как прохождение *в глубину*), вершины леса просматриваются в соответствии со следующей рекурсивной процедурой:

1. Посетить корень первого дерева.
2. Пройти в прямом порядке поддеревья первого дерева, если они есть.
3. Пройти в прямом порядке оставшиеся деревья, если они есть.

Для леса на рис. 2.12 вершины будут проходиться в следующем порядке: $a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r$.



Для бинарных деревьев эта процедура упрощается и выглядит следующим образом (пустое дерево проходится без выполнения каких-либо действий):

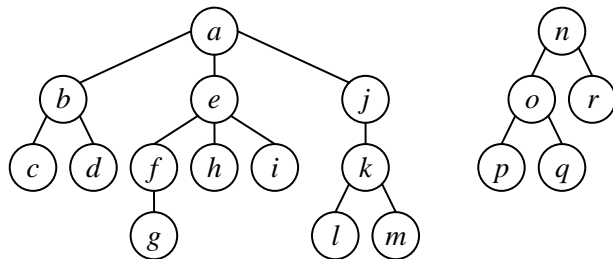
1. Посетить корень.
2. Пройти в прямом порядке левое поддерево.
3. Пройти в прямом порядке правое поддерево.

Следует обратить внимание на то, что прохождение леса в прямом порядке в точности соответствует прямому прохождению бинарного дерева, являющегося его представлением.

При *обратном* прохождении (известном также как прохождение *снизу вверх*) вершины леса проходятся в соответствии со следующей рекурсивной процедурой:

1. Пройти в обратном порядке поддерева первого дерева, если они есть.
2. Посетить корень первого дерева.
3. Пройти в обратном порядке оставшиеся деревья, если они есть.

Следует обратить внимание на то, что в момент посещения произвольной вершины все ее потомки уже пройдены. Для леса, изображенного на рис. 2.12, вершины проходятся в следующем порядке: $c, d, b, g, f, h, i, e, l, m, k, j, a, p, q, o, r, n$.



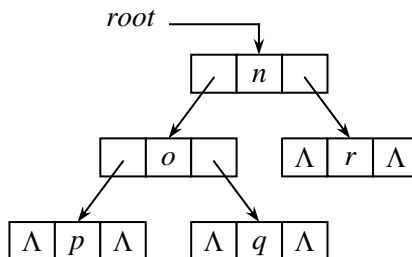
Для бинарных деревьев процедура имеет такой вид:

1. Пройти в обратном порядке левое поддерево.
2. Пройти в обратном порядке правое поддерево.
3. Посетить корень.

Симметричный порядок прохождения бинарных деревьев определяется следующим образом:

1. Пройти в симметричном порядке левое поддерево.
2. Посетить корень.
3. Пройти в симметричном порядке правое поддерево.

Для бинарного дерева на рис. 2.13 вершины будут проходиться в следующем порядке: p, o, q, n, r . Следует обратить внимание на то, что обратный порядок прохождения леса эквивалентен симметричному порядку прохождения соответствующего этому лесу бинарного дерева.



Рекурсивные процедуры прохождения бинарных деревьев очевидным образом можно записать в виде соответствующего псевдокода, например рекурсивная процедура прямого прохождения представлена алгоритмом 2.7.

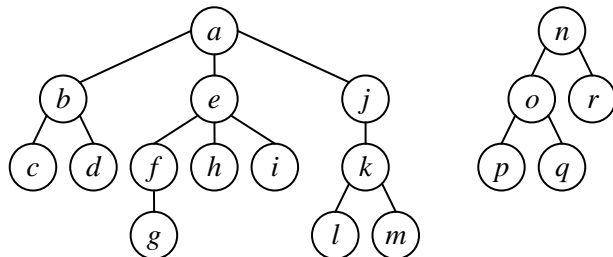
```

procedure PREORDER(p)
  if p ≠  $\Lambda$  then
    {
      visit(p)
      PREORDER(p.left)
      PREORDER(p.right)
    }
  return
  
```

Алгоритм 2.7. Рекурсивная процедура прямого прохождения бинарного дерева

Прохождение леса в *горизонтальном* порядке (известном также как прохождение *в ширину*) заключается в следующем. Вершины леса проходятся слева направо уровень за уровнем от корня вниз. Для леса, показанного на рис. 2.12, вершины проходятся так: *a, n, b, e, j, o, r, c, d, f, h, i, k, p, q, g, l, m*.

Если все вершины дерева пронумеровать в порядке посещения, то рассмотренные прохождения обладают рядом интересных свойств.



При нумерации в прямом порядке все вершины поддерева с корнем r имеют номера, не меньшие r . Если D_r – множество потомков вершины r (включая и саму вершину r), то v будет номером некоторой вершины из D_r , тогда и только тогда, когда $r \leq v < r + |D_r|$. Поставив в соответствие каждой вершине v ее номер в прямом порядке и количество ее потомков, легко определить, является ли некоторая вершина w потомком для v , за фиксированное время, не зависящее от размера дерева.

Номера, соответствующие обратному порядку, обладают аналогичным свойством.

Номера вершин бинарного дерева, соответствующие симметричному порядку, обладают тем свойством, что номера вершин в левом поддереве для вершины v меньше v , а в правом поддереве – больше v .

При сравнении рекурсивных процедур прохождения бинарных деревьев обнаруживается значительное их сходство. Это сходство позволяет построить общий нерекурсивный алгоритм, который может быть применен к каждому из этих прохождений. Для этого используется стек S для хранения пар, состоящих из указателя на узел бинарного дерева и целого i , значение которого указывает номер применяемой операции, когда пара достигнет вершины стека. Анализ процедур прохождения позволяет выделить три типа операций: посетить корень ($visit(p)$, где p – указатель на текущий узел), перейти к левому поддереву, что соответствует операции

if $p.left \neq \Lambda$ then $S \leftarrow (p.left, 1)$,

и перейти к правому поддереву, что соответствует операции

if $p.right \neq \Lambda$ then $S \leftarrow (p.right, 1)$.

Тогда общую процедуру прохождения бинарного дерева можно представить алгоритмом 2.8 [21].

$$\begin{array}{l}
S \leftarrow \emptyset \quad // \text{пустой стек} \\
S \leftarrow (root, 1) \\
\mathbf{while} \ S \neq \emptyset \ \mathbf{do} \ \left\{ \begin{array}{l}
(p, i) \leftarrow S \\
\mathbf{case} \ \left\{ \begin{array}{l}
i = 1: \begin{cases} S \leftarrow (p, 2) \\ \text{операция 1} \end{cases} \\
i = 2: \begin{cases} S \leftarrow (p, 3) \\ \text{операция 2} \end{cases} \\
i = 3: \text{операция 3} \end{array} \right.
\end{array} \right.
\end{array}$$

Алгоритм 2.8. Общий нерекурсивный алгоритм прохождения бинарного дерева

Непосредственная конкретизация общего нерекурсивного алгоритма для прохождения бинарного дерева в прямом порядке приведет к алгоритму 2.9. Очевидно, что полученный алгоритм неэффективен. В частности, после прохождения узла p (узла, на который указывает указатель p), когда $(p, 2)$ или $(p, 3)$ попадают

$$\begin{array}{l}
 S \leftarrow \emptyset \quad // \text{пустой стек} \\
 S \leftarrow (root, 1) \\
 \text{while } S \neq \emptyset \text{ do } \left\{ \begin{array}{l}
 (p, i) \leftarrow S \\
 \text{case } \left\{ \begin{array}{l}
 i = 1: \left\{ \begin{array}{l} S \leftarrow (p, 2) \\ \text{операция 1} \end{array} \right. \\
 i = 2: \left\{ \begin{array}{l} S \leftarrow (p, 3) \\ \text{операция 2} \end{array} \right. \\
 i = 3: \text{операция 3}
 \end{array} \right.
 \end{array} \right.
 \end{array}$$

$$\begin{array}{l}
 S \leftarrow \emptyset \quad // \text{пустой стек} \\
 S \leftarrow (root, 1) \\
 \text{while } S \neq \emptyset \text{ do } \left\{ \begin{array}{l}
 (p, i) \leftarrow S \\
 \text{case } \left\{ \begin{array}{l}
 i = 1: \left\{ \begin{array}{l} S \leftarrow (p, 2) \\ \text{visit}(p) \end{array} \right. \\
 i = 2: \left\{ \begin{array}{l} S \leftarrow (p, 3) \\ \text{if } p.left \neq \Lambda \text{ then } S \leftarrow (p.left, 1) \end{array} \right. \\
 i = 3: \text{if } p.right \neq \Lambda \\
 \quad \text{then } S \leftarrow (p.right, 1)
 \end{array} \right.
 \end{array} \right.
 \end{array}$$

Алгоритм 2.9. Конкретизация общего алгоритма для прямого прохождения

в вершину стека, единственное, что происходит, это $(p.left, 1)$ или $(p.right, 1)$ помещаются в стек. Эти шаги можно сделать раньше, когда в первый раз посещается узел p ; тогда отпадает необходимость трехкратного включения в стек указателя на каждый узел и, следовательно, сохранения в стеке номера i выполняемой операции. Поэтому этот алгоритм можно существенно упростить (алгоритм 2.10).

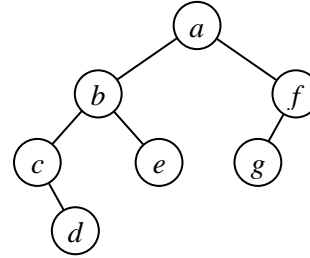
```

S ← ∅ // пустой стек
S ← root
while S ≠ ∅ do
  {
    p ← S
    visit(p)
    if p.right ≠ ∅ then S ← p.right
    if p.left ≠ ∅ then S ← p.left
  }

```

Алгоритм 2.10. Упрощенный вариант алгоритма 2.9

p	стек S	$Visit$	Операторы
	a		$S \leftarrow root$
a		a	$p \leftarrow S; visit(p)$
a	fb	a	$S \leftarrow p.right; S \leftarrow p.left$
b	f	ab	$p \leftarrow S; visit(p)$
b	fec	ab	$S \leftarrow p.right; S \leftarrow p.left$
c	fe	abc	$p \leftarrow S; visit(p)$
c	fed	abc	$S \leftarrow p.right$
d	fe	$abcd$	$p \leftarrow S; visit(p)$
e	f	$abcde$	$p \leftarrow S; visit(p)$
f		$abcdef$	$p \leftarrow S; visit(p)$
f	g	$abcdef$	$S \leftarrow p.left$
g		$abcdefg$	$p \leftarrow S; visit(p)$



$S \leftarrow \emptyset$ // пустой стек

$S \leftarrow root$

while $S \neq \emptyset$ **do** $\left\{ \begin{array}{l} p \leftarrow S \\ visit(p) \\ \text{if } p.right \neq \Lambda \text{ then } S \leftarrow p.right \\ \text{if } p.left \neq \Lambda \text{ then } S \leftarrow p.left \end{array} \right.$

Для полученного алгоритма можно продолжить процесс улучшений. В алгоритме указатель на каждый узел помещается в стек точно один раз. Однако можно сократить число операций со стеком. Если внимательно проанализировать процедуру прямого прохождения, то можно обнаружить, что в стек достаточно помещать только указатели на правых сыновей (если они есть), а по левым сыновьям продолжать продвижение, используя для этого рабочий указатель p (алгоритм 2.11).

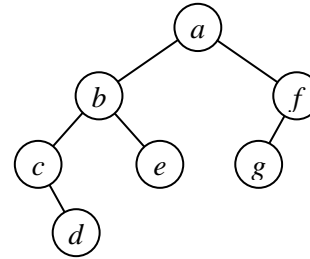
```

 $S \leftarrow \Lambda$  // занести в стек пустое значение указателя
 $p \leftarrow root$ 
while  $p \neq \Lambda$  do
   $\left\{ \begin{array}{l} \text{if } p.right \neq \Lambda \text{ then } S \leftarrow p.right \\ \text{if } p.left \neq \Lambda \text{ then } p \leftarrow p.left \text{ else } p \leftarrow S \end{array} \right.$ 

```

Алгоритм 2.11. Прямое прохождение бинарного дерева

p	стек S	$Visit$	Операторы
	Λ		$S \leftarrow \Lambda$
a	Λ	a	$p \leftarrow root; visit(p)$
b	Λf	a	$S \leftarrow p.right; p \leftarrow p.left$
b	Λf	ab	$visit(p)$
c	Λfe	ab	$S \leftarrow p.right; p \leftarrow p.left$
c	Λfe	abc	$visit(p)$
c	Λfed	abc	$S \leftarrow p.right$
d	Λfe	$abcd$	$p \leftarrow S; visit(p)$
e	Λf	$abcde$	$p \leftarrow S; visit(p)$
f	Λ	$abcdef$	$p \leftarrow S; visit(p)$
Λ		$abcdef$	$p \leftarrow S$



$S \leftarrow \Lambda$ // занести в стек пустое значение указателя
 $p \leftarrow root$

while $p \neq \Lambda$ **do** $\begin{cases} visit(p) \\ \text{if } p.right \neq \Lambda \text{ then } S \leftarrow p.right \\ \text{if } p.left \neq \Lambda \text{ then } p \leftarrow p.left \text{ else } p \leftarrow S \end{cases}$

Конкретизировав соответствующим образом общий алгоритм и упростив его, можно получить нерекурсивный алгоритм симметричного прохода бинарного дерева (алгоритм 2.12).

```

S ← ∅ // пустой стек
p ← root

while p ≠ ∅ do {
  while p.left ≠ ∅ do {
    S ← p
    p ← p.left
  }
  visit(p)
  while p.right = ∅ and S ≠ ∅ do {
    p ← S
    visit(p)
  }
  p ← p.right
}

```

Алгоритм 2.12. Симметричное прохождение бинарного дерева

Для получения нерекурсивного алгоритма обратного прохода достаточно применить соответствующую прямую конкретизацию общего алгоритма, поскольку существенных улучшений в этом случае добиться трудно.

Для реализации горизонтального прохождение бинарного дерева необходимо использовать очередь. При посещении некоторого узла его сыновья (если они есть) помещаются в очередь: сначала левый сын, а затем – правый. Детали реализации горизонтального прохождение представлены алгоритмом 2.13.

```

 $Q \leftarrow \emptyset$  // пустая очередь
 $Q \leftarrow root$ 

while  $Q \neq \emptyset$  do
   $p \leftarrow Q$ 
   $visit(p)$ 
  if  $p.left \neq \Lambda$  then  $Q \leftarrow p.left$ 
  if  $p.right \neq \Lambda$  then  $Q \leftarrow p.right$ 

```

Алгоритм 2.13. Горизонтальное прохождение бинарного дерева

2.5.4. Прошитые бинарные деревья

С целью повышения эффективности прохождение бинарных деревьев можно использовать так называемые *прошитые* деревья. В нижней части бинарного дерева с n узлами всегда имеется $n + 1$ полей указателей со значением Λ . Можно воспользоваться этим пространством пустых значений следующим образом. Пустые значения полей *left* заменяются указателями на предшествующий узел при прохождении в симметричном порядке, а пустые значения полей *right* – на последующий узел при прохождении в симметричном порядке. Полученные таким образом связи называются *нитями*, а само бинарное дерево – *симметрично прошитым* бинарным деревом. Ясно, что в узлах должны быть предусмотрены специальные средства, чтобы отличать нити от обычных связей (например, добавить дополнительный разряд к полям *left* и *right*). Пример симметрично прошитого бинарного дерева показан на рис. 2.16 (нити изображены пунктирными линиями).

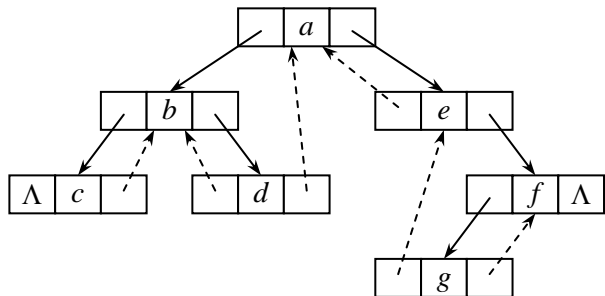


Рис. 2.16. Симметрично прошитое бинарное дерево

Симметричный порядок: $cbdaegf$

Аналогичным образом можно определить *прямопрошитые* бинарные деревья, в которых пустые поля *left* и *right* заменяются указателями соответственно на предшественников и преемников при прямом прохождении.

Наличие нитей в симметрично прошитом дереве позволяет повысить эффективность алгоритмов прохождения в прямом и симметричном порядках за счет исключения операций со стеками, т. е. эти прохождения можно реализовать без использования стека. Построение соответствующих алгоритмов предлагается в качестве упражнений.