Структуры данных
Деревья
Прохождения деревьев

Доцент кафедры ВТ ЧувГУ

Павлов Леонид Александрович

Содержание лекции

- Что такое прохождение деревьев?
- Прохождения деревьев (прямое, обратное, симметричное, горизонтальное
- Прошитые деревья

Деревья. 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 (обхода):

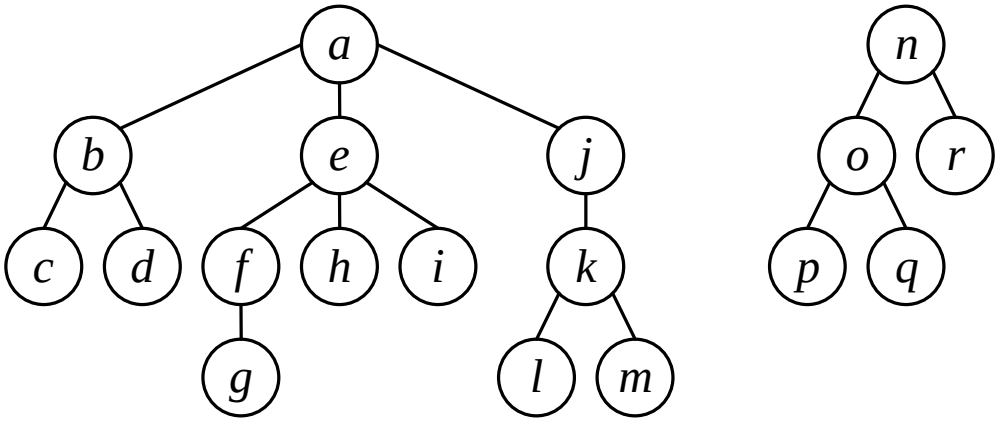
- ✓ в прямом порядке,
- ✓ в обратном порядке,
- ✓ в горизонтальном порядке
- ✓ в симметричном порядке (обычно определен только для бинарных деревьев)

Прохождение леса в прямом порядке (pre-order)

При прохождении леса в *прямом* порядке (известном также как прохождение *сверху вниз*, *pre-order*), 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 приведенного леса вершины будут проходиться в следующем порядке: *a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r.*

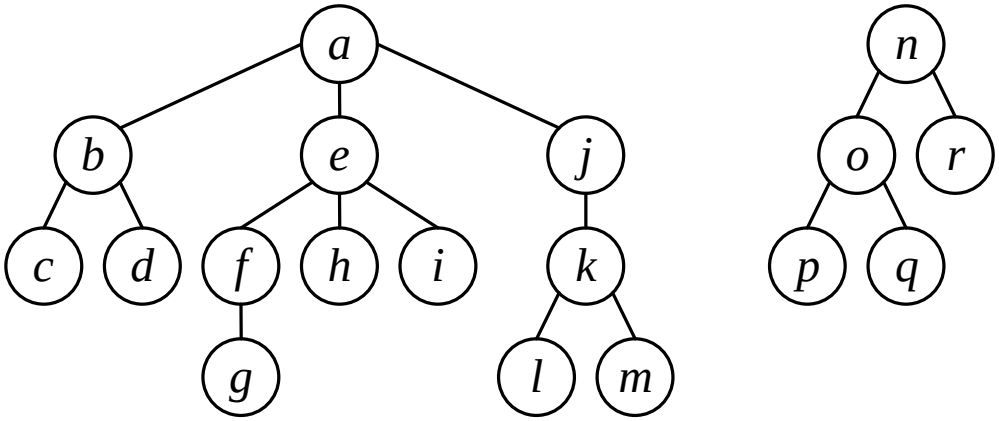


Прохождение леса в обратном порядке (post-order)

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

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

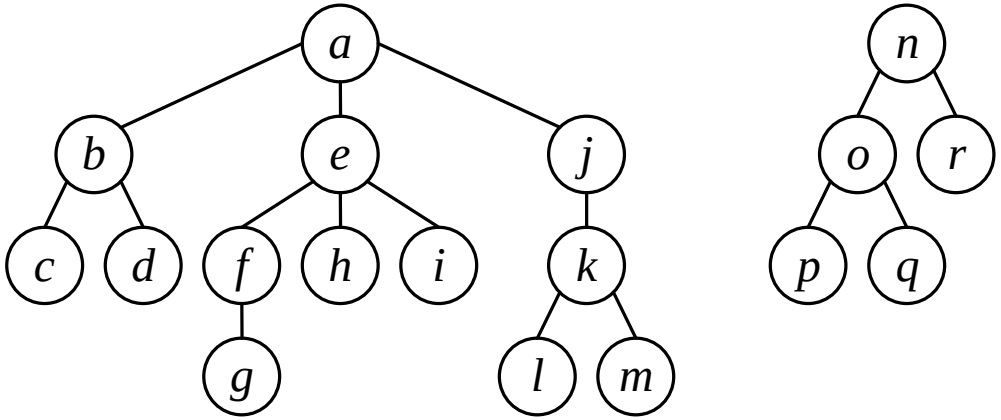
Следует обратить внимание на то, что в момент посещения произвольной вершины все ее потомки уже пройдены. Для приведенного леса вершины проходятся в следующем порядке: *c*, *d*, *b*, *g*, *f*, *h*, *i*, *e*, *l*, *m*, *k*, *j*, *a*, *p*, *q*, *o*, *r*, *n*.



Прохождение леса в горизонтальном порядке

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 приведенного леса вершины проходятся в следующем порядке:
a, *n*, *b*, *e*, *j*, *o*, *r*, *c*, *d*, *f*, *h*, *i*, *k*, *p*, *q*, *g*, *l*, *m*.



Прохождения бинарных деревьев

Традиционно для бинарных деревьев рассматривают следующие прохождения: прямое, обратное, симметричное и горизонтальное. Причем прямое, обратное и симметричное прохождения легко определяются рекурсивно. У этих рекурсивных алгоритмов есть много общего (пусть текущая вершина N является корнем соответствующего поддерева):

- ✓ рекурсивное прохождение левого поддерева (обозначим L),
- ✓ рекурсивное прохождение правого поддерева (обозначим R),
- ✓ посещение (обработка) корня N (обозначим N).

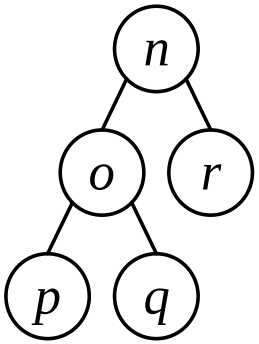
Порядок выполнения этих действий определяется соответствующим прохождением. Поскольку принято исследовать левое поддерево раньше правого, имеем три возможных порядка: NLR (прямое прохождение), LRN (обратное прохождение) и LNR (симметричное прохождение). Довольно часто данная аббревиатура используется для обозначения соответствующего прохождения.

Прямое прохождение бинарного дерева

При прохождении бинарного дерева в *прямом* порядке (*сверху вниз*, *pre-order*, *NLR*), вершины просматриваются в соответствии со следующей рекурсивной процедурой:

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

Для приведенного бинарного дерева вершины будут проходиться в следующем порядке: *n*, *o*, *p*, *q*, *r*.



Рекурсивную процедуру прямого прохождения бинарного дерева очевидным образом можно записать в виде соответствующего псевдокода:

procedure $PREORDER(p)$

if $p \neq \Lambda$ **then** $\begin{cases} visit(p) \\ PREORDER(p.left) \\ PREORDER(p.right) \end{cases}$

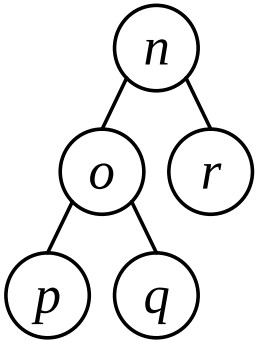
return

Обратное прохождение бинарного дерева

При прохождении бинарного дерева в *обратном* порядке (*снизу вверх*, *post-order*, *LRN*), вершины просматриваются в соответствии со следующей рекурсивной процедурой:

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

Для приведенного бинарного дерева вершины будут проходиться в следующем порядке: *p*, *q*, *o*, *r*, *n*.



Рекурсивную процедуру обратного прохождения бинарного дерева очевидным образом можно записать в виде соответствующего псевдокода:

procedure $POSTORDER(p)$

if $p \neq \Lambda$ **then** $\begin{cases} POSTORDER(p.left) \\ POSTORDER(p.right) \\ visit(p) \end{cases}$

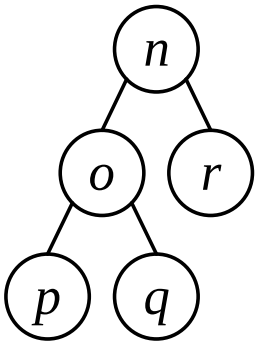
return

Симметричное прохождение бинарного дерева

При прохождении бинарного дерева в *симметричном* порядке (*центрированный обход*, *in-order*, *LNR*), вершины просматриваются в соответствии со следующей рекурсивной процедурой:

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

Для приведенного бинарного дерева вершины будут проходиться в следующем порядке: *p*, *o*, *q*, *n*, *r*.



Рекурсивную процедуру симметричного прохождения бинарного дерева очевидным образом можно записать в виде соответствующего псевдокода:

procedure $INORDER(p)$

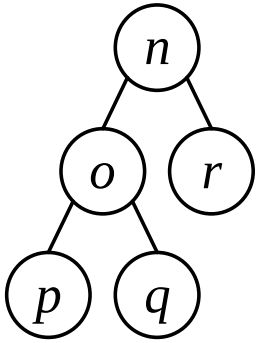
if $p \neq \Lambda$ **then** $\begin{cases} INORDER(p.left) \\ visit(p) \\ INORDER(p.right) \end{cases}$

return

Горизонтальное прохождение бинарного дерева

Прохождение бинарного дерева в *горизонтальном* порядке (*в ширину*) также, как и для леса, заключается в следующем: вершины дерева проходятся слева направо уровень за уровнем от корня вниз.

Для приведенного бинарного дерева вершины проходятся в следующем порядке: *n*, *o*, *r*, *p*, *q*.



Для реализации горизонтального прохождения 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 прохождения представлены в алгоритме:

$Q \leftarrow \varnothing$ // пустая очередь

$Q \Leftarrow root$

while $Q \neq \varnothing$ **do** $\begin{cases} p \Leftarrow Q \\ visit(p) \\ \textbf{if} \;\; p.left \neq \Lambda \;\; \textbf{then} \;\; Q \Leftarrow p.left \\ \textbf{if} \;\; p.right \neq \Lambda \;\; \textbf{then} \;\; Q \Leftarrow p.right \end{cases}$

Нерекурсивные алгоритмы прохождения бинарных деревьев

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

$$\textbf{if } p.left \neq \Lambda \textbf{ then } S \Leftarrow (p.left, 1),$$

- ✓ перейти к правому поддереву, что соответствует операции

$$\textbf{if } p.right \neq \Lambda \textbf{ then } S \Leftarrow (p.right, 1).$$

Тогда общую процедуру прохождения бинарного дерева можно представить алгоритмом, показанным на следующем слайде.

Общая нерекурсивная процедура прохождения бинарных деревьев

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

Для получения нерекурсивного алгоритма требуемого прохождения необходима его конкретизация.

Рассмотрим пример конкретизации для прямого прохождения.

Конкретизация общая нерекурсивной процедуры для прямого прохождения

$S \leftarrow \varnothing$ // пустой стек
$S \Leftarrow (root, 1)$

$$\textbf{while } S \neq \varnothing \textbf{ do} \begin{cases} (p, i) \Leftarrow S \\ \textbf{case} \begin{cases} i = 1: \begin{cases} S \Leftarrow (p, 2) \\ \text{операция 1} \end{cases} \\ i = 2: \begin{cases} S \Leftarrow (p, 3) \\ \text{операция 2} \end{cases} \\ i = 3: \text{ операция 3} \end{cases} \end{cases}$$

Очевидно, что полученный алгоритм неэффективен. В частности, после прохождения узла *p* (узла, на который указывает указатель *p*), когда (*p*, 2) или (*p*, 3) попадают 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 (см. след. слайд).

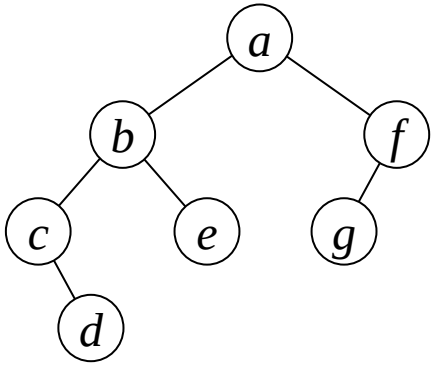
$S \leftarrow \varnothing$ // пустой стек
$S \Leftarrow (root, 1)$

$$\textbf{while } S \neq \varnothing \textbf{ do} \begin{cases} (p, i) \Leftarrow S \\ \textbf{case} \begin{cases} i = 1: \begin{cases} S \Leftarrow (p, 2) \\ visit(p) \end{cases} \\ i = 2: \begin{cases} S \Leftarrow (p, 3) \\ \textbf{if } p.left \neq \Lambda \textbf{ then } S \Leftarrow (p.left, 1) \end{cases} \\ i = 3: \textbf{ if } p.right \neq \Lambda \\ \qquad \textbf{then } S \Leftarrow (p.right, 1) \end{cases} \end{cases}$$

Упрощенный алгоритм прямого прохождения

$S \leftarrow \varnothing$ // пустой стек
$S \Leftarrow root$

$$\textbf{while } S \neq \varnothing \textbf{ do } \begin{cases} p \Leftarrow S \\ visit(p) \\ \textbf{if } p.right \neq \Lambda \textbf{ then } S \Leftarrow p.right \\ \textbf{if } p.left \neq \Lambda \textbf{ then } S \Leftarrow p.left \end{cases}$$



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

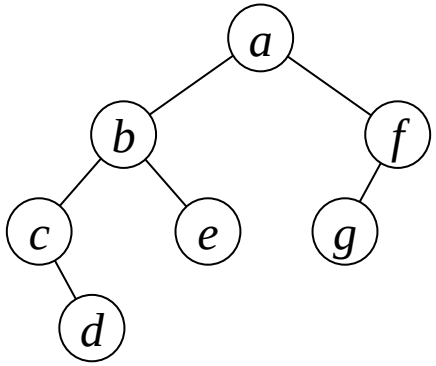
Для полученного алгоритма можно продолжить процесс улучшений. В алгоритме указатель на каждый узел помещается в стек точно один раз. Однако можно сократить число операций со стеком. Если внимательно проанализировать процедуру прямого прохождения, то можно обнаружить, что в стек достаточно помещать только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

Улучшенный алгоритм прямого прохождения

$S \Leftarrow \Lambda$ // занести в стек пустое значение указателя

$p \leftarrow root$

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



p	стек S	Visit	Операторы
	Λ		S ⇐ Λ
a	Λ	a	p ← root; visit(p)
b	Λf	a	S ⇐ p.right; p ← p.left
b	Λf	ab	visit(p)
c	Λfe	ab	S ⇐ p.right; p ← p.left
c	Λfe	abc	visit(p)
c	Λfed	abc	S ⇐ p.right
d	Λfe	abcd	p ⇐ S; visit(p)
e	Λf	abcde	p ⇐ S; visit(p)
f	Λ	abcdef	p ⇐ S; visit(p)
g	Λ	abcdefg	p ← p.left; visit(p)
Λ		abcdefg	p ⇐ S

Нерекурсивный алгоритм симметричного прохождения бинарного дерева

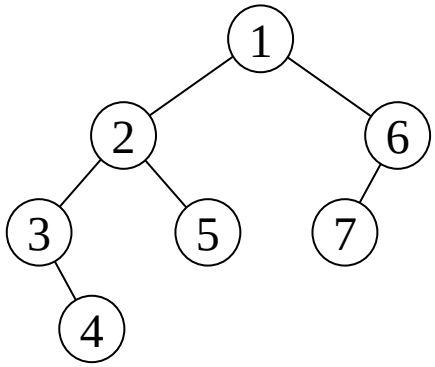
Конкретизировав соответствующим образом общий алгоритм и упростив его, можно получить нерекурсивный алгоритм симметричного прохождения бинарного дерева.

$$
\begin{array}{l}
S \leftarrow \varnothing \quad \text{// пустой стек} \\
p \leftarrow root \\
\textbf{while}\ \ p \neq \Lambda\ \ \textbf{do}
\begin{cases}
\textbf{while}\ \ p.left \neq \Lambda\ \ \textbf{do}
\begin{cases} S \Leftarrow p \\ p \leftarrow p.left \end{cases} \\
visit(p) \\
\textbf{while}\ \ p.right = \Lambda\ \textbf{and}\ S \neq \varnothing\ \ \textbf{do}
\begin{cases} p \Leftarrow S \\ visit(p) \end{cases} \\
p \leftarrow p.right
\end{cases}
\end{array}
$$

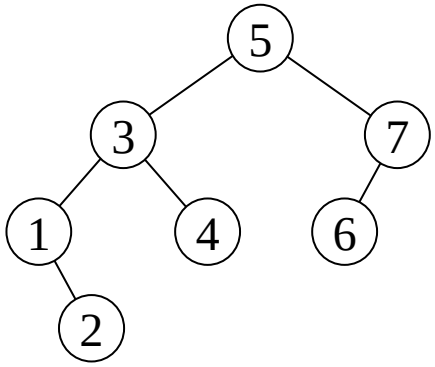
Для получения нерекурсивного алгоритма обратного прохождения достаточно применить соответствующую прямую конкретизацию общего алгоритма, поскольку существенных улучшений в этом случае добиться трудно.

Свойства прохождений

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 $r \le v < r + |D_r|$. 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.

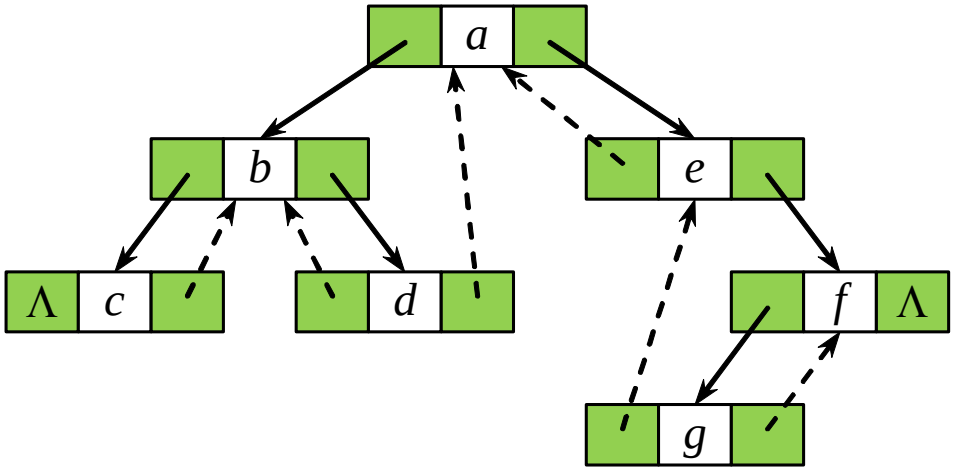
Прошитые бинарные деревья

С целью повышения эффективности прохождения 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 следующем слайде.

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

Пример симметрично прошитого бинарного дерева



В примере нити изображены пунктирными линиями.

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

Спасибо за внимание!