▷ **5.49** Приведите дерево, соответствующее рис. 5.18 при допущении, что элементы рассматриваются в порядке уменьшения их размеров.

● **5.50** Докажите лемму 5.3.

○ **5.51** Создайте функцию, решающую задачу о ранце, используя версию программы 5.12, в которой применяется восходящее динамическое программирование.

● **5.52** Создайте функцию, которая решает задачу о ранце методом нисходящего динамического программирования, но используя при этом рекурсивное решение, основывающееся на вычислении оптимального количества конкретного элемента, который должен быть помещен в ранец, исходя из определения (рекурсивно) оптимального способа упаковки ранца без этого элемента.

○ **5.53** Создайте функцию, решающую задачу о ранце, используя версию рекурсивного решения, описанного в упражнении 5.52, в которой применяется восходящее динамическое программирование.

● **5.54** Воспользуйтесь динамическим программированием для решения упражнения 5.4. Обеспечьте отслеживание общего количества сохраняемых вызовов функций.

5.55 Создайте программу, в которой нисходящее динамическое программирование применяется для вычисления биномиального коэффициента $\binom{N}{k}$, исходя из рекурсивного соотношения

$$\binom{N}{k}=\binom{N-1}{k}+\binom{N-1}{k-1}, \text{ при } \binom{N}{0}=\binom{N}{N}=1.$$

5.4 Деревья

Деревья — это математические абстракции, играющие главную роль при разработке и анализе алгоритмов, поскольку

- мы используем деревья для описания динамических свойств алгоритмов;
- мы строим и используем явные структуры данных, которые являются конкретными реализациями деревьев.

Мы уже встречались с примерами обоих применений деревьев. В главе 1 были разработаны алгоритмы для решения задачи подключения, которые основывались на структурах деревьев, а в разделах 5.2 и 5.3 структура вызовов рекурсивных алгоритмов была описана с помощью структур деревьев.

Мы часто встречаемся с деревьями в повседневной жизни — это основное понятие очень хорошо знакомо. Например, многие люди отслеживают предков и наследников с помощью генеалогического дерева; как будет показано, значительная часть терминов заимствована именно из этой области применения. Еще один пример — организация спортивных турниров; среди прочих, в исследовании этого применения принял участие и Льюис Кэрролл (Lewis Carroll). В качестве третьего примера можно привести организационную диаграмму большой корпорации; это применение отличается иерархическим разделением, характерным для алгоритмов типа "разделяй и властвуй". Четвертым примером служит дерево синтаксического разложения предло-

жения английского (или любого другого языка) на составляющие его части; такое дерево близко связано с обработкой компьютерных языков, как описано в части 5. Типичный пример дерева, в данном случае описывающего структуру этой книги, показан на рис. 5.19. В книге рассматривается и множество других примеров применения деревьев.

Применительно к компьютерам, одно из наиболее известных применений структур деревьев — для организации файловых систем. Файлы хранятся в *каталогах* (иногда называемых также *папками*), которые рекурсивно определяются как последовательности каталогов и файлов. Это рекурсивное определение снова отражает естественное рекурсивное разбиение на составляющие и идентично определению определенного типа дерева.

Существует множество различных типов деревьев, и важно понимать различие между абстракцией и конкретным представлением, с которым выполняется работа для данного приложения. Соответственно, мы подробно рассмотрим различные типы деревьев и их представления. Рассмотрение начнется с определения деревьев как абстрактных объектов и с ознакомления с большинством основных связанных с ними терминов. Мы неформально рассмотрим различные типы деревьев, которые следует рассматривать в порядке сужения самого этого понятия:

- Деревья
- Деревья с корнем
- Упорядоченные деревья
- М-арные и бинарные деревья

После этого неформального рассмотрения мы перейдем к формальным определениям и рассмотрим представления и приложения. На рис. 5.20 приведена иллюстрация многих из этих базовых концепций, которые будут сначала рассматриваться, а затем и определяться.

Дерево (tree) — это непустая коллекция вершин и ребер, удовлетворяющих определенным требованиям. *Вершина (vertex)* — это простой объект (называемый также *узлом (node)*), который может иметь имя и содержать другую связанную с ним информацию; *ребро (edge)* — это связь между двумя вершинами. *Путь (path)* в дереве — это список отдельных вершин, в котором следующие друг за другом вершины соединяются ребрами дерева. Определяющее свойство дерева — существование только одно пути, соединяющего любые два узла. Если между какой-либо парой узлов существует более одного пути или если между какой-либо парой узлов путь отсутствует, мы имеем граф, а не дерево. Несвязанный набор деревьев называется *бором (forest)*.

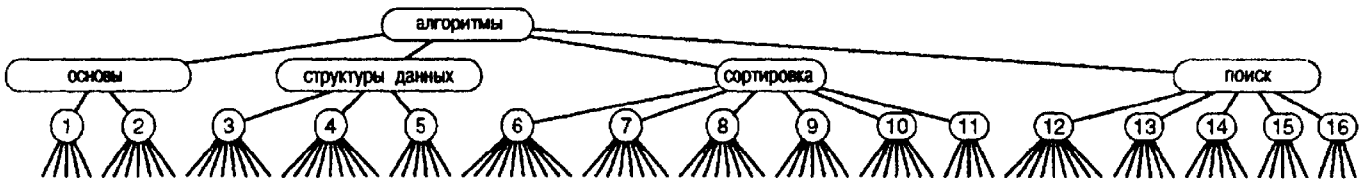


РИСУНОК 5.19 ДЕРЕВО

Это дерево описывает части, главы и разделы этой книги. Каждый элемент представлен узлом. Каждый узел связан нисходящими связями с составляющими его частями и восходящей связью — с большей частью, к которой он принадлежит.

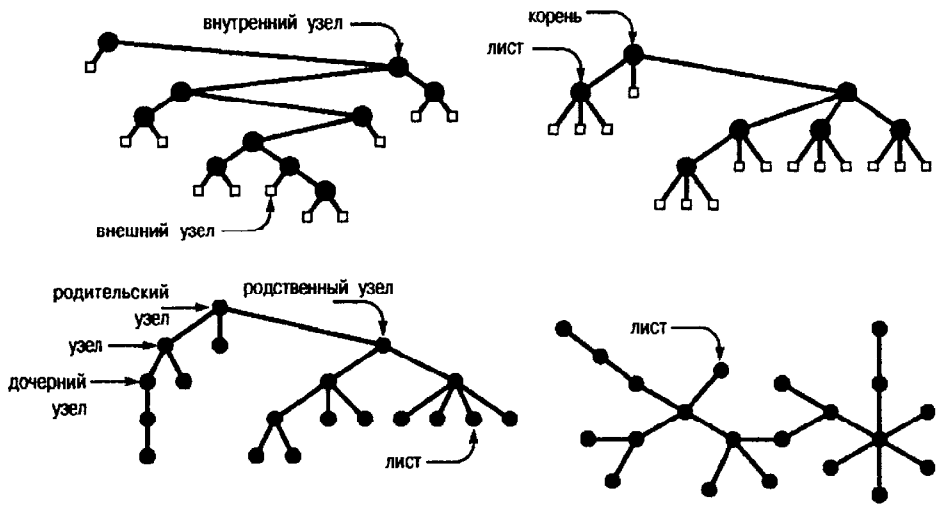


РИСУНОК 5.20 ТИПЫ ДЕРЕВЬЕВ
На этих схемах приведены примеры двоичного дерева (вверху слева), троичного дерева (вверху справа), дерева с корнем (внизу слева) и свободного дерева (внизу справа).

Дерево с корнем (единственным, rooted) — это дерево, в котором один узел назначен *корнем (root)* дерева. В области компьютеров термин *дерево* обычно применяется к деревьям с корнем, а термин *свободное дерево (free tree)* — к более общим структурам, описанным в предыдущем абзаце. В дереве с корнем любой узел является корнем поддерева, состоящего из него и расположенных под ним узлов.

Существует только один путь между корнем и каждым из других узлов дерева. Данное определение никак не определяет направление ребер; обычно считается, что все ребра указывают от корня или к корню, в зависимости от приложения. Обычно деревья с корнем рисуются с корнем, расположенным в верхней части (хотя на первый взгляд это соглашение кажется неестественным), и говорят, что узел *y* располагается *под* узлом *x* (а *x* располагается над *y*), если *x* находится на пути от *y* к корню (т.е., *y* находится под *x*, как нарисовано на странице, и соединяется с *x* путем, который не проходит через корень). Каждый узел (за исключением корня) имеет только один узел над ним, который называется его *родительским узлом (parent)*; узлы, расположенные непосредственно под данным узлом, называются его *дочерними узлами (children)*. Иногда аналогия с генеалогическими деревьями расширяется еще больше и тогда говорят об узлах-*предках (grand parent)* или *родственных (sibling)* узлах данного узла.

Узлы, не имеющие дочерних узлов, называются *листьями (leaves)* или *терминальными (оконечными, terminal)* узлами. Для соответствия с последним применением узлы, имеющие хотя бы один дочерний узел, иногда называются *нетерминальными (nonterminal)* узлами. В этой главе мы уже встречались с примером утилиты, различающей эти типы узлов. В деревьях, которые использовались для представления структуры вызовов рекурсивных алгоритмов (например, рис. 5.14), нетерминальные узлы (окружности) представляют вызовы функций с рекурсивными вызовами, а терминальные узлы (квадраты) представляют вызовы функций без рекурсивных вызовов.

В определенных приложениях способ упорядочения дочерних узлов каждого узла имеет значение; в других это не важно. *Упорядоченное (ordered)* дерево — это дерево с корнем, в котором определен порядок следования дочерних узлов каждого узла. Упорядоченные деревья — естественное представление: например, при рисовании дерева дочерние узлы размещаются в определенном порядке. Действительно, многие другие конкретные представления имеют аналогично предполагаемый порядок; на-

пример, обычно это различие имеет значение при рассмотрении представления деревьев в компьютере.

Если каждый узел *должен* иметь конкретное количество дочерних узлов, появляющихся в конкретном порядке, мы имеем *M-арное дерево.* В таком дереве часто можно определить специальные внешние узлы, которые не имеют дочерних узлов. Тогда внешние узлы могут действовать в качестве фиктивных, на которые ссылаются узлы, не имеющие указанного количества дочерних узлов. В частности, простейшим типом *M*-арного дерева является бинарное дерево. *Бинарное дерево (binary tree)* — это упорядоченное дерево, состоящее из узлов двух типов: внешних узлов без дочерних узлов и внутренних узлов, каждый из которых имеет ровно два дочерних узла. Поскольку два дочерних узла каждого внутреннего узла упорядочены, говорят о *левом дочернем узле (left child)* и *правом дочернем узле (right child)* внутренних узлов. Каждый внутренний узел должен иметь и левый, и правый дочерние узлы, хотя один из них или оба могут быть внешними узлами. *Лист* в *M*-арном дереве — это внутренний узел, все дочерние узлы которого являются внешними.

Все это общая терминология. Далее рассматриваются формальные определения, представления и приложения, в порядке расширения понятий:

- бинарные и *M*-арные деревья
- упорядоченные деревья
- деревья с корнем
- свободные деревья

Начав с наиболее характерной абстрактной структуры, мы должны быть в состоянии подробно рассмотреть конкретные представления, как станет понятно из дальнейшего изложения.

Определение 5.1 ***Бинарное дерево*** *— это либо внешний узел, либо внутренний узел, связанный с парой бинарных деревьев, которые называются левым и правым поддеревьями этого узла.*

Из этого определения становится понятно, что сами деревья — абстрактное математическое понятие. При работе с компьютерными представлениями мы работаем всего лишь с одной конкретной реализацией этой абстракции. Эта ситуация не отличается от представления действительных чисел значениями типа **float**, целых чисел значениями типа **int** и т.д. Когда мы рисуем дерево с узлом в корне, связанным ребрами с левым поддеревом, расположенным слева, и с правым поддеревом, расположенным справа, то выбираем удобное конкретное представление. Существует множество различных способов представления бинарных деревьев (см., например, упражнение 5.62), которые поначалу кажутся удивительными, но вполне отражают сущность, как и можно было ожидать, учитывая абстрактный характер определения.

Чаще всего применяется следующее конкретное представление при реализации программ, использующих и манипулирующих бинарными деревьями, — структура с двумя связями (правой и левой) для внутренних узлов (см. рис. 5.21). Эти структуры аналогичны связным спискам, но они имеют по две связи для каждого узла, а не по одной. Нулевые связи соответствуют внешним узлам. В частности, мы добавили связь

к стандартному представлению связного списка, приведенному в разделе 3.3, следующим образом:

```
struct node { Item item; node *l, *r; }
typedef node *link;
```

что представляет собой всего лишь код C++ для определения 5.1. Узлы состоят из элементов и пар указателей на узлы, и указатели на узлы называются также связями. Так, например, мы реализуем абстрактную операцию *перехода к левому поддереву* с помощью ссылки на указатель типа **x = x->l**.

Это стандартное представление позволяет построить эффективную реализацию операций, которые вызываются для перемещения по дереву *вниз* от корня, но не для перемещения по дереву *вверх* от дочернего узла к его родительскому узлу. Для алгоритмов, требующих использования таких операций, можно добавить третью связь для каждого узла, направленную к его родительскому узлу. Эта альтернатива аналогична двухсвязным спискам. Как и в случае со связными списками (см. рис. 3.6), в определенных ситуациях узлы дерева хранятся в массиве, а в качестве связей используются индексы, а не указатели. Конкретный пример такой реализации исследуется в разделе 12.7. Для определенных специальных алгоритмов используются другие представления бинарных деревьев, что наиболее полно исследуется в главе 9.

Из-за наличия такого множества различных возможных представлений можно было бы разработать ADT (Abstract Data Type) (абстрактный тип данных) бинарного дерева, инкапсулирующий важные операции, которые нужно выполнять, и разделяющий использование и реализацию этих операций. В данной книге данный подход не используется, поскольку

- чаще всего мы используем представление с двумя связями;
- мы используем деревья для реализации ADT более высокого уровня, и хотим сосредоточить внимание на этой теме;
- мы работаем с алгоритмами, эффективность которых зависит от конкретного представления, — это обстоятельство может быть упущено в ADT.

По этим же причинам мы используем уже знакомые конкретные представления массивов и связных списков. Представление бинарного дерева, отображенное на рис. 5.21 — один из фундаментальных инструментов, который теперь добавлен к этому краткому списку.

Анализируя связные списки, мы начали рассмотрение с элементарных операций вставки и удаления узлов (см. рис. 3.3 и 3.4). При использовании стандартного представления бинарных деревьев такие операции необязательно являются элементарными из-за наличия второй связи. Если нужно удалить узел из бинарного дерева, приходится решать принципиальную проблему наличия двух дочерних узлов и только одного родительского, с которыми нужно работать после удаления узла. Существуют три естественных операции, для которых подобное осложнение не возникает: вставка нового узла в нижней части дерева (замена нулевой связи связью с новым узлом), удаление листа (замена связи с ним нулевой связью) и объединение двух деревьев посредством создания нового корня, левая связь которого указывает на одно дерево, а правая — на другое. Эти операции интенсивно используются при манипулировании бинарными деревьями.

Определение 5.2 ***M-арное дерево*** *— это внешний узел, либо внутренний узел, связанный с упорядоченной последовательностью M деревьев, которые также являются M-арными деревьями.*

Обычно узлы в *M*-арных деревьях представляются либо в виде структур с *M* именованными связями (как в бинарных деревьях), либо в виде массивов *M* связей. Например, в главе 15 рассмотрены 3-арные (или *троичные*) деревья, в которых используются структуры с тремя именованными связями (левой, средней и правой), каждая из которых имеет специальное значение для связанных с ними алгоритмов. В остальных случаях использование массивов для хранения связей вполне подходит, поскольку значение *M* фиксировано, хотя, как будет показано, при использовании такого представления особое внимание потребуется уделить интенсивному использованию памяти.

Определение 5.3 ***Дерево*** *(также называемое* ***упорядоченным деревом****) — это узел (называемый корнем), связанный с последовательностью несвязанных деревьев. Такая последовательность называется* ***бором****.*

Различие между упорядоченными деревьями и *M*-арными деревьями состоит в том, что узлы в упорядоченных деревьях могут иметь любое количество дочерних узлов, в то время как узлы в *M*-арных деревьях должны иметь точно *M* дочерних узлов. Иногда в контекстах, в которых требуется различать упорядоченные и *M*-арные деревья, мы используем термин *главное дерево (general tree)*.

Поскольку каждый узел в упорядоченном дереве может иметь любое количество связей, представляется естественным рассматривать его с использованием связного списка, а не массива, для хранения связей с дочерними узлами. Пример такого представления приведен на рис. 5.22. Из этого примера видно, что тогда каждый узел содержит две связи: одну для связного списка, соединяющего его с родственными узлами, и вторую для связного списка его дочерних узлов.

Лемма 5.4 *Существует однозначное соответствие между бинарными деревьями и упорядоченными борами.*

Это соответствие показано на рис. 5.22. Любой бор можно представить в виде бинарного дерева, сделав левую связь каждого узла указывающей на его левый дочерний узел, а правую связь каждого узла — указывающей на родственный узел, расположенный справа.

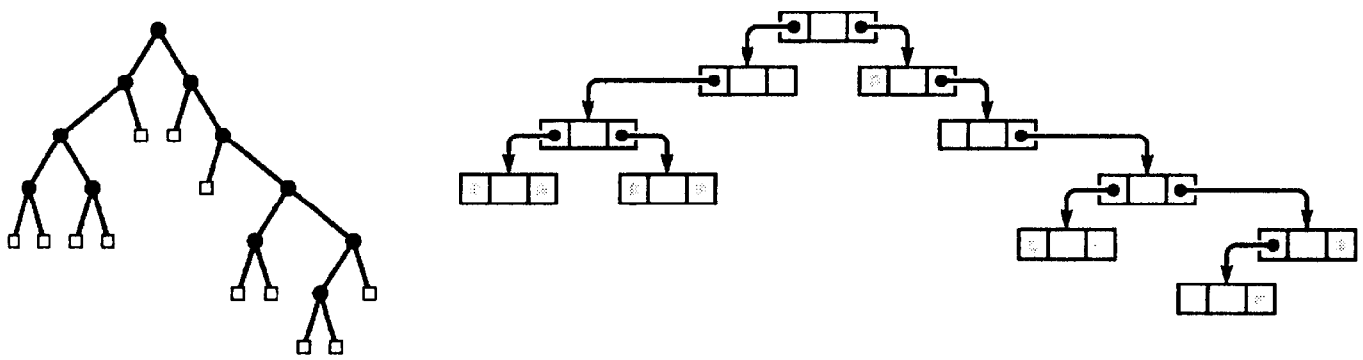


РИСУНОК 5.21 ПРЕДСТАВЛЕНИЕ БИНАРНОГО ДЕРЕВА

В стандартном представлении бинарного дерева используются узлы с двумя связями: левой связью с левым поддеревом и правой связью с правым поддеревом. Нулевые связи соответствуют внешним узлам.

Определение 5.4 ***Дерево с корнем*** *(или* ***неупорядоченное дерево****) — это узел (называемый корнем), связанный с множественным набором деревьев с корнем. (Такой множественный набор называется неупорядоченным бором.)*

Деревья, с которыми мы встречались в главе 1, посвященной проблеме связности, являются неупорядоченными деревьями. Такие деревья могут быть определены в качестве упорядоченных деревьев, в которых порядок рассмотрения дочерних узлов узла не имеет значения. Неупорядоченные деревья можно было бы также определить в виде набора взаимосвязей между родительскими и дочерними узлами. Такое представление может казаться не связанным с рассматриваемыми рекурсивными структурами, но, вероятно, является конкретным представлением, которое в основном соответствует абстрактному подходу.

Неупорядоченное дерево можно было бы представить в компьютере упорядоченным деревом; при этом приходится признать, что несколько различных упорядоченных деревьев могут представлять одно и то же неупорядоченное дерево. Действительно, обратная задача определения того, представляют ли два различные упорядоченные дерева одно и то же неупорядоченное дерево (задача *изоморфизма* дерева) трудно подается решению.

Наиболее общим типом деревьев является дерево, в котором не выделен ни один корневой узел. Например, остовные деревья, полученные в результате работы алгоритмов связности из главы 1, обладают этим свойством. Для правильного определения *деревьев без корня*, *неупорядоченных деревьев* и *свободных деревьев* потребуется начать с определения *графов (graphs).*

Определение 5.5 ***Граф*** *— это набор узлов с набором ребер, которые соединяют пары отдельных узлов (причем любую пару узлов соединяет только одно ребро).*

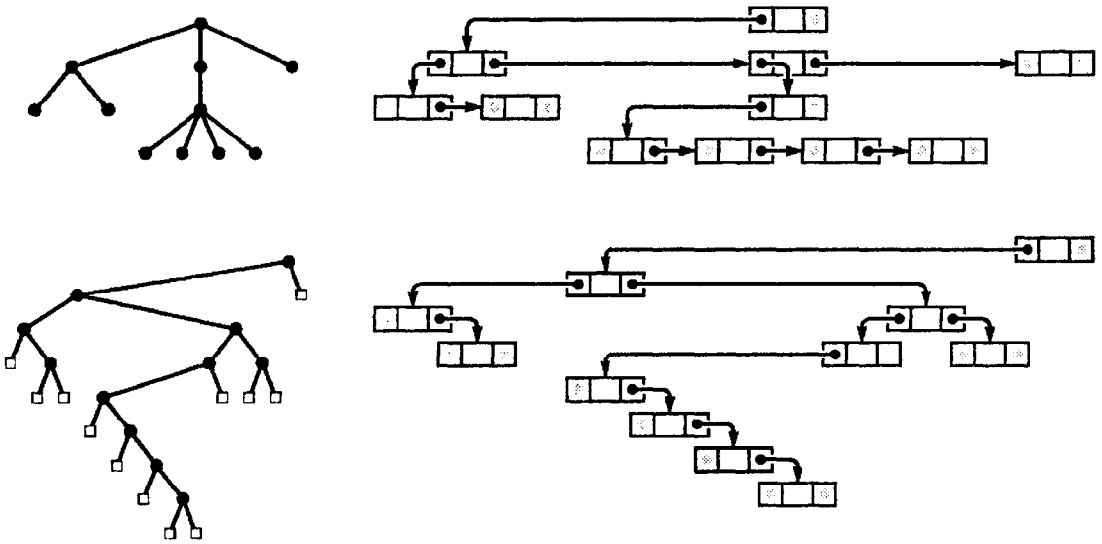


РИСУНОК 5.22 ПРЕДСТАВЛЕНИЕ ДЕРЕВА

Представление упорядоченного дерева за счет поддержания связного списка дочерних узлов каждого узла эквивалентно представлению его в виде двоичного дерева. На схеме справа вверху показано представление в виде связного списка дочерних узлов для дерева, показанного слева вверху; при этом список реализован в правых связях узлов, а левая связь каждого узла указывает на первый узел в связном списке его дочерних узлов. На схеме справа внизу приведена несколько измененная версия верхней схемы; она представляет бинарное дерево, изображенное слева внизу. Таким образом, бинарное дерево можно рассматривать в качестве представления дерева.

Представьте себе, что перемещаетесь вдоль ребра от одного какого-либо узла до другого, затем от этого узла к следующему и т.д. Последовательность ребер, ведущих от одного узла до другого, когда ни один узел не посещается дважды, называется *простым путем*. Граф является *связным (connected)*, если существует простой путь, связывающий любую пару узлов. Простой путь, у которого первый и последний узел совпадают, называется *циклом (cycle)*.

Каждое дерево является графом; а какие же графы являются деревьями? Граф считается деревом, если он удовлетворяет любому из следующих четырех условий:

- Граф имеет $N-1$ ребер и ни одного цикла.
- Граф имеет $N-1$ ребер и является связным.
- Только один простой путь соединяет каждую пару вершин в графе.
- Граф является связным, но перестает быть таковым при удалении любого ребра.

Любое из этих условий — необходимое и достаточное условие для выполнения остальных трех. Формально, одно из них должно было бы служить определением *свободного дерева*; неформально, они все вместе служат определением.

Мы представляем свободное дерево в виде коллекции ребер. Если представлять свободное дерево неупорядоченным, упорядоченным или даже бинарным деревом, придется признать, что в общем случае существует множество различных способов представления каждого свободного дерева.

Абстракция дерева используется часто, и рассмотренные в этом разделе различия важны, поскольку знание различных абстракций деревьев — существенная составляющая определения эффективного алгоритма и соответствующих структур данных для решения данной задачи. Часто приходится работать непосредственно с конкретными представлениями деревьев без учета конкретной абстракции, но в то же время часто имеет смысл поработать с соответствующей абстракцией дерева, а затем рассмотреть различные конкретные представления. В книге приведено множество примеров этого процесса.

Прежде чем вернуться к алгоритмам и представлениям, мы рассмотрим ряд основных математических свойств деревьев; эти свойства будут использоваться при разработке и анализе алгоритмов в виде деревьев.

Упражнения

▷ **5.56** Приведите представления свободного дерева, показанного на рис. 5.20, в форме дерева с корнем и бинарного дерева.

• **5.57** Сколько существует различных способов представления свободного дерева, показанного на рис. 5.20, в форме упорядоченного дерева?

▷ **5.58** Нарисуйте три упорядоченных дерева, которые изоморфны по отношению к упорядоченному дереву, показанному на рис. 5.20. Другими словами, должна существовать возможность преобразования всех четырех деревьев одного в другое путем обмена дочерними узлами.

○ **5.59** Предположите, что деревья содержат элементы, для которых определена операция **operator==**. Создайте рекурсивную программу, которая удаляет в бинарном дереве все листья, содержащие элементы, равные данному (см. программу 5.5).

○ **5.60** Измените функцию "разделяй и властвуй" для поиска максимального элемента в массиве (программа 5.6), чтобы она делила массив на k частей, размер которых различался бы не более чем на 1, рекурсивно находила максимум в каждой части и возвращала максимальный из максимумов.

5.61 Нарисуйте 3-арные и 4-арные деревья, соответствующие использованию $k = 3$ и $k = 4$ в рекурсивной конструкции, предложенной в упражнении 5.60, для массива, состоящего из 11 элементов (см. рис. 5.6).

○ **5.62** Бинарные деревья эквивалентны двоичным строкам, в которых нулевых битов на 1 больше, чем единичных при соблюдении дополнительного ограничения, что в любой позиции k количество нулевых битов слева от k не больше, чем количество единичных битов слева от k. Бинарное дерево — это либо нуль, либо две таких строки, объединенные вместе, которым предшествует 1. Нарисуйте бинарное дерево, соответствующее строке

```
1110010110001011000
```

○ **5.63** Упорядоченные деревья эквивалентны согласованным парам круглых скобок: упорядоченное дерево — это либо нуль, либо последовательность упорядоченных деревьев, заключенных в круглые скобки. Нарисуйте упорядоченное дерево, соответствующее строке

```
(((()(()())())(()()())))
```

●● **5.64** Создайте программу для определения того, представляют ли два массива N целочисленных значений между 0 и $N-1$ изоморфные неупорядоченные деревья, если интерпретируются (как в главе 1) в форме связей типа родительский-дочерний в дереве с узлами, пронумерованными от 0 до $N-1$. То есть, программа должна определять, существует ли способ изменения нумерации узлов в одном дереве, чтобы представление в виде массива одного дерева было идентичным представлению в виде массива другого дерева.

●● **5.65** Создайте программу для определения того, представляют ли два бинарных дерева изоморфные неупорядоченные деревья.

▷ **5.66** Нарисуйте все упорядоченные деревья, которые могли бы представлять дерево, определенное набором ребер **0-1, 1-2, 1-3, 1-4, 4-5**.

● **5.67** Докажите, что если в связанном графе, состоящем из N узлов, удаление любого ребра влечет за собой разъединение графа, то он имеет $N-1$ ребер и ни одного цикла.

5.5 Математические свойства бинарных деревьев

Прежде чем приступить к рассмотрению алгоритмов обработки деревьев, продолжим математическую тему, рассмотрев ряд базовых свойств деревьев. Мы сосредоточим внимание на бинарных деревьях, поскольку они используются в книге чаще других. Понимание их основных свойств послужит фундаментом для понимания характеристик производительности различных алгоритмов, с которыми мы встретимся — не только тех, в которых бинарные деревья используются в качестве явных структур данных, но и рекурсивных алгоритмов типа "разделяй и властвуй" и других аналогичных применений.

Лемма 5.5 *Бинарное дерево с N внутренними узлами имеет $N + 1$ внешних узлов.*

Эта лемма доказывается методом индукции: бинарное дерево без внутренних узлов имеет один внешний узел, следовательно, лемма справедлива для N = 0. Для $N > 0$ любое бинарное дерево с N внутренними узлами имеет k внутренних узлов в левом поддереве и $N-1-k$ внутренних узлов в правом поддереве *для* некоторого k в диапазоне между 0 и $N-1$, поскольку корень является внутренним узлом. В соответствии с индуктивным предположением левое поддерево имеет $k + 1$ внешних узлов, а правое поддерево — $N-k$ внешних узлов, что в сумме составляет $N + 1$.

Лемма 5.6 *Бинарное дерево с N внутренними узлами имеет $2N$ связей: $N-1$ связей с внутренними узлами и $N + 1$ связей с внешними узлами.*

В каждом дереве с корнем каждый узел, за исключением корня, имеет единственный родительский узел, и каждое ребро соединяет узел с его родительским узлом; следовательно, существует $N-1$ связей, соединяющих внутренние узлы. Аналогично, каждый из $N + 1$ внешних узлов имеет одну связь со своим единственным родительским узлом.

Характеристики производительности многих алгоритмов зависят не только от количества узлов в связанных с ними деревьях, но и от различных структурных свойств.

Определение 5.6 ***Уровень (level)*** *узла в дереве на единицу выше уровня его родительского узла (корень размещается на уровне 0).* ***Высота (height)*** *дерева — максимальный из уровней узлов дерева.* ***Длина пути (path length)*** *дерева — сумма уровней всех узлов дерева.* ***Длина внутреннего пути (internal path length)*** *бинарного дерева — сумма уровней всех внутренних узлов дерева.* ***Длина внешнего пути (external path length)*** *бинарного дерева — сумма уровней всех внешних узлов дерева.*

Удобный способ вычисления длины пути дерева заключается в суммировании произведений k на число узлов на уровне k для всех k.

Из этих соотношений следуют также простые рекурсивные определения, вытекающие непосредственно из рекурсивных определений деревьев и бинарных деревьев. Например, высота дерева на 1 больше максимальной высоты поддеревьев его корня, а длина пути дерева, имеющего N узлов равна сумме длин путей поддеревьев его корня плюс $N-1$. Приведенные соотношения также непосредственно связаны с анализом рекурсивных алгоритмов. Например, для многих рекурсивных вычислений высота соответствующего дерева в точности равна максимальной глубине рекурсии, или размеру стека, необходимого для поддержки вычисления.

Лемма 5.7 *Длина внешнего пути любого бинарного дерева, имеющего N внутренних узлов, на $2N$ больше длины внутреннего пути.*

Эту лемму можно было бы доказать методом индукции, но есть другое, более наглядное доказательство (которое применимо и для доказательства леммы 5.6). Обратите внимание, что любое бинарное дерево может быть сконструировано при помощи следующего процесса: начните с бинарного дерева, состоящего из одного внешнего узла. Затем повторите следующее N раз: выберите внешний узел и замените его новым внутренним узлом с двумя дочерними внешними узлами. Если выбранный внешний узел располагается на уровне k, длина внутреннего пути увеличивается на k, но длина внешнего пути увеличивается на $k + 2$ (один внешний

узел на уровне k удаляется, но два на уровне $k + 1$ добавляются). Процесс начинается с дерева, длина внутреннего и внешнего путей которого равны 0, и на каждом из N шагов длина внешнего пути увеличивается на 2 больше, чем длина внутреннего пути.

Лемма 5.8 *Высота бинарного дерева с N внутренними узлами не меньше* $\lg N$ *и не больше* $N - 1$.

Худший случай — вырожденное дерево, имеющее только один лист и $N - 1$ связь от корня до листа (см. рис. 5.23). В лучшем случае мы имеем уравновешенное дерево с 2^i внутренними узлами на каждом уровне i, за исключением самого нижнего (см. рис. 5.23). Если высота равна h, то должно быть справедливо соотношение

$$2^{h-1} < N + 1 \leq 2^h$$

поскольку существует $N + 1$ внешних узлов. Из этого неравенства следует провозглашенная лемма: в лучшем случае высота равна $\lg N$, округленному до ближайшего целого числа.

Лемма 5.9 *Длина внутреннего пути бинарного дерева с N внутренними узлами не меньше чем* $N \lg(N/4)$ *и не превышает* $N(N-1)/2$.

Худший и лучший случай соответствуют тем же деревьям, которые упоминались при рассмотрении леммы 5.8 и показаны на рис. 5.23. В худшем случае длина внутреннего пути дерева равна $0+1+2+\ldots+(N-1)=N(N-1)/2$. В лучшем случае дерево имеет $(N+1)$ внутренних узлов при высоте, не превышающей $\lfloor \lg N \rfloor$. Перемножая эти значения и применяя лемму 5.7, мы получаем предельное значение $(N+1)\lfloor \lg N \rfloor - 2N < N \lg(N/4)$.

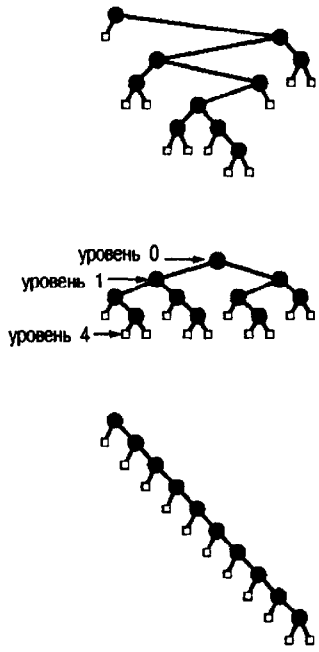


РИСУНОК 5.23 БИНАРНОЕ ДЕРЕВО С 10 ВНУТРЕННИМИ УЗЛАМИ

Бинарное дерево, показанное вверху, имеет высоту равную 7, длину внутреннего пути равную 31 и длину внешнего пути равную 51. Полностью уравновешенное бинарное дерево (в центре) с 10 внутренними узлами имеет высоту равную 4, длину внутреннего пути равную 19 и длину внешнего пути равную 39 (ни одно двоичное дерево с 10 узлами не может иметь меньшие значения любых из этих параметров). Вырожденное дерево (внизу) с 10 внутренними узлами имеет высоту равную 10, длину внутреннего пути равную 45 и длину внешнего пути равную 65 (ни одно бинарное дерево с 10 узлами не может иметь большие значения любых из этих параметров).

Как мы увидим, бинарные деревья часто используются в компьютерных приложениях, и при этом производительность наивысшая, когда бинарные деревья полностью уравновешены (или близки к этому). Например, деревья, которые использовались нами для описания алгоритмов "разделяй и властвуй", подобных бинарному поиску и сортировке слиянием, полностью уравновешены (см. упражнение 5.74). В главах 9 и 13 исследуются явные структуры данных, основывающиеся на уравновешенных деревьях.

Эти основные свойства деревьев предоставляют информацию, которая потребуется для разработки эффективных алгоритмов решения многих практических задач. Более подробный анализ нескольких специфичных алгоритмов, с которыми придется встретиться, требует сложного математического анализа, хотя часто полезные прибли-

женные оценки можно получить, прибегнув к прямолинейным индуктивным аргументам, подобным использованным в этом разделе. В последующих главах мы продолжим рассмотрение математических свойств деревьев по мере возникновения необходимости. Пока можно вернуться к теме алгоритмов.

Упражнения

▷ **5.68** Сколько внешних узлов существует в M-арном дереве с N внутренними узлами? Используйте свой ответ для определения объема памяти, необходимого для представления такого дерева, если считать, что для каждой связи и каждого элемента требуется по одному слову памяти.

5.69 Приведите верхнее и нижнее граничные значения высоты M-арного дерева с N внутренними узлами.

○ **5.70** Приведите верхнее и нижнее граничные значения длины внутреннего пути M-арного дерева с N внутренними узлами.

5.71 Приведите верхнее и нижнее граничные значения количества листьев в бинарном дереве с N узлами.

● **5.72** Покажите, что если листья внешних узлов в бинарном дереве различаются на константу, то высота составляет $O(\log N)$.

○ **5.73** *Дерево Фибоначчи* высотой $n > 2$ — это бинарное дерево с деревом Фибоначчи высотой $n - 1$ в одном поддереве и дерево Фибоначчи высотой $n - 2$ — в другом. Дерево Фибоначчи высотой 0 — это единственный внешний узел, а дерево Фибоначчи высотой 1 — единственный внутренний узел с двумя внешними дочерними узлами (см. рис. 5.14). Выразите высоту и длину внешнего пути дерева Фибоначчи высотой n в виде функции N (количества узлов в дереве).

5.74 *Дерево типа "разделяй и властвуй"*, состоящее из N узлов, — это бинарное дерево с корнем, обозначенным N, деревом типа "разделяй и властвуй", состоящим из $\lfloor N/2 \rfloor$ узлов, в одном поддереве и деревом типа "разделяй и властвуй", состоящим из $\lceil N/2 \rceil$ узлов, в другом. (Дерево типа "разделяй и властвуй" показано на рис. 5.6.) Нарисуйте дерево типа "разделяй и властвуй" с 11, 15, 16 и 23 узлами.

○ **5.75** Докажите методом индукции, что длина внутреннего пути дерева типа "разделяй и властвуй" находится в пределах между $N \lg N$ и $N \lg N + N$.

5.76 *Дерево типа "объединяй и властвуй"*, состоящее из N узлов, — это бинарное дерево с корнем, обозначенным N, деревом типа "объединяй и властвуй", состоящим из $\lfloor N/2 \rfloor$ узлов, в одном поддереве и деревом типа "объединяй и властвуй", состоящим из $\lceil N/2 \rceil$ узлов, в другом (см. упражнение 5.18). Нарисуйте дерево типа "объединяй и властвуй" с 11, 15, 16 и 23 узлами.

5.77 Докажите методом индукции, что длина внутреннего пути дерева типа "объединяй и властвуй" находится в пределах между $N \lg N$ и $N \lg N + N$.

5.78 *Полное (complete)* бинарное дерево — это дерево, в котором все уровни, кроме, возможно, последнего, который заполняется слева направо, заполнены, как проиллюстрировано на рис. 5.24. Докажите, что длина внутреннего пути полного дерева с N узлами лежит в пределах между $N \lg N$ и $N \lg N + N$.

5.6 Обход дерева

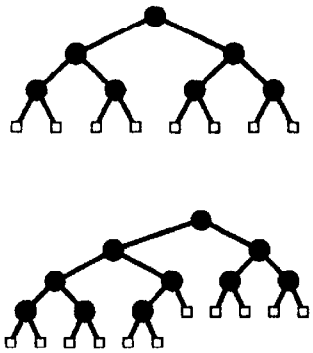


РИСУНОК 5.24 ПОЛНЫЕ БИНАРНЫЕ ДЕРЕВЬЯ С СЕМЬЮ И ДЕСЯТЬЮ ВНУТРЕННИМИ УЗЛАМИ

Когда количество внешних узлов является степенью 2 (верхний рисунок), все внешние узлы в полном бинарном дереве располагаются на одном уровне. В противном случае (нижний рисунок) внешние узлы располагаются в двух уровнях при размещении внутренних узлов слева от внешних узлов предпоследнего уровня.

Прежде чем рассматривать алгоритмы, конструирующие бинарные деревья и деревья, рассмотрим алгоритмы для реализации наиболее общей функции обработки деревьев — *обхода дерева*; при наличии указателя на дерево требуется систематически обработать все узлы в дереве. В связном списке переход от одного узла к другому выполняется за счет отслеживания единственной связи; однако, в случае деревьев приходится принимать решения, поскольку может существовать несколько связей, по которым можно следовать.

Начнем рассмотрение с процесса для бинарных деревьев. В случае со связными списками имелись две основные возможности (см. программу 5.5): обработать узел, а затем следовать связи (в этом случае узлы посещались бы по порядку), или следовать связи, а затем обработать узел (в этом случае узлы посещались бы в обратном порядке). При работе с бинарными деревьями существуют две связи и, следовательно, три основных порядка возможного посещения узлов:

- *Прямой обход (сверху вниз)*, при котором мы посещаем узел, а затем левое и правое поддеревья
- *Поперечный обход (слева направо)*, при котором мы посещаем левое поддерево, затем узел, а затем правое поддерево
- *Обратный обход (снизу вверх)*, при котором мы посещаем левое и правое поддеревья, а затем узел.

Эти методы можно легко реализовать с помощью рекурсивной программы, как показано в программе 5.14, которая является непосредственным обобщением программы 5.5 обхода связного списка. Для реализации обходов в других порядках достаточно соответствующим образом переставить вызовы функций в программе 5.14. Порядок посещения узлов в примере дерева при использовании каждого из порядков обхода показан на рис. 5.26. На рис. 5.25 приведена последовательность вызовов функций, которые выполняются при вызове программы 5.14 применительно к примеру дерева из рис. 5.26.

Программа 5.14 Рекурсивный обход дерева

Эта рекурсивная функция принимает в качестве аргумента ссылку на дерево и вызывает функцию **visit** для каждого из узлов дерева. В приведенном виде функция реализует прямой обход; если поместить обращение к **visit** между рекурсивными вызовами, получится поперечный обход; а если поместить обращение к **visit** после рекурсивных вызовов — то обратный обход.

```
void traverse(link h, void visit(link))
   {
      if (h == 0) return;
      visit(h);
      traverse(h->l, visit);
      traverse(h->r, visit);
   }
```

С этими же основными рекурсивными процессами, на которых основываются различные методы обхода дерева, мы уже встречались в рекурсивных программах типа "разделяй и властвуй" (см. рис. 5.8 и 5.11) и в арифметических выражениях. Например, выполнение прямого обхода соответствует рисованию вначале меток на линейке, а затем выполнению рекурсивных вызовов (см. рис. 5.11); выполнение поперечного обхода соответствует перемещению самого большого диска в решении задачи о ханойских башнях между рекурсивными вызовами, которые перемещают все остальные диски; выполнение обратного обхода соответствует вычислению постфиксных выражений и т.д. Эти соответствия позволяют немедленно заглянуть внутрь механизмов, лежащих в основе обхода дерева. Например, известно, что при поперечном обходе каждый второй узел является внешним, по той же причине, по какой при решении задачи о ханойских башнях каждое второе перемещение затрагивает маленький диск.

Полезно также рассмотреть нерекурсивные реализации, в которых используется явный стек. Для простоты мы начнем с рассмотрения абстрактного стека, содержащего элементы дерева, инициализированного деревом, которое требуется обойти. Затем мы войдем в цикл, в котором выталкивается и обрабатывается верхняя запись стека, и который продолжается, пока стек не опустеет. Если вытолкнутая запись является элементом, мы посещаем его; если вытолкнутая запись — дерево, мы выполняем последовательность операций выталкивания, которая зависит от требуемого порядка:

- Для *прямого обхода* заталкивается правое поддерево, затем левое поддерево, а затем узел.
- Для *поперечного обхода* заталкивается правое поддерево, затем узел, а затем левое поддерево.
- Для *обратного обхода* заталкивается узел, затем правое поддерево, а затем левое поддерево.

Нулевые деревья в стек не заталкиваются. На рис. 5.27 показано содержимое стека при использовании каждого из этих трех методов для обхода примера дерева, приведенного на рис. 5.26. Методом индукции можно легко убедиться, что для любого бинарного дерева этот метод приводит к такому же результату, как и рекурсивный метод.

```
traverse E
  visit E
  traverse D
    visit D
    traverse B
      visit B
      traverse A
        visit A
        traverse *
        traverse *
      traverse C
        visit C
        traverse *
        traverse *
    traverse *
  traverse H
    visit H
    traverse F
      visit F
      traverse *
      traverse G
        visit G
        traverse *
        traverse *
    traverse *
```

РИСУНОК 5.25 ВЫЗОВЫ ФУНКЦИЙ ПРЯМОГО ОБХОДА

Эта последовательность вызовов функций определяет прямой обход для примера дерева, показанного на рис. 5.26.

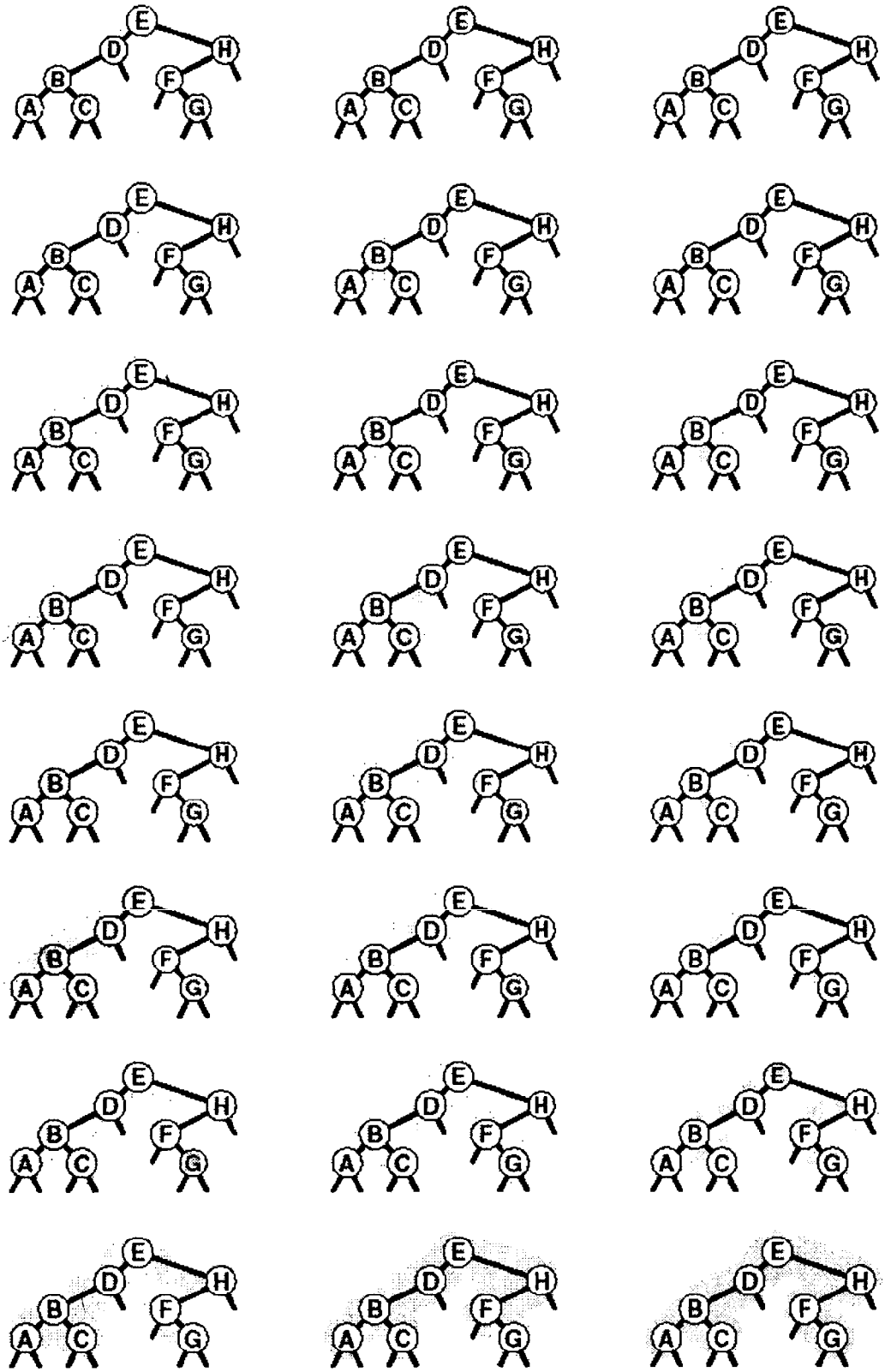
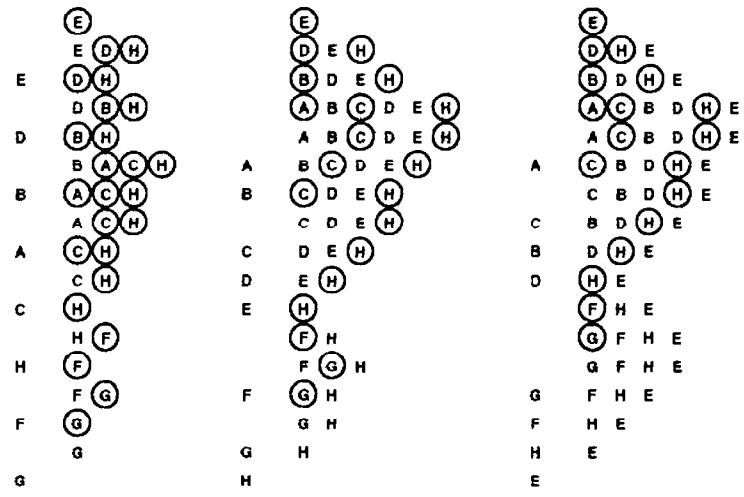


РИСУНОК 5.26 ПОРЯДКИ ОБХОДА ДЕРЕВА

Эти последовательности показывают порядок посещения узлов для прямого (слева), поперечного (в центре) и обратного (справа) обхода дерева.

РИСУНОК 5.27 СОДЕРЖИМОЕ СТЕКА ДЛЯ АЛГОРИТМОВ ОБХОДА ДЕРЕВА

Эти последовательности отражают содержимое стека для прямого (слева), поперечного (в центре) и обратного (справа) обхода дерева (см. рис. 5.26) для идеализированной модели вычислений, аналогичной использованной в примере на рис. 5.5, когда элемент и два его поддерева помещаются в стек в указанном порядке.



Описанная в предыдущем абзаце схема является концептуальной, охватывающей три метода обхода дерева, однако реализации, используемые на практике, несколько проще. Например, для выполнения прямого обхода не обязательно заталкивать узлы в стек (мы посещаем корень каждого выталкиваемого дерева), поэтому можно воспользоваться простым стеком, состоящим только из одного типа элементов (связей дерева), как это сделано в нерекурсивной реализации в программе 5.15. Системный стек, поддерживающий рекурсивную программу, содержит адреса возврата и значения аргументов, а не элементы или узлы, но фактическая последовательность выполнения вычислений (посещения узлов) остается одинаковой для рекурсивного метода и метода с использованием стека.

Программа 5.15 Прямой обход (нерекурсивная реализация)

Эта нерекурсивная функция с использованием стека — функциональный эквивалент ее рекурсивного аналога, программы 5.14.

```
void traverse(link h, void visit(link))
  { STACK<link> s(max);
    s.push(h);
    while (!s.empty())
      {
          visit(h = s.pop());
          if (h->r != 0) s.push(h->r);
          if (h->l != 0) s.push(h->l);
      }
  }
```

Четвертая естественная стратегия обхода — простое посещение узлов дерева в порядке, в котором они отображаются на странице — сверху вниз и слева направо. Этот метод называется обходом *по уровням*, поскольку все узлы каждого уровня посещаются вместе, по порядку. Посещение узлов дерева, показанного на рис. 5.26, при обходе по уровням показано на рис. 5.28.

Как ни удивительно, обход по уровням можно получить, заменив в программе 5.15 стек очередью, что демонстрирует программа 5.16. Для реализации прямого обхода используется структура данных типа "последним вошел, первым вышел" (LIFO); для реализации обхода по уровням используется структура данных типа "первым вошел, первым вышел" (FIFO). Эти программы заслуживают внимательного изучения,

поскольку они представляют существенно различающиеся подходы к организации остающейся невыполненной работы. В частности, обход по уровням не соответствует рекурсивной реализации, связанной с рекурсивной структурой дерева.

Программа 5.16 Обход по уровням

Изменение структуры данных, лежащей в основе прямого обхода (см. программу 5.15) со стека на очередь приводит к преобразованию обхода в обход по уровням.

```
void traverse(link h, void visit(link))
 { QUEUE<link> q(max);
   q.put(h);
   while (!q.empty())
    {
        visit(h = q.get());
        if (h->l != 0) q.put(h->l);
        if (h->r != 0) q.put(h->r);
    }
 }
```

Прямой обход, обратный обход и обход по уровням однозначно определяются и для боров. Чтобы определения были единообразными, представьте себе бор в виде дерева с воображаемым корнем. Тогда правило для прямого обхода формулируется следующим образом: "посетить корень, а затем каждое из поддеревьев"; а правило для обратного обхода — "посетить каждое из поддеревьев, а затем корень". Правило для обхода по уровням то же, что и для бинарных деревьев. Непосредственные реализации этих методов — простые обобщения программ прямого обхода с использованием стека (программы 5.14 и 5.15) и программы обхода по уровням с использованием очереди (программа 5.16) для бинарных деревьев, которые мы только что рассмотрели. Соображения по поводу реализаций не приводятся, поскольку в разделе 5.8 мы рассмотрели более общую процедуру.

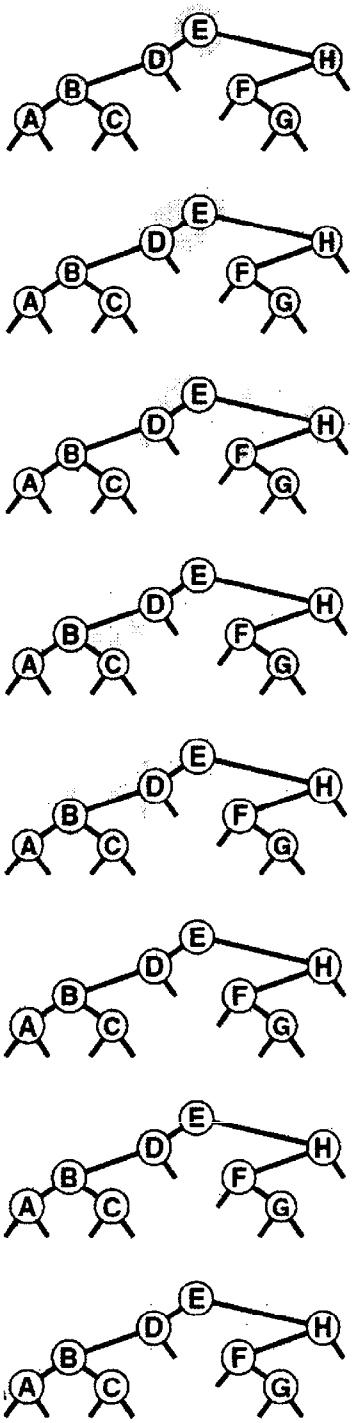
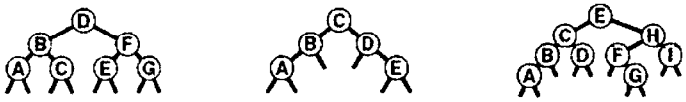


РИСУНОК 5.28 ОБХОД ПО УРОВНЯМ

Эта последовательность отображает результат посещения узлов дерева в порядке сверху вниз и слева направо.

Упражнения

▷ **5.79** Приведите порядок посещения узлов для прямого, поперечного, обратного и обхода по уровням для следующих бинарных деревьев:



▷ **5.80** Отразите содержимое очереди во время обхода по уровням (программа 5.16) на рис. 5.28, аналогично показанному на рис. 5.27.

5.81 Покажите, что прямой обход бора равноценен прямому обходу соответствующего бинарного дерева (см. лемму 5.4), а обратный обход бора совпадает с поперечным обходом бинарного дерева.

○ **5.82** Приведите нерекурсивную реализацию поперечного обхода.

● **5.83** Приведите нерекурсивную реализацию обратного обхода.

● **5.84** Создайте программу, которая преобразует прямой и поперечный обходы бинарного дерева в обход дерева по уровням.

5.7 Рекурсивные алгоритмы бинарных деревьев

Алгоритмы обхода дерева, рассмотренные в разделе 5.6, наглядно демонстрируют необходимость рассмотрения рекурсивных алгоритмов бинарных деревьев, что обусловлено самой рекурсивной структурой этих деревьев. Для решения многих задач можно непосредственно применять рекурсивные алгоритмы типа "разделяй и властвуй", которые, по существу, обобщают алгоритмы обхода деревьев. Обработка дерева выполняется посредством обработки корневого узла и (рекурсивно) его поддеревьев; вычисление можно выполнять перед, между или после рекурсивных вызовов (или же использовать все три метода).

Часто требуется определять различные структурные параметры дерева, имея только ссылку на дерево. Например, программа 5.17 содержит рекурсивные функции для вычисления количества узлов и высоту заданного дерева. Функции определяются непосредственно исходя из определения 5.6. Ни одна из этих функций не зависит от порядка обработки рекурсивных вызовов: они обрабатывают все узлы дерева и возвращают одинаковый результат, если, например, рекурсивные вызовы поменять местами. Не все параметры дерева вычисляются так легко: например, программа для эффективного вычисления длины внутреннего пути бинарного дерева является более сложной (см. упражнения 5.88—5.90).

Программа 5.17 Вычисление параметров дерева

Для выяснения базовых структурных свойств дерева можно использовать простые рекурсивные процедуры, подобные следующим.

```
int count(link h)
  {
    if (h == 0) return 0;
    return count(h->l) + count(h->r) + 1;
  }
int height(link h)
  {
    if (h == 0) return -1;
    int u = height(h->l), v = height(h->r);
    if (u > v) return u+1; else return v+1;
  }
```

Еще одна функция, которая полезна при создании программ, обрабатывающих деревья, — функция, которая выводит на печать структуру или рисует дерево. Например, программа 5.18 — рекурсивная процедура, выводящая дерево в формате, приведенном на рис. 5.29. Эту же базовую рекурсивную схему можно использовать для

рисования более сложных представлений деревьев, подобным использованным на рисунках в этой книге (см. упражнение 5.85).

Программа 5.18 Функция быстрого вывода дерева

Эта рекурсивная программа отслеживает высоту дерева и использует эту информацию для вывода его представления, которое можно использовать при отладке программ обработки деревьев (см. рис. 5.29). В программе предполагается, что элементы в узлах имеют тип **Item**, для которого определена перегруженная операция <<.

```
void printnode(Item x, int h)
  { for (int i = 0; i < h; i++) cout << " ";
    cout << x << endl;
  }
void show(link t, int h)
  {
    if (t == 0) { printnode('*', h); return;
    }
    show(t->r, h+1);
    printnode(t->item, h);
    show(t->l, h+1);
  }
```

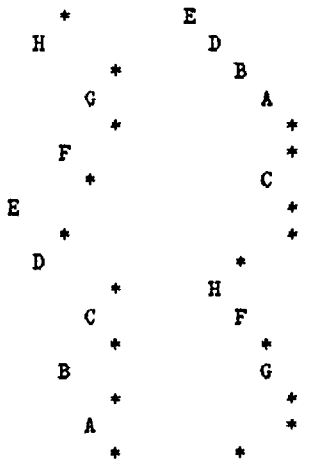


РИСУНОК 5.29 ВЫВОД ДЕРЕВА (ПРИ ПОПЕРЕЧНОМ ОБХОДЕ И ПРЯМОМ ОБХОДЕ)

Вывод, приведенный на рисунке слева, получен в результате использования программы 5.18 применительно к примеру дерева, приведенному на рис. 5.26, и он демонстрирует структуру дерева аналогично использованному графическому представлению, повернутому на 90 градусов. Вывод, показанный на рисунке справа, получен в результате выполнения этой же программы при перемещении оператора вывода в начало программы; он демонстрирует структуру дерева в привычном формате.

Программа 5.18 выполняет поперечный обход, но если выводить элемент перед рекурсивными вызовами, получаем прямой обход; этот вариант также приведен на рис. 5.29. Этот формат привычен, например, при отображении генеалогического дерева, списка файлов в файловой структуре в виде дерева или при создании структуры печатного документа. Например, выполнение прямого обхода дерева, отображенного на рис. 5.19, приводит к выводу варианта таблицы оглавления этой книги.

Вначале рассмотрим пример программы, которая строит явную структуру дерева, связанного с приложением определения максимума, которая была рассмотрена в разделе 5.2. Наша цель — построение *турнира* — бинарного дерева, в котором элементом каждого внутреннего узла является копия большего из элементов его двух дочерних элементов. В частности, элемент в корне — копия наибольшего элемента в турнире. Элементы в листья (узлах, которые не имеют дочерних узлов) образуют представляющие интерес данные, а остальная часть дерева — структура данных, которая позволяет эффективно находить наибольший из элементов.

Программа 5.19 — рекурсивная программа, которая строит турнир из элементов массива. Будучи расширенной версией программы 5.6, она использует стратегию "разделяй и властвуй": чтобы построить турнир для единственного элемента, программа создает лист, содержащий этот элемент, и выполняет возврат. Чтобы построить турнир для $N > 1$ элементов, в программе используется стратегия "разделяй и властвуй": программа делит все множество элементов пополам, строит турнир для каждой половины и создает новый узел со связями с двумя турнирами и с элементом, который является копией большего элемента в корнях обоих турниров.

Программа 5.19 Конструирование турнира

Эта рекурсивная функция делит массив **a[1], ..., a[r]** на две части **a[1], ..., a[m]** и **a[m+1], ..., a[r]**, строит турниры для двух частей (рекурсивно) и создает турнир для всего массива, связывая новый узел с рекурсивно построенными турнирами и помещая в него копию большего элемента в корнях двух рекурсивно построенных турниров.

```
struct node
 { Item item; node *l, *r;
   node(Item x)
     { item = x; l = 0; r = 0; }
 };
typedef node* link;
link max(Item a[], int l, int r)
 { int m = (l+r)/2;
   link x = new node(a[m]);
   if (l == r) return x;
   x->l = max(a, l, m);
   x->r = max(a, m+1, r);
   Item u = x->l->item, v = x->r->item;
   if (u > v)
     x->item = u; else x->item = v;
   return x;
 }
```

На рис. 5.30 приведен пример явной структуры дерева, построенной программой 5.19. Построение таких рекурсивных структур — вероятно, предпочтительней отысканию максимума путем просмотра данных, как это было сделано в программе 5.6, поскольку структура дерева обеспечивает определенную гибкость для выполнения других операций. Важным примером служит и сама операция, использованная для построения турнира. При наличии двух турниров их можно объединить в один турнир, создав новый узел, левая связь которого указывает на один турнир, а правая на другой, и приняв больший из двух элементов (помещенных в корнях двух данных турниров) в качестве наибольшего элемента объединенного турнира. Можно также рассмотреть алгоритмы добавления и удаления элементов и выполнения других операций. Здесь мы не станем рассматривать такие операции, поскольку аналогичные структуры данных, обладающие подобной гибкостью, исследуются в главе 9.

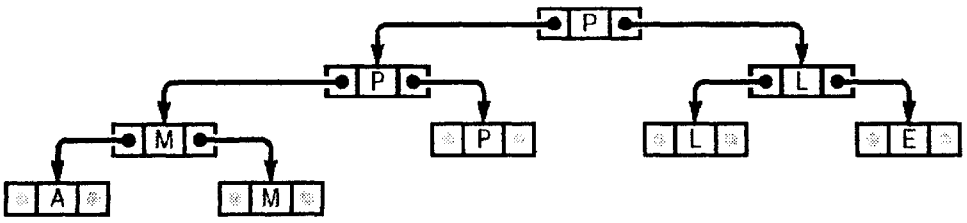


РИСУНОК 5.30 ЯВНОЕ ДЕРЕВО ДЛЯ ОТЫСКАНИЯ МАКСИМУМА (ТУРНИР)

На этом рисунке отображена структура дерева, сконструированная программой 5.19 на основании ввода элементов A M P L E. Элементы данных помещаются в листьях. Каждый внутренний узел содержит копию большего из элементов в двух дочерних узлах, следовательно, в соответствии с методом индукции наибольшим элементом является корень.