7 Деревья

В этой главе

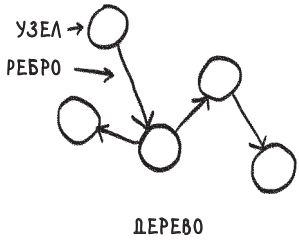
- ✓ Вы узнаете, что такое дерево и чем деревья отличаются от графов.
- ✓ Вы освоите выполнение алгоритмов с деревьями.
- ✓ Вы изучите поиск в глубину и узнаете, чем он отличается от поиска в ширину.
- ✓ Вы познакомитесь с кодом Хаффмана — алгоритмом сжатия, использующим деревья.

Что общего у алгоритмов сжатия и хранения информации в базе данных? В их основе часто лежит дерево, выполняющее всю черную работу. Деревья составляют подмножество графов. Деревья стоит рассмотреть отдельно, так как у них существует много специализированных разновидностей. Например, в коде Хаффмана — алгоритме сжатия, с которым вы познакомитесь в этой главе, — применяются бинарные деревья.

Многие базы данных используют сбалансированное дерево (например, B-дерево, о котором

вы узнаете в следующей главе). Видов деревьев довольно много. Чтобы вам было легче разобраться в них, в главах 7 и 8 будут представлены необходимая терминология и концепции.

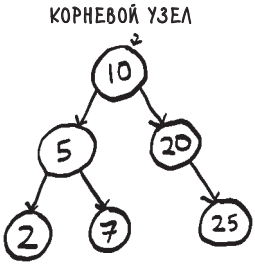
Ваше первое дерево



ДЕРЕВО

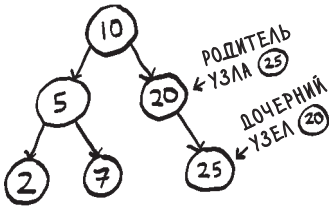
Дерево — это разновидность графа. Более подробное определение дадим ниже, а пока узнаем несколько терминов и рассмотрим пример.

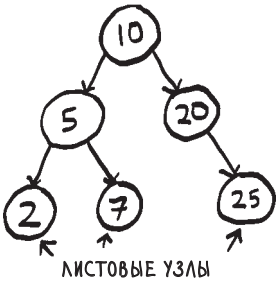
Как и графы, деревья состоят из узлов и ребер.



В этой книге мы будем работать с корневыми деревьями. У корневого дерева имеется один узел, от которого можно перейти к любому другому узлу.

Мы будем работать исключительно с корневыми деревьями, так что, говоря о *дереве* в этой главе, я всегда имею в виду корневое дерево. У узлов могут быть дочерние узлы, а у дочерних узлов может быть родительский узел.





В дереве узлы имеют по крайней мере одного родителя.

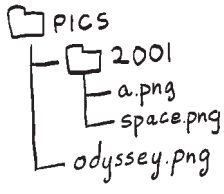
Существует только один узел без родителя — это корневой узел.

Узлы, не имеющие дочерних узлов, называются листовыми узлами (листьями).

Если вы поняли, что такое корень, лист, родительский и дочерний узел, значит, вы готовы двигаться дальше!

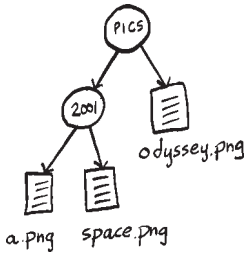
Каталоги файлов

Так как дерево является разновидностью графа, к нему можно применить алгоритм, работающий с графом. В главе 6 был рассмотрен поиск в ширину — алгоритм для нахождения кратчайшего пути в графе. Поиск в ширину можно использовать и с деревом. Если вы еще не освоились с поиском в ширину, обращайтесь к главе 6.



Каталог файлов представляет собой дерево, с которым каждый, кто работает с компьютером, сталкивается ежедневно. Представьте, что у нас есть такой каталог.

Требуется вывести имена всех файлов в каталоге pics, включая все его подкаталоги. Однако в данном случае существует только один подкаталог — 2001. Задачу можно решить поиском в ширину! Для начала я покажу, как этот каталог выглядит в виде дерева.



Ранее мы использовали алгоритм поиска в ширину как инструмент поиска, однако этим его возможности не ограничиваются. Поиск в ширину является алгоритмом обхода; это значит, что он посещает каждый узел дерева («обходит» его). А это именно то, что нужно! Нам нужен алгоритм, который перейдет к каждому файлу в дереве и выведет его имя. Для перебора файлов в каталоге можно воспользоваться поиском в ширину. Алгоритм также заходит в подкаталоги, ищет в них файлы и выводит имена обнаруженных файлов. Логика выглядит так:

1. Посетить каждый узел в дереве.
2. Если узел является файлом, вывести его имя.
3. Если узел является папкой, добавить его в очередь папок, чтобы найти находящиеся в нем файлы.

Ниже приведен код реализации. Он очень похож на код поиска продавца манго из главы 6:

```
from os import listdir
from os.path import isfile, join
from collections import deque

def printnames(start_dir):
  search_queue = deque()
  search_queue.append(start_dir)
  while search_queue:
    dir = search_queue.popleft()
    for file in sorted(listdir(dir)):
      fullpath = join(dir, file)
      if isfile(fullpath):
        print(file)
      else:
        search_queue.append(fullpath)
printnames("pics")
```

Очередь используется для отслеживания папок, в которых должен проводиться поиск

Пока очередь не пуста, извлечь из нее папку для проверки

Перебрать все файлы и папки в этой папке

Если это файл, вывести его имя

Если это папка, добавить ее в конец очереди папок для поиска

Здесь используется очередь, как и в примере с продавцом манго. В очереди хранится информация о том, в каких еще папках нужно провести поиск файлов. Конечно, в том примере мы останавливались сразу же после нахождения продавца манго, но здесь пройдем по всему дереву.

Впрочем, в этом примере присутствует одно важное отличие от кода поиска продавца манго. Удастся ли вам его найти?

Как вы помните, в примере с продавцом манго нам приходилось следить, проводился ли уже поиск этого человека:

```
...
    if person not in searched:
      if person_is_seller(person):
...
```

Искать этого человека, только если его поиск еще не проводился

Здесь этого делать не нужно! В деревьях нет циклов, и у каждого узла только один родитель. Мы никогда не сможем случайно провести поиск в одной папке несколько раз либо ввести программу в бесконечный цикл, поэтому нет необходимости отслеживать, в каких папках мы уже искали. Зайти снова в одну и ту же папку при обходе попросту невозможно.

Благодаря этому свойству деревьев наш код стал проще. Это важный вывод, который стоит запомнить: в деревьях не бывает циклов.

О СИМВОЛИЧЕСКИХ ССЫЛКАХ

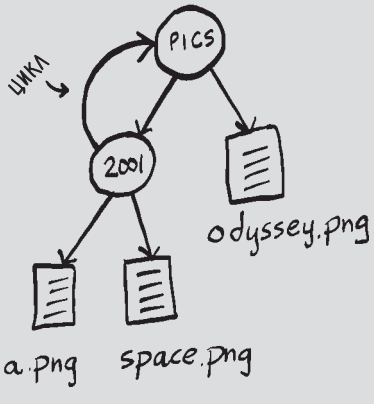
Вероятно, вы знаете, что такое символические ссылки. Для тех, кто не знает: символические ссылки открывают возможность создания циклов в каталоге файлов. Символическую ссылку в macOS или Linux можно создать командой

```
ln -s pics/ pics/2001/pics
```

а в Windows — командой

```
mklink /d pics/ pics/2001/pics
```

Если бы я добавил символическую ссылку, дерево выглядело бы примерно так:



Теперь каталог уже не является деревом! Чтобы не переусложнять, в этом примере мы не будем рассматривать символические ссылки. Если у вас они есть, Python достаточно сообразителен, чтобы не войти в бесконечный цикл. Он выдаст такую ошибку:

```
OSError: [Errno 62] Too many levels of symbolic links[1]:
'pics/2001/pics'
```

[1] Слишком много уровней символических ссылок. — *Примеч. пер.*

Космическая одиссея: поиск в глубину

Еще раз обойдем каталог файлов, но на этот раз рекурсивно:

```
from os import listdir
from os.path import isfile, join

def printnames(dir):
  for file in sorted(listdir(dir)):
    fullpath = join(dir, file)
    if isfile(fullpath):
      print(file)
    else:
      printnames(fullpath)

printnames("pics")
```

Перебрать все файлы и все папки в текущей папке

Если это файл, вывести его имя

Если это папка, вызвать для нее функцию рекурсивно, чтобы провести поиск файлов и папок

Обратите внимание: на этот раз очередь не используется. Вместо этого при обнаружении папки мы сразу заходим в нее, чтобы найти новые файлы и папки. В нашем распоряжении появились два способа вывода имен файлов. Удивительно, но *они выводят имена файлов в разном порядке*!

Одно решение выводит имена так:

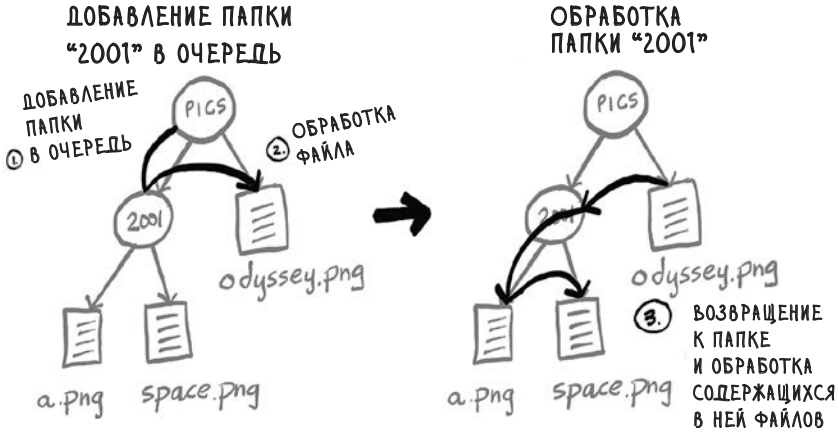
```
a.png
space.png
odyssey.png
```

А другое — так:

```
odyssey.png
a.png
space.png
```

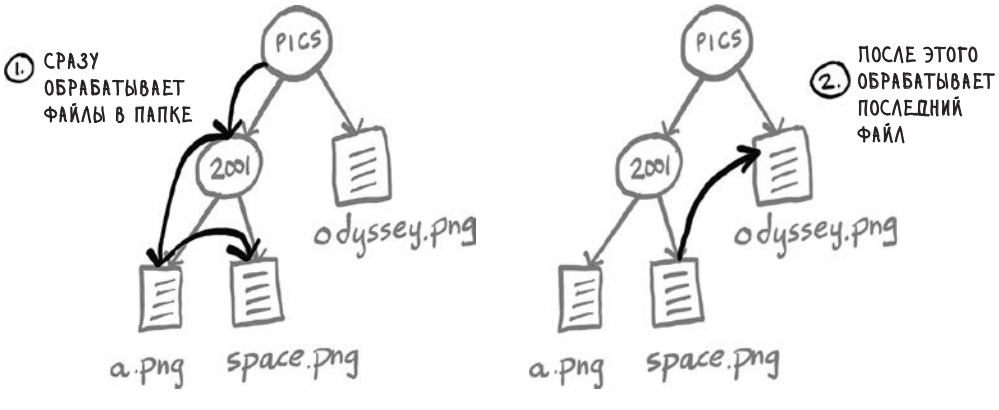
Можете ли вы определить, какое решение использует тот или иной порядок и почему? Проверьте себя, прежде чем двигаться дальше.

В первом решении используется поиск в ширину. Когда он находит папку, эта папка добавляется в очередь для проверки в будущем. Таким образом, алгоритм переходит к папке `2001`, не заходит в нее, но добавляет в очередь для последующей проверки. Он выводит все имена файлов в папке `pics/`, после чего переходит к папке `2001/` и выводит имена содержащихся в ней файлов.



Как видите, алгоритм сначала посещает папку 2001, но не заходит в нее. Эта папка просто добавляется в очередь, а поиск в ширину переходит к odyssey.png.

Во втором решении используется алгоритм, называемый поиском в глубину. Поиск в глубину тоже работает с графом и представляет собой алгоритм обхода дерева. При обнаружении папки он сразу заходит в нее, вместо того чтобы добавлять ее в очередь.

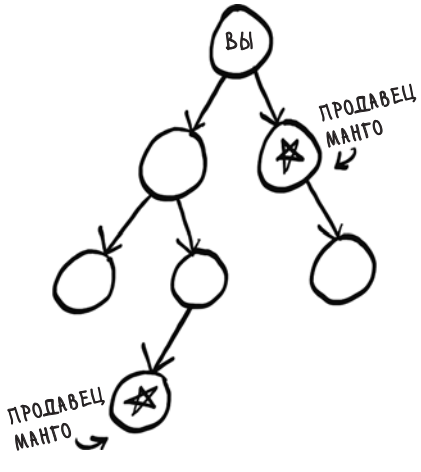


Второе решение выводит следующий результат:

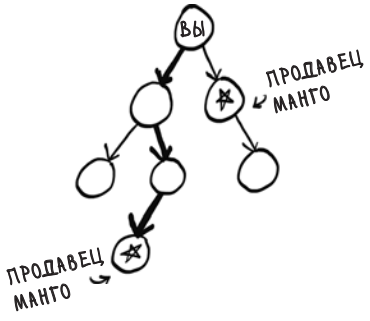
```
a.png
space.png
odyssey.png
```

Поиск в ширину и поиск в глубину тесно связаны друг с другом, и там, где речь идет об одном из них, будет упоминаться и другой. Оба алгоритма выводят все имена файлов, так что они оба подойдут для нашего примера. Однако между ними существует важное различие. Поиск в глубину не может использоваться для нахождения кратчайшего пути!

В примере с продавцом манго мы не могли применить поиск в глубину. Мы опирались на тот факт, что все друзья первого уровня проверяются раньше друзей второго уровня и т. д. Именно так работает поиск в ширину. В отличие от него, поиск в глубину сразу заходит на максимально возможную глубину. Он может сначала обнаружить продавца манго в трех уровнях от вас, хотя есть контакт ближе. Представим, что ваше дерево друзей выглядит так:

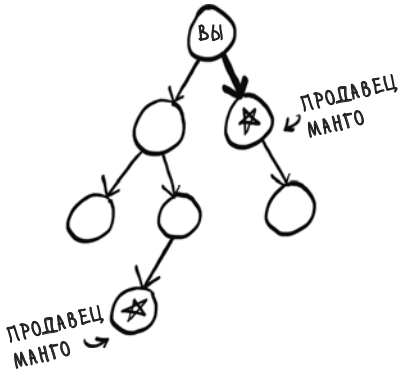


Предположим, что узлы обрабатываются слева направо. Поиск в глубину переходит к крайнему левому дочернему узлу, после чего продолжает работу.



Так как поиск зашел в глубину от левого узла, он не смог понять, что правый узел — продавец манго, находящийся намного ближе к вам.

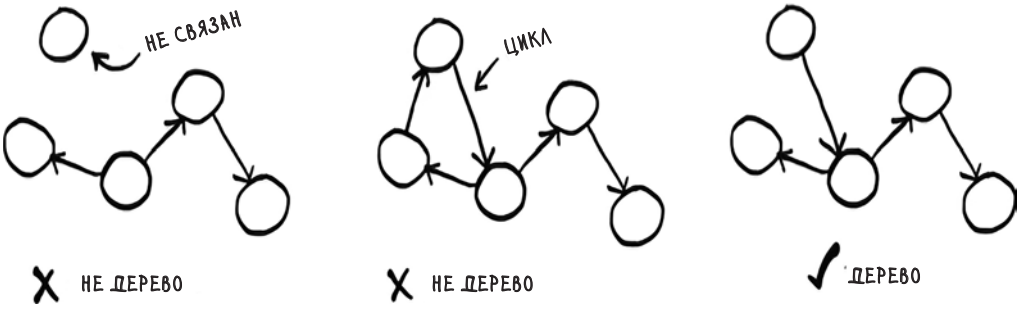
Поиск в ширину верно находит ближайшего продавца манго.



Таким образом, хотя оба алгоритма выводят списки файлов, для нахождения кратчайшего пути подходит только поиск в ширину. У поиска в глубину находятся другие применения. Его можно использовать для топологической сортировки — мы кратко рассмотрели ее в главе 6.

Правильное определение дерева

После рассмотрения примера можно привести более точное определение дерева. Дерево представляет собой *связный ациклический граф*.



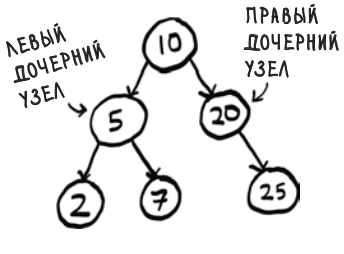
Как я уже сказал, мы работаем исключительно с корневыми деревьями, так что у всех наших деревьев есть корень. А еще мы работаем только со связными графами. Таким образом, самое важное, что следует запомнить, — *в деревьях не может быть циклов*.

Итак, вы увидели, как работает дерево. Рассмотрим подробнее один из его типов.

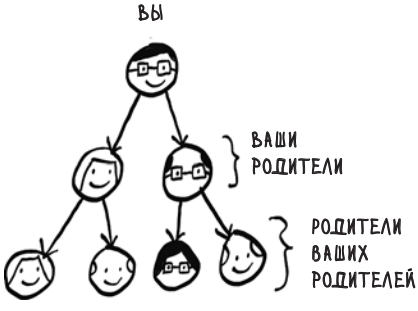
Бинарные деревья

В computer science полно разных деревьев. Бинарные деревья используются очень часто. Далее в этой и следующей главе мы будем в основном работать с бинарными деревьями.

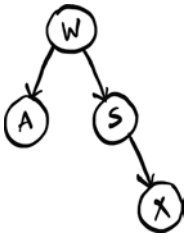
Бинарное дерево представляет собой особую разновидность дерева, узлы которого могут иметь не более двух дочерних узлов (отсюда и название). Дочерние узлы традиционно называются *левым* и *правым* узлами.



Пример бинарного дерева — генеалогическое древо, так как у каждого узла имеются два биологических родителя.



В этом примере между узлами существует четкая связь — все они составляют одну семью. Однако данные могут быть абсолютно произвольными.



Важно то, что ни у одного узла не может быть больше двух дочерних узлов. Иногда встречаются термины «левое поддерево» и «правое поддерево».