Деревья, построение дерева Хаффмана

Основная терминология

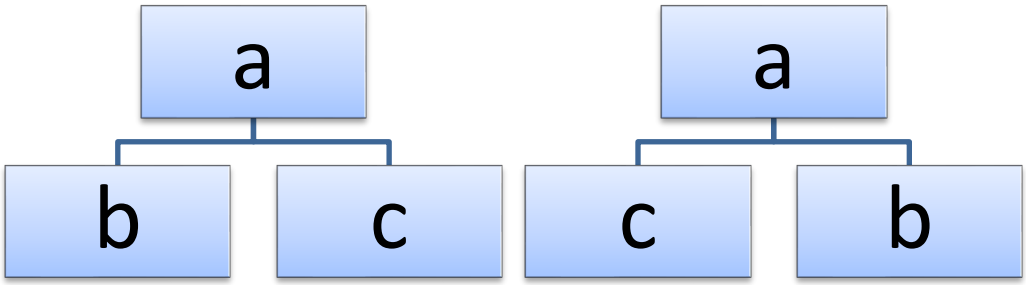
Определение. Дерево – это совокупность элементов, называемых узлами (один из которых определён, как корень), и отношений, образующих иерархическую структуру узлов. Узлы могут быть элементами любого типа. Формально дерево определяется рекуррентно следующим образом:

- Один узел является деревом. Этот же узел также является корнем этого дерева.
- Пусть n - это узел, а $T_1, T_2, \dots, T_k$ - деревья с корнями $n_1, n_2, \dots, n_k$ соответственно. Можно построить новое дерево, сделав n родителем узлов $n_1, n_2, \dots, n_k$. В этом дереве n будет корнем, а $T_1, T_2, \dots, T_k$ - поддеревья этого корня. Узлы $n_1, n_2, \dots, n_k$ называются сыновьями узла n.

Дерево без узлов называется нулевым деревом и обозначается Λ. Узлы дерева без потомков называются листьями.

Порядок узлов

Сыновья узла обычно упорядочиваются слева направо. Поэтому два дерева, приведённые ниже различны, так как порядок сыновей узла a различен.



Если порядок сыновей игнорируется, то такое дерево называют неупорядоченным, в противном случае дерево называется упорядоченным. Упорядочивание слева направо сыновей можно использовать для сопоставления узлов, которые не связаны отношениями предки-потомки. Соответствующее правило звучит следующим образом: если узлы a и b являются сыновьями одного родителя и узел a лежит слева от узла b, то все потомки узла a будут находиться слева от любых потомков узла b.

Прямой, обратный и симметричный обходы дерева

Существует несколько способов обхода узлов дерева. Три наиболее часто используемых способа обхода дерева называются обход в прямом порядке, обход в обратном порядке и обход во

внутреннем порядке (симметричный обход). Все три способа обхода рекурсивно можно определить следующим образом:

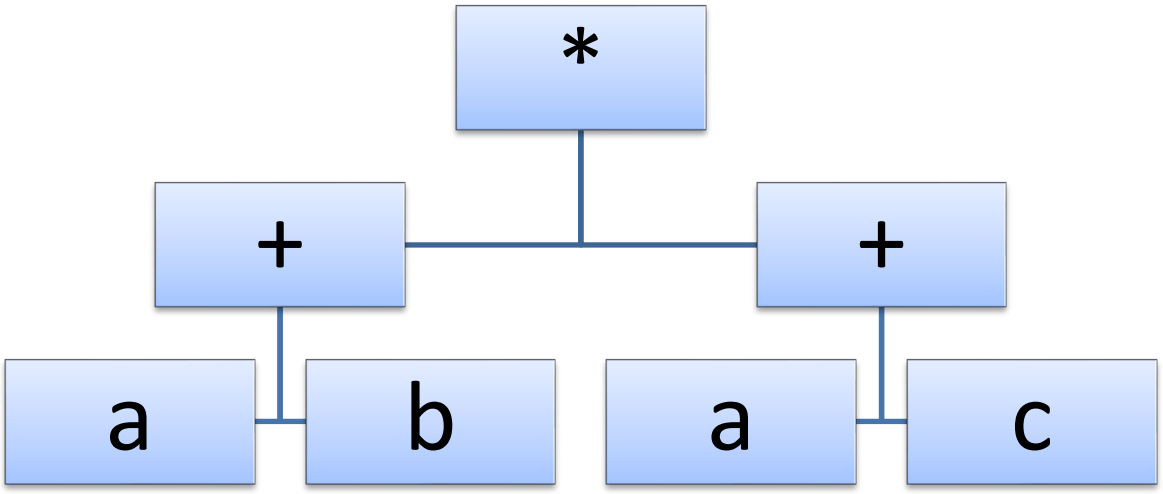
- Если дерево T является нулевым деревом, то в список обхода заносится пустая запись
- Если дерево T состоит из одного узла, то в список обхода записывается этот узел
- Если дерево T - дерево с корнем n и поддеревьями $T_1, T_2, \dots, T_k$, то для различных вариантов обхода имеем:
  1. При прохождении в прямом порядке узлов дерева T сначала посещается корень n, затем узлы поддерева T_1, далее все узлы поддерева T_2 и т.д. Последними посещаются узлы поддерева T_k.
  2. При симметричном обходе узлов дерева T сначала посещаются в симметричном порядке все узлы поддерева T_1, далее корень n, затем последовательно в симметричном порядке все узлы поддеревьев $T_2, \dots, T_k$.
  3. Во время обхода в обратном порядке сначала посещаются в обратном порядке все узлы поддерева T_1, затем последовательно посещаются все узлы поддеревьев $T_2, \dots, T_k$, также в обратном порядке, последним посещается корень n.

Помеченные деревья и деревья выражений

Часто бывает полезным сопоставить каждому узлу дереву метку или значение. Дерево, у которого узлам сопоставлены метки, называется помеченным деревом. Метка узла – это не имя узла, а значение, которое хранится в узле. В некоторых приложениях мы даже будем изменять значение метки, поскольку имя узла сохраняется постоянным.

Для представления арифметических выражений часто строятся деревья выражений. Правила составления меток этих деревьев следующие:

- Метка каждого листа соответствует операнду и содержит его значение.
- Метка каждого внутреннего узла соответствует оператору. Предположим, что узел n помечен бинарным оператором Θ и левый сын этого узла соответствует выражению E_1, а правый выражению E_2. Тогда узел n и его сыновья представляют выражение $E_1 \Theta E_2$.



Часто при обходе деревьев составляется список не имён узлов, а их меток. В случае дерева выражений при прямом упорядочивании получаем известную префиксную форму выражений, где оператор предшествует и левому, и правому операндам. Отметим, что в префиксных формах нет

необходимости отделять или выделять отдельные префиксные выражения скобками. Выражение в префиксной форме - $* + ab + ac$.

Обратное упорядочивание меток дерева выражений даёт так называемое постфиксное представление выражений. При использовании постфиксной формы также нет необходимости в использовании скобок. Выражение в постфиксной форме – $ab + ac + *$.

При симметричном обходе дерева выражений мы получим так называемую инфиксную форму выражении, которая совпадает с привычной нам «стандартной»формой записи выражений, но тоже не использует скобок. Выражение в инфиксной форме – $a + b * a + c$.

Абстрактная структура дерево

В этом разделе мы рассмотрим несколько полезных операторов, выполняемых над деревьями, и покажем, как использовать эти операторы в различных алгоритмах.

- $PARENT(n, T)$. Эта функция возвращает родителя узла n в дереве T. Если n является корнем, который не имеет родителя, то в этом случае возвращается Λ.
- $LEFTMOST\_CHILD(n, T)$. Эта функции возвращает самого левого сына узла n в дереве T. Если n является узлом, то возвращается Λ.
- $RIGHT\_SIBLING(n, T)$. Эта функции возвращает правого брата узла n в дереве T и значение Λ, если такового не существует. Для этого находится родитель p узла n и все сыновья узла p, затем среди этих сыновей находится узел, расположенный непосредственно справа от узла n.
- $LABEL(n, T)$. Возвращает метку узла n дерева T.
- $CREATEi(v, T_1, T_2, \dots, T_i)$. Эта функция, которая для каждого $i = 0,1,2, \dots$ создаёт новый корень r с меткой v и далее для этого корня создаёт i сыновей, которые становятся корнями поддеревьев $T_1, T_2, \dots, T_i$. Эти функции возвращают дерево с корнем r. Заметим, что если $i = 0$, то возвращается один узел r, который одновременно является и корнем, и листом.
- $ROOT(T)$. Эта функция возвращает узел, являющимся корнем дерева T. Если T пустое дерево, то возвращается Λ.
- $MAKENULL(T)$. Этот оператор делает дерево T пустым деревом.

Напишем процедуру обхода дерева в прямом порядке с использованием этих абстрактных функций. Причём напишем рекурсивный ($PREORDER$) и нерекурсивный ($NPREORDER$) варианты обхода дерева, начиная с узла n. Для обхода всего дерева необходимо выполнить вызов $PREORDER(ROOT(T))$.

Листинг 1

```
procedure PREORDER(n: node);
var
c: node;
begin
  writeln(LABEL(n,T));
  c:= LEFTMOST_CHILD(n,T);
  while c<> Λ do
  begin
    PREORDER(c);
    c:= RIGHT_SIBLING(n,T);
  end;
end;
```

Теперь напишем нерекурсивную процедуру для печати узлов дерева в прямом порядке. Чтобы совершить обход дерева, используем стек S. Основная идея разрабатываемого алгоритма заключается в том, что, когда мы дошли до узла n, стек хранит путь от корня до этого узла, причём корень находится на дне стека, а узел n - в вершине стека.

Алгоритм, который мы сейчас рассмотрим, выполняет два вида операций. Операции первого вида осуществляют обход по направлению к потомкам самого левого ещё не проверенного пути дерева до тех пор, пока не встретится лист, при этом выполняется печать узлов этого пути и занесение их в стек.

Во втором режиме выполнения программы осуществляется возврат по пройденному пути с поочерёдным извлечением узлов из стека до тех пор, пока не встретится узел, имеющий ещё не описанного правого брата. Тогда программа опять переходит в первый режим и исследует новый путь, начиная с правого брата.

Программа начинается в первом режиме с нахождения корня дерева и определения, является ли стек пустым.

Листинг 2

```
procedure NPREORDER(T: TREE);
var
m: node;
S: STACK;
begin
  MAKENULL(S);
  m:= ROOT(T);
  while true do
    if m<> Λ then
    begin
      writeln(LABEL(n,T));
      push(m,S);
      m:= LEFTMOST_CHILD(m,T);
    end
    else
    begin
      if EMPTY(S) then
        return;
    end;
end;
```

Двоичные деревья

Деревья, которые мы определили в начале, называются упорядоченными ориентированными деревьями, поскольку сыновья любого узла упорядочены слева направо, а пути по дереву ориентированы от начального узла пути к его потомкам. Двоичное или бинарное дерево – совершенно другой вид. Двоичное дерево может быть или пустым деревом, или деревом, у которого любой узел или не имеет сыновей, или имеет либо левого сына, либо правого сына, либо обоих. Тот факт, что каждый сын любого узла определён как левый или как правый сын, существенно отличает двоичное дерево от упорядоченного ориентированного дерева.

Коды Хаффмана

Приведём пример применения двоичных деревьев в качестве структур данных. Для этого рассмотрим задачу конструирования кодов Хаффмана. Предположим, мы имеем сообщения, состоящие из последовательности символов. В каждом сообщении символы независимы и появляются с известной частотой, не зависящей от позиции в сообщении. Например, мы имеем сообщения, состоящие из пяти символов a, b, c, d, e, которые появляются в сообщениях с вероятностью $0.12, 0.4, 0.15, 0.08, 0.25$ соответственно.

Мы хотим, закодировать каждый символ последовательностью из нулей и единиц так, чтобы код любого символа являлся префиксом кода сообщения, состоящего из последующих символов. Это префиксное свойство позволяет декодировать строку из нулей и единиц последовательным удалением префиксов (т.е. кодов символов) из этой строки.

Таблица 1

Символ	Вероятность	Код 1	Код 2
a	0.12	000	000
b	0.40	001	11
c	0.15	010	01
d	0.08	011	001
e	0.25	100	10

Ясно, что первый код обладает префиксным свойством, поскольку любая последовательность из трёх битов будет префиксом для другой последовательности из трёх битов. Проще говоря, префиксная последовательность однозначно идентифицируется символом. Алгоритм декодирования этого кода очень прост: надо поочерёдно брать по три бита и преобразовать каждую группу битов в соответствующие символы. Например, последовательность 001010011 соответствует исходному сообщению bcd.

Легко проверить, что второй код также обладает префиксными свойствами. Процесс декодирования здесь не отличается от аналогичного процесса для первого кода. Единственная сложность для второго кода заключается в том, что нельзя сразу всю последовательность битов разбить на отдельные сегменты, соответствующие символам, так как символы могут кодироваться и двумя, и тремя битами. Для примера рассмотрим двоичную последовательность 1101001, которая опять представляет символы bcd. Первые два бита 11 однозначно соответствуют символу b, поэтому их можно удалить, тогда получится 01001. Здесь 01 также однозначно определяет символ c и т.д.

Задача конструирования кодов Хаффмана заключается в следующем: имея множество символов и значения частоты их появления в сообщениях, построить такой код с префиксным свойством, чтобы средняя длина кода последовательности была минимальной. Мы хотим минимизировать среднюю длину кода для того, чтобы уменьшить длину вероятного сообщения. Чем короче среднее значение длины кода символов, тем короче закодированное сообщение. Первый код из таблицы имеет среднюю длину 3. Второй код имеет среднюю длину 2.2, поскольку символы a и d имеют суммарную вероятность 0.20 и длина их кода составляет три бита, тогда как другие символы имеют код длиной 2.

Но можно придумать код, который был бы лучше второго кода. Его средняя длина составит 2.15. Это наилучший возможный код для символов с заданной частотой. Способ нахождения оптимального префиксного кода называется алгоритмом Хаффмана. В этом алгоритме два символа с наименьшими частотами появления (например, a и b) заменяются фиктивным

символом x, частота появления которого равна сумме частот появления двух исходных символов. Затем, используя эту процедуру рекурсивно, находим оптимальный префиксный код для меньшего множества символов. Код для исходного множества символов получается из кодов замещающих символов путём добавления 0 и 1 перед кодом замещающего символа, и эти два новых кода принимаются как коды заменяемых символов. Например, код символа a будет соответствовать коду символа x с добавленным нулём перед этим кодом, а для кода символа b перед кодом символа x будет добавлена единица.

Можно рассматривать префиксные коды как пути на двоичном дереве: прохождение от узла к его левому сыну соответствует 0 в коде, а к правому сыну – 1. Если мы пометим листья дерева кодируемыми символами, то получим представление префиксного кода в виде двоичного дерева. Префиксное свойство гарантирует, что нет символом, которые были бы метками внутренних узлов дерева, и наоборот, помечая кодируемыми символами только листья дерева, мы обеспечиваем префиксное свойство этих кода этих символов.

Алгоритм построения кодов Хаффмана

Опишем алгоритм построения дерева Хаффмана и получения кодов Хаффмана.

1. Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который равен частоте появления символа
2. Выбираются два свободных узла дерева с наименьшими весами
3. Создается их родитель с весом, равным их суммарному весу
4. Родитель добавляется в список свободных узлов, а двое его детей удаляются из этого списка
5. Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой — бит 0
6. Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

С помощью этого алгоритма мы можем получить коды Хаффмана для заданного алфавита с учётом частоты появления символов.

Таблица 2

Символ	Вероятность	Код Хаффмана
a	0.12	1111
b	0.40	0
c	0.15	110
d	0.08	1110
e	0.25	10