**Деревья и леса**

**Основные определения**

Дерево – это иерархическая структура, представляющая собой иерархию элементов, называемых *узлами (вершинами).* На самом верхнем уровне иерархии имеется только один узел – *корень дерева.* Узлы могут быть элементами любого типа. Каждый узел, кроме корня, связан только с одним узлом на более высоком уровне, называемым *исходным узлом (предком, отцом).* Каждый элемент может быть связан посредством ребер (ветвей) с одним или несколькими элементами на следующем, более низком уровне *(с порожденными узлами, дочерними узлами, сыновьями, потомками).* Элементы, расположенные в конце ветви, т.е. не имеющие порожденных, называются *листьями.*

Путем из узла n1 до узла nk называется последовательность узлов n1,n2,…,nk, где для всех i є 1,k-1 узел ni является предком узла ni+1. От корня до любого узла существует только один путь.

Длиной пути называется число, на 1 меньшее числа узлов, составляющих этот путь (или количество ребер, соединяющих узлы данного пути).

Длина пути от корня до некоторого узла называется уровнем или глубиной уровня этого узла. Уровень корня нулевой .

Высотой узла называется длина *самого длинного пути* из этого узла до какого-либо листа. Высотой дерева называется высота его корня (т.е. наибольшая длина пути от корня до листьев дерева).

Любой узел дерева с его потомками на всех уровнях образует *поддерево.* Дочерние узлы этого узла также образуют поддеревья. Следовательно, каждый узел имеет столько поддеревьев, сколько у него дочерних узлов.

Во многих алгоритмах обработки деревьев используется понятие *пустого дерева*, т.е. дерева, не имеющего ни одного узла. Тогда возникает *рекурсивное определение дерева*: древовидная структура с базовым типом Т – это либо 1) пустая структура, либо 2) узел типа Т, с которым связано конечное число древовидных структур с базовым типом Т, называемых поддеревьями. Верхний узел называется корнем.

Число поддеревьев данного узла образует *степень узла*. Максимальное значение m степени всех узлов дерева является *степенью дерева.* Дерево степени m называется *m-арным деревом.* Если все узлы дерева, кроме листьев, имеют степень, равную m, то такое дерево называется *полным m-арным деревом.* Дерево степени 2 называется *бинарным (двоичным) деревом.* Оно также может быть полным или неполным.

высота узла В=3;

степень дерева =3;

(тернарное)

Степень узла: 3

уровни

Полное бинарное дерево:

C

А

N

Y

I

H

D

O

G

0й

3й

2й

a

c

b

2

1

3

B

L

E

1й

0

1

g

f

e

d

0

0

2

2

M

F

Листья имеют степень 0

0

0

0

0

0

K

Если в дереве на каждом уровне задан порядок следования вершин, то такое дерево называется упорядоченным деревом. Например,

b

c

a

a

и

c

b

- это два различных упорядоченных дерева.

Обычно деревья считаются упорядоченными. Порядок следования порожденных вершин любого узла считается заданным слева направо.

Каждая вершина m-арного дерева может быть описана строкой символов алфавита (0,1,…,m-1), при этом корень дерева описывается пустой строкой. Узлам первого уровня приписываются односимвольные строки 0,1,…,m-1; узлам второго уровня – двухсимвольные строки, причем первым является символ, описывающий предка данного узла;… узлы i-го уровня описываются строками, состоящими из i символов. Любой потомок узла nk k-го уровня характеризуется строкой, первые k символов которой (*префикс*) составляет описание предка (т.е. узла nk), а последним, (k+1)-м символом является один из символов 0,1,…,m-1 в зависимости от позиции узла-потомка в ниже лежащем (k+1)-м уровнем.

Строка, описывающая лист дерева, не является префиксом ни для каких других строк, описывающих другие узлы дерева, т.е. является уникальным.

Множество строк, соответствующих листьям некоторого дерева, образуют *префиксный код* этого дерева. Рассмотрим тернарное (m=3) дерево:

уровни

1й

2

1

0

2й

22

21

20

12

11

00

10

011

3й

210

012

011

010

Префиксный код дерева: (00,010,011,012,10,11,12,20,210,22) – он однозначно позволяет построить дерево.

Строки, описывающие листья, могут иметь разную длину, (в зависимости от уровня каждого листа). Если листьям дерева поставить в соответствие символы некоторого алфавита, то элементы префиксного кода дерева можно рассматривать как коды соответствующих символов алфавита. Тогда для символов алфавита получаем коды различной длины. Этот подход используется при сжатии данных по алгоритму Хаффмана, когда символы, встречающиеся чаще, имеют коды меньшей длины.

**Обходы деревьев**

Существует несколько способов обхода (прохождение) всех узлов дерева. Обход узлов дерева равнозначен упорядочиванию этих узлов по какому-либо правилу.

Известны 3 наиболее часто используемых способа обхода дерева:

1. обход в прямом порядке;
2. обход в обратном порядке;
3. обход во внутреннем порядке (или симметричный обход).

Все эти 3 способа обхода рекурсивно можно определить следующим образом:

- Если дерево T является пустым деревом, то в список обхода заносится пустая запись.

- Если дерево Т состоит из 1-го узла, то в список обход записывается этот узел.

- Пусть Т – дерево с корнем n и поддеревьями T1, T2,…,Tk, тогда для различныx способов обхода имеем следующее.

n

n

Tk

T2

…

T1

1. При обходе в прямом порядке (т.е. при прямом упорядочивании) узлов дерева Т сначала берется корень n, затем узлы поддерева T1, далее узлы поддерева Т2 и т.д. Последними берутся узлы Тk.

2. При обходе в обратном порядке сначала берутся в обратном порядке (т.е. от нижнего уровня к верхнему) все узлы поддерева Т1, затем последовательно берутся в обратном порядке все узлы поддеревьев Т2,…,Тk. Последним берется корень n.

3.При симметричном обходе узлов дерева Т сначала берутся все узлы поддерева Т1, далее корень n, затем последовательно все узлы поддеревьев Т2,…, Тk.

Рассмотрим три способа обхода на примере дерева

1

1. При прямом обходе надо нарисовать непрерывный контур вокруг дерева, начиная от корня дерева, рисуя контур против часовой стрелки и поочередно обходя все наружные части дерева. Далее записать узлы в соответствии с этим контуром:

7

6

5

3

4

2

1,2,3,5,8,9,6,10,4,7.

10

9

8

1. Обратный обход: 2,8,9,5,10,6,3,7,4,1

10

9

8

7

4

2

1

6

5

3

T1

1. Симметричный

T3

обход: 2,1,8,5,9,3,10,6,7,4

T2

Отметим, что при любом способе обхода листья записываются в порядке слева направо; при этом при симметричном обходе между потомками записывается предок.

**Представление деревьев в памяти компьютера**

Элементы древовидный структуры могут быть размещены как в последовательной (векторной) памяти, так и в динамически получаемых областях памяти.

1. Представление деревьев в векторной памяти.

Пусть Т-дерево с узлами 1,2,…,n (1-корень).

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i = | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  | 0 | 1 | 1 | 2 | 2 | 5 | 5 | 5 | 3 | 3 |

Самым простым представлением Т будет одномерный массив А, где каждый элемент А[i] равен номеру предка для узла i; А[i]=0, если узел i является корнем. Пример, дереву Т соответствует следующий массив А:

1

9

3

8

7

6

5

4

2

10

Данное представление использует то свойство деревьев, что каждый узел, отличный от корня, имеет только одного предка.

Используя это представление, предка любого узла можно найти за фиксированное время.

Представление дерева в векторной памяти допустимо только тогда, когда в процессе обработки объем памяти, занимаемой его элементами, не превышает фиксированного объема векторной памяти, т.е. число элементов дерева не должно превышать количества элементов массива. Элементы дерева занимают последовательные ячейки векторной памяти.

Пример: реализуем оператор определения для узла с номером n его “правого брата”, т.е. соседнего узла с номером, большим n, имеющего с данным узлом n одного и того же предка. При этом считается, что все узлы упорядочены по номерам слева направо (т.е. узел с большим номером лежит правее).

Type NODE=integer; {номера узлов}

TREE=array[1..maxnodes] of NODE; {массив, описывающий дерево}

{Кол-во узлов}

Function RIGHT\_SIBLING (n: NODE; T: TREE):node;

var i,parent:node;

begin RIGHT\_SIBLING:=0; {если правого брата не существует, то возвращается 0}

parent:=T[n];

for i:=n+1 to maxnodes do

if T[i]=parent then begin RIGHT\_SIBLING:=i;

break;

end;

end.

Перечислим другие полезные операторы, выполняемые над деревьями:

1. parent(n,T) – эта функция возвращает предка узла n в дереве Т. Если n

является корнем, то возвращается значение Λ – нулевой узел,

указывающее на выход за пределы дерева.

1. LEFTMOST\_CHILD(n,T) – функция возвращает самого левого потомка узла n

в дереве Т. Если n-лист, то возвращается Λ.

1. LABEL(n, T) – функция возвращает метку узла n дерева Т. Для выполнения

этой функции требуется, чтобы все узлы дерева были помечены.

1. ROOT(T) – функция возвращает узел, являющийся корнем дерева Т.

Если дерево Т пустое, то возвращается Λ.

1. MAKENULL(T) – делает Т пустым деревом.
2. CREATEi(v, T1,T2,…,Ti) – семейство функций, которые для каждого i=0,1,2…

создают новый корень r c меткой v и далее для этого корня создают i потомков, которые становятся корнями поддеревьев T1,T2,…,Ti. Эти функции возвращают дерево с корнем r. Если i=0, то возвращается один узел r, который и образует дерево.

1. Представление деревьев с помощью связанных списков .

Другой полезный способ представления деревьев состоит в формировании для каждого узла списка его потомков. Чаще всего для этих целей используют *связанные списки*.

Пример:

cellspace

header

|  |  |
| --- | --- |
| 2 |  |

|  |  |
| --- | --- |
| 3 |  |

|  |  |
| --- | --- |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |

1

8

7

6

5

4

2

|  |  |
| --- | --- |
| 4 |  |

|  |  |
| --- | --- |
| 5 |  |

|  |  |
| --- | --- |
| 9 |  |

|  |  |
| --- | --- |
| 10 |  |

|  |  |
| --- | --- |
| 6 |  |

|  |  |
| --- | --- |
| 7 |  |

|  |  |
| --- | --- |
| 8 |  |

10

9

3

Здесь имеется массив ячеек заголовков, индексированный номерами (они же являются именами) узлов. Каждый заголовок (header) указывает на связанный список, состоящий из узлов. Элементы списка header[i] являются потомками узла i.

Прежде чем разрабатывать необходимую структуру данных, надо в терминах абстрактного типа данных LIST(список узлов) сделать отдельную реализацию списков потомков и посмотреть, как эти списки согласуются между собой.

Начнем со следующего объявления переменной – списка потомков

var cellspace: array[1..maxnodes] of record

node: integer; {номер узла-потомка}

next: integer {номер след. узла-потомка}

end;

Дерево описывается типом

Type TREE =record

header: array[1..maxnodes] of integer; {массив из номеров узлов}

labels: array[1..maxnodes] of {тип меток};

end;

Предполагается, что корень дерева хранится отдельно в поле root. Для обозначения пустого (нулевого) узла используется 0.

Функция поиска предка для узла n будет следующей:

Function PARENT (n: integer; T: TREE): integer;

var p: integer; {номер узла – возможного предка узла n}

i: integer; {номер узла – потомка для узла p}

begin PARENT:=0; {нач. значение соответствует случаю, когда предок не найден,

т.е. для корня}

for p:=1 to maxnodes do

begin i:=T.header[p]; {i- потомок узла p}

while i<>0 do {если потомки у узла p еще есть}

if cellspace[i].node=n then PARENT:=p

else i:=cellspace[i].next

end

end;

Замечание для себя: в данном случае i=cellspace[i].node (номеру узла), а это в общем случае необязательно!

**Объединение деревьев в одно с использованием представления дерева с помощью связанных списков**

Для объединения двух и более деревьев в одно при представлении дерева с помощью связанных списков желательно, чтобы узлы всех деревьев располагались в одной общей области памяти. Каждый узел будет характеризоваться его меткой (label) самым левым потомком (leftmost\_child) и правым “братом” (right\_sibling), т.е. информация об узлах будет находиться в массиве.

var cellspace: array[1..maxnodes] of record

leftmost\_child: integer;

label:{тип метки};

right\_sibling: integer

end;

тип TREE будет целым (integer), соответствующим номеру корня дерева.

Опишем дерево Т вида

Воспроизведем дерево:

Корень

Д

С

В

10

5

11

2

А

10

|  |  |  |
| --- | --- | --- |
|  |  |  |
| - | Д | - |
|  |  |  |
| - | В | 11 |
|  |  |  |
| 5 | А | - |
| 2 | С | - |
|  |  |  |

А

с-д

2

11

С

В

5

Д

2

5

В и С – потомки одного предка (А) => связь А-С

а-в

|  |  |
| --- | --- |
| Т | 10 |

Корень

10

11

Поля: leftmost -child label right –sibling

Массив: cellspace

Используя такое представление, можно реализовать все операторы работы с деревьями, описанные выше, за исключением оператора PARENT (определение предка узла), путем прямых вычислений. Для выполнения оператора PARENT можно добавить в элементы массива cellspace 4-е поле для непосредственного указания на предка узла.

Рассмотрим в качестве примера реализацию функции CREATE2, возвращающей новое дерево с меткой корня V и поддеревьями Т1 и Т2. При этом предполагаем, что все неиспользованные ячейки массива cellspace связаны в один свободный список avail, и ячейки этого списка связаны посредством поля right\_sibling. На рис. показаны сплошными линиями старые связи и пунктирными – новые связи, появившиеся в процессе создания нового дерева.

|  |  |
| --- | --- |
| Т1 |  |

|  |  |  |
| --- | --- | --- |
|  |  | - |

T1

T2

T2

T1

T1

V

Корень Т2

Корень дерева Т1

|  |  |
| --- | --- |
| Т2 | установка в “0” |

|  |  |  |
| --- | --- | --- |
|  |  | - |

|  |  |  |
| --- | --- | --- |
|  |  |  |

…

|  |  |
| --- | --- |
| avail |  |

|  |  |  |
| --- | --- | --- |
| берется под новый корень |  |  |

|  |  |
| --- | --- |
| temp |  |

**V**

В принципе м.б. и другой

Function CREATE2(v: {тип метки}; T1,T2: integer): integer;

var temp: integer; {хранит индекс 1й свободной ячейки для корня нового дерева}

begin temp:=avail;

avail:=cellspace[avail].right\_sibling; {ячейки с этим номером становится 1й

свободной ячейкой}

cellspace[temp].leftmost\_child:=T1;

cellspace[temp].label:=V;

cellspace[temp].right\_sibling:=0;

cellspace[T1].right\_sibling:=T2; {T2 - правый брат для Т1}

CREATE2:=temp {корень нового дерева}

end;

Для уменьшения области памяти, занимаемой узлами дерева, можно в поле right\_sibling самого правого потомка вместо 0 поместить указатель на предка. Но тогда для избежания двусмысленности придется в каждую ячейку поместить логическую переменную, которая будет показывать, что содержится в поле right\_sibling: указатель на правого брата или указатель на предка. При этом увеличивается время выполнения операторов.

При такой реализации можно найти для заданного узла его родителя, следуя за указателями поля right\_sibling, пока не встретится указатель на предка. В этом случае время, необходимое для поиска предка, пропорционально количеству потомков у него.

**Бинарные (двоичные)**

**деревья поиска**

Бинарные деревья чаще всего применяются для представления множеств данных, элементы которых ищутся по уникальному, только им присущему ключу. Если бинарное дерево организовано таким образом, что для каждого узла *ti* все ключи в левом поддереве меньше ключа ti, а ключи в правом поддереве больше ключа ti, то это дерево называется бинарным деревом поиска. В дереве поиска можно найти место каждого ключа, двигаясь начиная от корня и переходя на левое или правое поддерево каждого узла в зависимости от значения ключа. Таким образом, места элементов в дереве определяются как значениями ключей, так и последовательностью их поступления. Определяющим фактором является значение ключа, от последовательности поступления элементов зависит степень сбалансированности дерева. При случайном распределении ключей в исходной последовательности получается почти сбалансированное дерево. Если же исходная последовательность упорядочена по возрастанию или убыванию ключей, то дерево вырождается в последовательный список. Высота такого дерева равна числу элементов дерева, уменьшенному на 1.

Создание дерева поиска заключается в следующем. Первый элемент образует корень дерева. Для последующих элементов осуществляется поиск места включения по ветвям дерева до тех пор, пока не будет найден подходящий узел с нулевым указателем, туда и подключается элемент. Для каждого узла запрашивается динамическая память, ее адрес заносится в указатель узла-предка, данные элемента помещаются в узел, и обнуляются левый и правый указатели нового узла. Дерево поиска, построенное из последовательности ключей

9,17,20,16,12,21,6,3,11,4,19,14,13,1,5,2,8,18,7,10,15,

имеет вид, приведенный на рис.

17

9

1

3

6

8

7

5

4

20

16

21

19

12

18

14

11

2

15

13

10

Далее см. 4 отсканированных листа (файлы с именами trees1 001,trees2 001,trees3 001,trees4 001.jpg)