**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Базы знаний и экспертные системы»**

**Тема: Рекурсивные структуры данных (деревья)**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6381 |  | Шевелева А.М. |
| Преподаватель |  | Сучков А.И. |

Санкт-Петербург

2019

Цель работы.

Изучение и исследование рекурсивных структур данных в языке Visual Prolog на примере деревьев.

Основные теоретические положения

Деревья, также как и списки, являются рекурсивным типом данных. Дерево – это структура данных, которая может быть разделена на корень дерева, левое и правое поддеревья. Так как левое и правое поддеревья в свою очередь являются деревьями, структура рекурсивна. Кроме того, дерево является еще и составным объектом данных.

Дерево, которое имеет только два поддерева, называется двоичным или бинарным. В том случае, если для каждого корня дерева выполняется условие, при котором значение, находящееся в корне дерева меньше значения, находящегося в корне левого поддерева и больше значения, находящегося в корне правого поддерева, двоичное дерево называется упорядоченным.

В Visual Prolog можно определить дерево следующим образом (см. рис. 1).

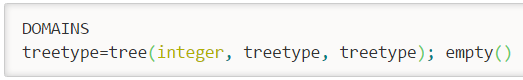


Рисунок 1 − Объявление списка

Такое определение говорит о том, что дерево является составным объектом, состоящим из трех составных частей: корня, принадлежащего домену integer и двух поддеревьев, принадлежащих домену treetype, так как именно этот домен и описывает структуру данных типа дерево. Так как дерево является составным объектом, его составные части объединяет функтор tree. Кроме того, дерево может находиться в двух состояниях: быть непустым (иметь хотя бы один корень) или пустым (не иметь ни одного корня). Пустое дерево описывается функтором empty без параметров. Если у функтора нет параметров, пустые скобки можно не указывать и записывать только имя функтора. Имена функторов tree и empty, домена treetype выбраны произвольно. Такое определение позволяет записать структуру данных:, представленную на рис. 2.

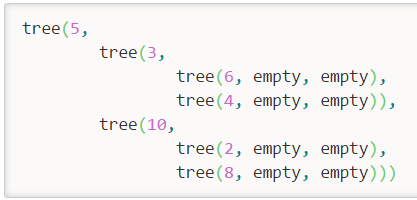


Рисунок 2 − Пример дерева

Следует отметить, что в большинстве случает рекурсия, используемая при работе с деревьями, хвостовой не является, так приходится обрабатывать левое и правое поддеревья, что дает две рекурсивные цели в одном предложении и, соответственно, не выполняется первое правило хвостовой рекурсии - рекурсивный вызов должен быть последней целью в хвостовой части правила вывода.

Постановка задачи.

Реализовать на языке Visual Prolog программу, выполняющую заданные операции над деревьями в соответствии с индивидуальным вариантом задания. Порядок выполнения работы следующий:

1. Написать на языке Visual Prolog программу, реализующую заданные операции над списками в соответствии с индивидуальным заданием.
2. Произвести отладку программы в системе Visual Prolog.
3. Построить трассу программы при выполнении каждого запроса.

Выполнение работы.

Написана программа на языке Visual Prolog считающая среднее арифметическое листьевых вершин бинарного дерева. Полный код программы представлен в приложении А.

На вход программа принимает дерево, которое впоследствии выводится слева направо. Программа считает сумму значений листьевых вершин и их количество, а потом находит среднее арифметическое листьевых вершин. Пример работы программы представлен на рис. 3.

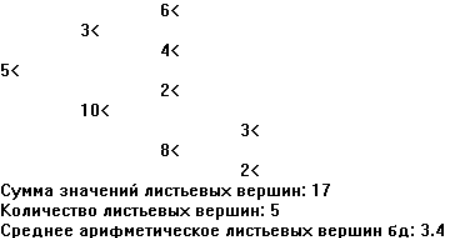


Рисунок 3 − Поиск среднего арифметического значения листьевых вершин

Трасса выполнения программы представлена в приложении Б.

Написана программа на языке Visual Prolog, проверяющая упорядоченность бинарного дерева. Полный код программы представлен в приложении В.

На вход программа принимает дерево, которое впоследствии выводится слева направо. Программа проверяет дерево на упорядоченность и выводит результат. Пример работы программы представлен на рис. 4-5.

|  |  |
| --- | --- |
|  |  |
| Рисунок 4 − Пример упорядоченного дерева | Рисунок 5 − Пример не упорядоченного дерева |

Трасса выполнения программы представлена в приложении Г.

Написана программа на языке Visual Prolog, которая вычисляет глубину бинарного дерева (глубина пустого дерева равна 0, глубина одноузлового дерева равна 1). Полный код программы представлен в приложении Д.

На вход программа принимает дерево, которое впоследствии выводится слева направо. Программа проверяет дерево на упорядоченность и выводит результат. Пример работы программы представлен на рис. 6.

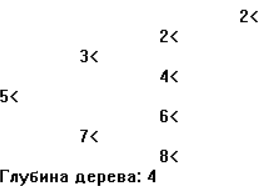


Рисунок 6 − Поиск глубины дерева

Трасса выполнения программы представлена в приложении Е.

Выводы.

В программе Visual Prolog были написаны три программы, реализующие задачи работы с деревьями:

1. Нахождение среднего арифметического листьевых вершин бд
2. Проверка упорядоченности бд;
3. Вычисление глубины бд.

Для каждой программы был создан иерархический вывод дерева слева направо, то есть слева находится корень, а справа поддеревья. При выводе узел сдвигался вправо в соответствии с номером уровня, на котором он расположен в дереве.

Каждая программа была протестирована, а также были созданы трассы для запросов каждой программы.

**ПРИЛОЖЕНИЕ А**

**КОД ПРОГРАММЫ, ВЫВОДЯЩИЙ ПОДСЧЕТ СРЕДНЕГО АРИФМЕТИЧЕСКОГО ЗНАЧЕНИЯ ЛИСТЬЕВЫХ ВЕРШИН**

domains

type = integer

tree = tree(type,tree,tree); empty

predicates

list\_count(tree,integer,integer)

print\_tree(tree,integer)

print\_spaces(integer)

clauses

list\_count(empty,0,0).

list\_count(tree(List,empty,empty),List,1):-!.

list\_count(tree(\_,L,R),List,N):-

list\_count(L,List1,N1),

list\_count(R,List2,N2),

N=N1+N2,

List=List1+List2.

print\_tree(empty, \_Depth):-!.

print\_tree(tree(TopValue, Left, Right), Depth):-

SubtreesDepth = Depth + 1,

print\_tree(Left, SubtreesDepth),

print\_spaces(Depth), write(TopValue), write("<"), nl,

print\_tree(Right, SubtreesDepth).

print\_spaces(SpaceNumber):-

SpaceNumber <= 0, !;

write("\t"),

TailSpaceNumber = SpaceNumber - 1,

print\_spaces(TailSpaceNumber).

goal

Tree=tree(5,tree(3,tree(6, empty, empty),tree(4, empty, empty)),

tree(10,tree(2, empty, empty),tree(8, tree(3, empty, empty),

tree(2, empty, empty)))),

print\_tree(Tree,0),

list\_count(Tree,List,N),write("Сумма значений листьевых вершин: ",List),nl,

write("Количество листьевых вершин: ",N),nl,

Sred=List/N,write("Среднее арифметическое листьевых вершин бд: ",Sred),nl.

**ПРИЛОЖЕНИЕ Б**

**ТРАССА ВЫПОЛНЕНИЯ ЗАПРОСА ПРОГРАММЫ ПОИСКА СРЕДНЕГО АРИФМЕТИЧЕСКОГО ЗНАЧЕНИЯ ЛИСТЬЕВЫХ ВЕРШИН**

Трасса выполнения запроса поиска среднего арифметического значения листьевых вершин дерева tree(5, tree(6, empty, empty), empty) представлена в табл. 1.

Таблица 1 − Трасса выполнения запроса поиска среднего арифметического

|  |  |  |
| --- | --- | --- |
| № шага | Переменная | Стек вызовов |
| 1 | Tree = \_  List = \_  N = \_  Sred = \_ | 1 \_PROLOG\_Goal() |
| 2 | Tree = tree(5, tree(6, empty, empty), empty)  List = \_  N = \_  Sred = \_ | 1 \_PROLOG\_Goal() |
| 3 | - | 1 list\_count(tree(5, tree(6, empty, empty), empty),\_,\_)  2 \_PROLOG\_Goal() |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| 4 | List = \_ | 1 list\_count(tree(5, tree(6, empty, empty), empty),\_,\_)  2 \_PROLOG\_Goal() |
| 5 | L = \_  R = 1  List = \_  N = \_  List1 = \_  N1 = \_  List2 = \_  N2 = \_ | 1 list\_count(tree(5, tree(6, empty, empty), empty),\_,\_)  2 \_PROLOG\_Goal() |
| 6 | L = tree(6,empty,empty)  R = empty  List = \_  N = \_  List1 = \_  N1 = \_  List2 = \_  N2 = \_ | 1 list\_count(tree(5, tree(6, empty, empty), empty),\_,\_)  2 \_PROLOG\_Goal() |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| 7 | - | 1 list\_count(tree (6, empty, empty),\_,\_)  2 1 list\_count(tree(5, tree(6, empty, empty), empty),\_,\_)  3 \_PROLOG\_Goal() |
| 8 | List = \_ | 1 list\_count(tree (6, empty, empty),\_,\_)  2 1 list\_count(tree(5, tree(6, empty, empty), empty),\_,\_)  3 \_PROLOG\_Goal() |
| 9 | L = tree(6,empty,empty)  R = empty  List = \_  N = \_  List1 = 6  N1 = 1  List2 = \_  N2 = \_ | 1 list\_count(tree(5, tree(6, empty, empty), empty),\_,\_)  2 \_PROLOG\_Goal() |
| 10 | - | 1 list\_count(empty,\_,\_)  2 1 list\_count(tree(5, tree(6, empty, empty), empty),\_,\_)  3 \_PROLOG\_Goal() |

Окончание таблицы 1

|  |  |  |
| --- | --- | --- |
| 11 | L = tree(6,empty,empty)  R = empty  List = \_  N = \_  List1 = 6  N1 = 1  List2 = 0  N2 = 0 | 1 list\_count(tree(5, tree(6, empty, empty), empty),\_,\_)  2 \_PROLOG\_Goal() |
| 12 | L = tree(6,empty,empty)  R = empty  List = \_  N = 1  List1 = 6  N1 = 1  List2 = 0  N2 = 0 | 1 list\_count(tree(5, tree(6, empty, empty), empty),\_,\_)  2 \_PROLOG\_Goal() |
| 13 | Tree = tree(5, tree(6, empty, empty), empty)  List = 6  N = 1  Sred = 6 | 1 \_PROLOG\_Goal() |

**ПРИЛОЖЕНИЕ В**

**КОД ПРОГРАММЫ, ПРОВЕРЯЮЩИЙ УПОРЯДОЧЕННОСТЬ ДЕРЕВА**

DOMAINS

Tree =tree(integer,Tree,Tree);empty()

PREDICATES

order(Tree)

order(Tree,integer,integer)

left(Tree,integer)

right(Tree,integer)

print\_tree(Tree, integer)

print\_spaces(integer)

CLAUSES

order(empty):-!.

order(tree(H,empty,RightTree)):-right(RightTree,R),order(RightTree,H,R),!.

order(tree(H,LeftTree,empty)):-left(LeftTree,L),order(LeftTree,L,H),!.

order(tree(H,LeftTree,RightTree)):-right(RightTree,R),left(LeftTree,L),

order(LeftTree,L,H),order(RightTree,H,R).

left(tree(H,empty,\_),H1):-!,H1=H-1.

left(tree(\_,LeftTree,\_),L):-left(LeftTree,L).

right(tree(H,\_,empty),H1):-!,H1=H+1.

right(tree(\_,\_,RightTree),R):-right(RightTree,R).

order(empty,\_,\_).

order(tree(H,LeftTree,RightTree),A,B):-H>A,H<B,

order(LeftTree,A,H),order(RightTree,H,B).

print\_tree(empty, \_Depth):-!.

print\_tree(tree(TopValue, Left, Right), Depth):-

SubtreesDepth = Depth + 1,

print\_tree(Left, SubtreesDepth),

print\_spaces(Depth), write(TopValue), write("<"), nl,

print\_tree(Right, SubtreesDepth).

print\_spaces(SpaceNumber):-

SpaceNumber <= 0, !;

write("\t"),

TailSpaceNumber = SpaceNumber - 1,

print\_spaces(TailSpaceNumber).

GOAL

Tree =tree(5,tree(3,tree(2,empty,empty),tree(9,empty,empty)),tree(7,tree(6,empty,empty),tree(8,empty,empty))),

print\_tree(Tree,0),

order(Tree),!,write("Дерево упорядоченное: ");write("Дерево не упорядоченное: ").

**ПРИЛОЖЕНИЕ Г**

**ТРАССА ВЫПОЛНЕНИЯ ЗАПРОСА ПРОВЕРКИ УПОРЯДОЧЕННОСТИ ДЕРЕВА**

Трасса выполнения запроса проверки упорядоченности дерева tree(5,tree(3,empty,empty),tree(9,empty,empty))представлена в табл. 2.

Таблица 2 − Трасса выполнения проверки упорядоченности дерева.

|  |  |  |
| --- | --- | --- |
| № шага | Переменная | Стек вызовов |
| 1 | Tree = \_ | 1 \_PROLOG\_Goal() |
| 2 | Tree = tree(5, tree(3, empty, empty), tree(9, empty, empty)) | 1 \_PROLOG\_Goal() |
| 3 | - | 1 order(tree(5, tree(3, empty, empty), tree(9, empty, empty)))  2 \_PROLOG\_Goal() |
| 4 | H = \_  RightTree = \_  R = \_ | 1 order(tree(5, tree(3, empty, empty), tree(9, empty, empty)))  2 \_PROLOG\_Goal() |
| 5 | H = \_  LeftTree = \_  L = \_ | 1 order(tree(5, tree(3, empty, empty), tree(9, empty, empty)))  2 \_PROLOG\_Goal() |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
| 6 | H = \_  LeftTree = \_  RightTree = \_  R = \_  L = \_ | 1 order(tree(5, tree(3, empty, empty), tree(9, empty, empty)))  2 \_PROLOG\_Goal() |
| 7 | H = 5  LeftTree = tree(3, empty, empty)  RightTree = tree(9, empty, empty)  R = 10  L = 2 | 1 order(tree(5, tree(3, empty, empty), tree(9, empty, empty)))  2 \_PROLOG\_Goal() |
| 8 | - | 1 order(tree (3, empty, empty), 2, 5)  2 order(tree(5, tree(3, empty, empty), tree(9, empty, empty)))  3 \_PROLOG\_Goal() |
| 9 | H = \_  LeftTree = \_  RightTree = \_  R = 2  L = 5 | 1 order(tree (3, empty, empty), 2, 5)  2 order(tree(5, tree(3, empty, empty), tree(9, empty, empty)))  3 \_PROLOG\_Goal() |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
| 10 | H = 3  LeftTree = empty  RightTree = empty  R = 2  L = 5 | 1 order(tree (3, empty, empty), 2, 5)  2 order(tree(5, tree(3, empty, empty), tree(9, empty, empty)))  3 \_PROLOG\_Goal() |
| 11 | - | 1 order(empty, 2, 3)  2 order(tree (3, empty, empty), 2, 5)  3 order(tree(5, tree(3, empty, empty), tree(9, empty, empty)))  4 \_PROLOG\_Goal() |
| 12 | H = 3  LeftTree = empty  RightTree = empty  R = 2  L = 5 | 1 order(tree (3, empty, empty), 2, 5)  2 order(tree(5, tree(3, empty, empty), tree(9, empty, empty)))  3 \_PROLOG\_Goal() |
| 13 | - | 1 order(empty, 3, 5)  2 order(tree(5, tree(3, empty, empty), tree(9, empty, empty)))  3 \_PROLOG\_Goal() |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
| 14 | H = 5  LeftTree = tree(3, empty, empty)  RightTree = tree(9, empty, empty)  R = 10  L = 2 | 1 order(tree(5, tree(3, empty, empty), tree(9, empty, empty)))  2 \_PROLOG\_Goal() |
| 15 | - | 1 order(tree(5, tree(3, empty, empty), tree(9, empty, empty)))  2 \_PROLOG\_Goal() |
| 16 | H = \_  LeftTree = \_  RightTree = \_  R = 5  L = 10 | 1 order(tree(5, tree(3, empty, empty), tree(9, empty, empty)))  2 \_PROLOG\_Goal() |
| 17 | H = 9  LeftTree = empty  RightTree = empty  R = 5  L = 10 | 1 order(tree(5, tree(3, empty, empty), tree(9, empty, empty)))  2 \_PROLOG\_Goal() |
| 18 | - | 1 order(empty, 5, 9)  2 order(tree(5, tree(3, empty, empty), tree(9, empty, empty)))  3 \_PROLOG\_Goal() |

Окончание таблицы 2

|  |  |  |
| --- | --- | --- |
| 19 | H = 9  LeftTree = empty  RightTree = empty  R = 5  L = 10 | 1 order(tree (9, empty, empty), 5, 10)  2 \_PROLOG\_Goal() |
| 20 | - | 1 order(empty, 9, 10)  2 \_PROLOG\_Goal() |
| 21 | Tree = tree(5, tree(3, empty, empty), tree(9, empty, empty)) | 1 \_PROLOG\_Goal() |

**ПРИЛОЖЕНИЕ Д**

**КОД ПРОГРАММЫ, ВЫЧИСЛЯЮЩИЙ ГЛУБИНУ ДЕРЕВА**

domains

tree = tree(integer, tree, tree); empty

predicates

tree\_height(tree, integer)

max(integer, integer, integer)

print\_tree(tree, integer)

print\_spaces(integer)

clauses

tree\_height(empty, 0):-!.

tree\_height(tree(\_Value, LeftSubtree, RightSubtree), Height):-

tree\_height(LeftSubtree, LeftSubtreeHeight),

tree\_height(RightSubtree, RightSubtreeheight),

max(LeftSubtreeHeight, RightSubtreeheight, MaxSubtreeHeight),

Height = MaxSubtreeHeight + 1.

max(A, B, A):- A > B, !.

max(\_A, B, B).

print\_tree(empty, \_Depth):-!.

print\_tree(tree(TopValue, Left, Right), Depth):-

SubtreesDepth = Depth + 1,

print\_tree(Left, SubtreesDepth),

print\_spaces(Depth), write(TopValue), write("<"), nl,

print\_tree(Right, SubtreesDepth).

print\_spaces(SpaceNumber):-

SpaceNumber <= 0, !;

write("\t"),

TailSpaceNumber = SpaceNumber - 1,

print\_spaces(TailSpaceNumber).

goal

Tree =tree(5,tree(3,tree(2,tree(2,empty,empty),empty),

tree(4,empty,empty)),tree(7,tree(6,empty,empty),tree(8,empty,empty))),

print\_tree(Tree,0),

tree\_height(Tree, D), write("Глубина дерева: ",D), nl.

**ПРИЛОЖЕНИЕ Е**

**ТРАССА ВЫПОЛНЕНИЯ ЗАПРОСА ВЫЧИСЛЕНИЯ ГЛУБИНЫ ДЕРЕВА**

Трасса выполнения запроса вычисления глубины дерева tree(2,empty,empty)представлена в табл. 3.

Таблица 3 − Трасса выполнения запроса вычисления глубины дерева

|  |  |  |
| --- | --- | --- |
| № шага | Переменная | Стек вызовов |
| 1 | Tree = \_  D = \_ | 1 \_PROLOG\_Goal() |
| 2 | Tree = tree(2,empty,empty)  D = \_ | 1 \_PROLOG\_Goal() |
| 3 | - | 1 tree\_height(tree(2,empty,empty),\_)  2 \_PROLOG\_Goal() |
| 4 | \_Value = \_  LeftSubtree = \_  RightSubtree = \_  Height = \_  LeftSubtreeHeight = \_  RightSubtreeheight = \_  MaxSubtreeheight = \_ | 1 tree\_height(tree(2,empty,empty),\_)  2 \_PROLOG\_Goal() |

Продолжение таблицы 3

|  |  |  |
| --- | --- | --- |
| 5 | \_Value = 2  LeftSubtree = empty  RightSubtree = empty  Height = \_  LeftSubtreeHeight = \_  RightSubtreeheight = \_  MaxSubtreeheight = \_ | 1 tree\_height(tree(2,empty,empty),\_)  2 \_PROLOG\_Goal() |
| 6 | - | 1 tree\_height(empty,\_)  2 tree\_height(tree(2,empty,empty),\_)  3 \_PROLOG\_Goal() |
| 7 | \_Value = 2  LeftSubtree = empty  RightSubtree = empty  Height = \_  LeftSubtreeHeight = 0  RightSubtreeheight = \_  MaxSubtreeheight = \_ | 1 tree\_height(tree(2,empty,empty),\_)  2 \_PROLOG\_Goal() |
| 8 | - | 1 tree\_height(empty,\_)  2 tree\_height(tree(2,empty,empty),\_)  3 \_PROLOG\_Goal() |

Окончание таблицы 3

|  |  |  |
| --- | --- | --- |
| 9 | \_Value = 2  LeftSubtree = empty  RightSubtree = empty  Height = \_  LeftSubtreeHeight = 0  RightSubtreeheight = 0  MaxSubtreeheight = \_ | 1 tree\_height(tree(2,empty,empty),\_)  2 \_PROLOG\_Goal() |
| 10 | A = 0  B = 0 | 1 max (0,0,\_)  2 tree\_height(tree(2,empty,empty),\_)  3 \_PROLOG\_Goal() |
| 11 | \_Value = 2  LeftSubtree = empty  RightSubtree = empty  Height = \_  LeftSubtreeHeight = 0  RightSubtreeheight = 0  MaxSubtreeheight = 0 | 1 tree\_height(tree(2,empty,empty),\_)  2 \_PROLOG\_Goal() |
| 12 | Tree = tree(2,empty,empty)  D = 1 | 1 \_PROLOG\_Goal() |