Т. 12. №6 2026
https://doi.org/10.33619/2414-2948/127

УДК 004.9 https://doi.org/10.33619/2414-2948/127/28

# ПРЕДСТАВЛЕНИЕ ДЕРЕВЬЕВ НА ЯЗЫКЕ PROLOG

***©Окулов М. Д.,*** *ORCID: 0009-0006-9243-6574, SPIN-код: 8379-2487, Национальный исследовательский университет «МЭИ», г. Москва, Россия, maxim999555@yandex.ru*
***©Денисенко В. К.,*** *ФГБОУ ВО «Национальный исследовательский университет «МЭИ», г. Москва, Россия, lubaok@bk.ru*

# REPRESENTATION OF TREES IN THE PROLOG LANGUAGE

***©Okulov M.,*** *ORCID: 0009-0006-9243-6574, SPIN-code: 8379-2487, National Research University Moscow Power Engineering Institute, Moscow, Russia, maxim999555@yandex.ru*
***©Denisenko V.,*** *National Research University Moscow Power Engineering Institute, Moscow, Russia, lubaok@bk.ru*

*Аннотация.* Данная статья посвящена представлению деревьев на языке Prolog, выявление их преимуществ и недостатков. Проведен обзор основных компонентов, а также исследованы различные методы и примеры использования для построения дерева. Рассмотрены основные этапы процесса разработки, включая моделирование знаний, разработку правил, а также оценку и тестирование системы. В результате предложены рекомендации для использования деревьев на практике.

*Abstract*. This article is devoted to the representation of trees in the Prolog language, identifying their advantages and disadvantages. An overview of the main components is carried out, as well as various methods and examples of use for building a tree are investigated. The main stages of the development process are considered, including knowledge modeling, rule development, as well as evaluation and testing of the system. As a result, recommendations for the use of trees in practice are proposed.

*Ключевые слова:* Prolog, деревья, код, разработка.

*Keywords:* Prolog, trees, code, development.

Деревья: Дерево в математике и информатике — это абстрактная структура данных, состоящая из узлов, связанных между собой ребрами. В корневом узле дерева находится один узел, от которого отходят ветви (ребра), которые в свою очередь могут иметь дочерние узлы (ветви), и так далее. Основные классификации деревьев: а) Бинарное (двоичное) дерево - каждый узел имеет по максимум два дочерних узла (две ветви); б) N-арное дерево - каждый узел имеет не более n дочерних узлов (n ветвей); в) Сбалансированное дерево - структура данных, в которой высота поддеревьев у каждого узла различается не более чем на 1; г) Дерево поиска - бинарное дерево с условием наличия правил сортировки (обычно значением в левой ветви меньше, чем в правой ветви). Примеры использования деревьев: а) Дерево разбора - используется в компиляторах для преобразования кода на языке программирования в промежуточное представление; б) Организация базы данных - дерево используется для упорядочивания и поиска информации; в) Алгоритм Дейкстры - также использует структуру дерева для нахождения кратчайшего пути в графе; г) Графический интерфейс пользователя - дерево используется для организации меню и структуры программ.

Т. 12. №6 2026
https://doi.org/10.33619/2414-2948/127

Описание деревьев на языке Prolog: в языке Prolog представление дерева основывается на использовании логических переменных и структур данных [1].

Логические переменные - это переменные, которые могут принимать только два значения: истина (true) или ложь (false). Структуры данных - это тип данных, обеспечивающие группировку значений переменных или объектов в один объект.

В языке Prolog дерево может быть представлено в виде структуры данных, которая имеет название (имя), назначение (арность) и атрибуты (значения), которые могут содержать логические переменные и другие структуры данных.

Пример представления дерева на языке Prolog

```
tree(A, B, C):
  A = 1, % верхняя вершина дерева
  B = tree(2, tree(3, nil, nil), tree(4, nil, nil)),  %левый и правый поддеревья
  C = tree(5, tree(6, nil, nil), tree(7, nil, nil)).  %левый и правый поддеревья
```

В данном примере создана структура данных "tree", которая содержит три атрибута: A: значение верхнего узла дерева; B: поддерево с левой стороны;C: поддерево с правой стороны;

Поддеревья также могут быть представлены структурой данных "tree" или, если узел является листом, путем назначения атрибутов "nil". Например: leaf(nil).

В этом коде определена структура данных "leaf" с нулевой арностью, которая может быть использована для представления листовых узлов в дереве.

Кроме того, на языке Prolog можно использовать рекурсивные функции для работы с деревьями, такие как обход дерева в глубину и ширину, поиск элемента в дереве, вставка и удаление элементов в дереве [2].

Для этого необходимо описать функции, которые будут рекурсивно вызывать себя для обработки следующих узлов дерева. Например, для обхода дерева в глубину можно использовать следующий код:

```
depth_first_search(nil).
depth_first_search(tree(Value, Left, Right)) :-
  write(Value), nl,
depth_first_search(Left),
depth_first_search(Right).
```

В этом фрагменте кода определены две функции:

"depth_first_search(nil)" - базовый случай для рекурсивной функции (когда нет узлов для обработки)

"depth_first_search(tree(Value, Left, Right))" - рекурсивно обрабатывает узел дерева и переходит к обработке следующих узлов

В данном случае обработка узла заключается в выводе значения "Value" на экран, а затем вызове "depth_first_search" для левого и правого поддеревьев для продолжения рекурсивного обхода дерева.

*Рекурсивные функции для работы с деревьями:*

*Обход дерева в глубину*

```
depth_first_search(nil).
depth_first_search(tree(Value, Left, Right)) :-
  write(Value), nl,
depth_first_search(Left),
depth_first_search(Right).
```

Эта функция рекурсивно обходит дерево в глубину и выводит значения узлов на экран. Она начинает с корня дерева (верхнего узла), выводит его значение и переходит к левому

Т. 12. №6 2026
https://doi.org/10.33619/2414-2948/127

поддереву. Если левое поддерево не существует, то функция заканчивает свою работу на этом уровне и переходит к правому поддереву. Функция продолжает рекурсивно вызывать себя для обработки узлов дерева до тех пор, пока все узлы не будут пройдены.

*Обход дерева в ширину*

```
breadth_first_search(Tree) :-
breadth_first_search([Tree], []).
breadth_first_search([], _) :- !.
breadth_first_search([nil|T], Processed) :-
breadth_first_search(T, Processed).
breadth_first_search([tree(Value, Left, Right)|T], Processed) :-
 write(Value), nl,
 append(T, [Left, Right], NewList),
 breadth_first_search(NewList, [Value|Processed]).
```

Эта функция обходит дерево в ширину, начиная с корня дерева. Она использует два списка: первый список "T" содержит все узлы, которые должны быть обработаны, а второй список "Processed" хранит значения уже обработанных узлов. Функция перебирает элементы первого списка, выводит их значения на экран и добавляет их потомков (если они существуют) в конец первого списка [3].

Использование append позволяет добавить все потомки сразу в конец списка. Далее функция вызывает себя рекурсивно с новыми списками, чтобы продолжить обработку узлов дерева.

*Поиск элемента в дереве*

```
search_tree(Element, tree(Element, _, _)) :- !.
search_tree(Element, tree(Value, Left, _)) :-
 Element < Value, !,
search_tree(Element, Left).
search_tree(Element, tree(_, _, Right)) :-
search_tree(Element, Right).
```

Эта функция рекурсивно ищет заданный элемент в дереве. Она начинает поиск с корня дерева.

Если значение текущего узла совпадает с заданным элементом, то поиск завершается и возвращается корневой узел.

Если заданный элемент меньше значения текущего узла, то функция вызывает search_tree для левого поддерева.

Если заданный элемент больше значения текущего узла, то функция вызывает search_tree для правого поддерева.

*Вставка элемента в дерево*

```
insert_tree(Element, nil, tree(Element, nil, nil)) :- !.
insert_tree(Element, tree(Element, Left, Right), tree(Element, Left, Right)) :- !.
insert_tree(Element, tree(Value, Left, Right), tree(Value, NewLeft, Right)) :-
Element < Value, !,
insert_tree(Element, Left, NewLeft).
insert_tree(Element, tree(Value, Left, Right), tree(Value, Left, NewRight)) :-
insert_tree(Element, Right, NewRight).
```

Эта функция рекурсивно вставляет заданный элемент в дерево.

Если дерево пустое, то создается новый корневой узел с заданным значением.

Т. 12. №6 2026
https://doi.org/10.33619/2414-2948/127

Если заданный элемент уже присутствует в дереве, то функция возвращает исходное дерево без изменений.

Если заданный элемент меньше значения текущего узла, то функция вызывает insert_tree для левого поддерева и создает новый узел слева от текущего узла.

Если заданный элемент больше значения текущего узла, то функция вызывает insert_tree для правого поддерева и создает новый узел справа от текущего узла.

*Удаление элемента из дерева*

```
delete_tree(_, nil, nil) :- !.
delete_tree(Element, tree(Element, nil, Right), Right) :- !.
delete_tree(Element, tree(Element, Left, nil), Left) :- !.
delete_tree(Element, tree(Element, Left, Right), tree(Replacement, NewLeft, Right)) :-
  find_replacement(Left, Replacement, NewLeft), !.
delete_tree(Element, tree(Value, Left, Right), tree(Value, NewLeft, Right)) :-
  Element < Value, !,
  delete_tree(Element, Left, NewLeft).
delete_tree(Element, tree(Value, Left, Right), tree(Value, Left, NewRight)) :-
  delete_tree(Element, Right, NewRight).
find_replacement(nil, _, nil) :-
  !.
find_replacement(tree(Replacement, nil, nil), Replacement, nil) :-
  !.
find_replacement(tree(Replacement, Left, nil), Replacement, Left) :-
  !.
find_replacement(tree(Value, Left, Right), Replacement, tree(Value, NewLeft, Right)) :-
  find_replacement(Left, Replacement, NewLeft).
```

Эта функция рекурсивно удаляет заданный элемент из дерева.

Если узел, который нужно удалить, не имеет потомков, то он просто удаляется из дерева.

Если у удаляемого узла есть только один потомок, то он заменяется потомком.

Если у удаляемого узла есть два потомка, то он заменяется на наибольший элемент в левом поддереве.

Для этого используется функция find_replacement, которая находит наибольший элемент в левом поддереве и возвращает его значение в качестве Replacement, а также новое левое поддерево. Далее функция удаляет заменяемый элемент из левого поддерева и заменяет им удаляемый узел. Если заданный элемент не найден в дереве, то функция возвращает исходное дерево без изменений.

Применение деревьев на языке Prolog может быть полезным при работе с базами данных, поисковыми алгоритмами и анализе текстовых данных [4].

Вот некоторые из примеров использования деревьев на языке Prolog.

*Использование деревьев для организации базы данных.* Деревья могут быть использованы для организации базы данных, чтобы упростить поиск и выдачу информации. (Например, можно создать дерево с категориями товаров и подкатегориями для магазина и хранить информацию о товарах в соответствующих узлах дерева). Пользователь может выбрать категорию или подкатегорию, а затем получить список товаров, относящихся к выбранной категории.

*Использование деревьев в поисковых алгоритмах.* Деревья могут использоваться в поисковых алгоритмах для быстрого и эффективного поиска. (Например, можно создать

Т. 12. №6 2026
https://doi.org/10.33619/2414-2948/127

дерево с ключевыми словами для поисковых запросов и их сочетаний и использовать его для быстрого поиска документов или страниц.)

*Использование деревьев для анализа текстовых данных.* Деревья могут использоваться для анализа структуры и содержания текстовых данных, таких как документы или страницы веб-сайта [5].

Например, можно создать дерево, которое отображает структуру веб-сайта, а затем использовать его для анализа содержания страниц и выявления повторяющихся паттернов или проблем с оптимизацией страницы. Также можно использовать деревья для выявления ключевых слов и терминов в тексте и их взаимосвязей. Пример простейшего дерева на языке Prolog (Рисунок. 1-10).

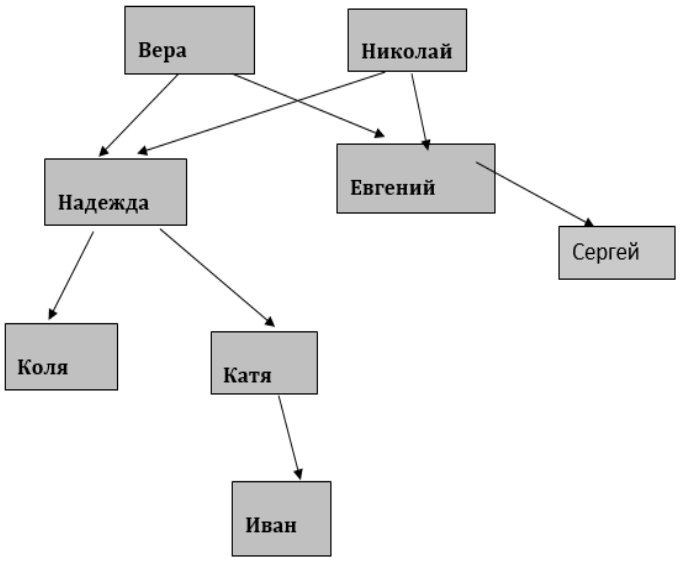


Рисунок 1. Генеалогическое дерево

```
parent('Вера','Евгений').
parent('Вера','Наджеда').
parent('Николай','Евгений').
parent('Николай','Надежда').
parent('Надежда','Коля').
parent('Надежда','Катя').
parent('Евгений','Сергей').
parent('Катя','Иван').

person('Вера','ж',86).
person('Николай','м',88).
person('Надежда','ж',61).
person('Евгений','м',64).
person('Коля','м',20).
person('Катя','ж',38).
person('Сергей','м',37).
person('Иван','м',7).
```

Рисунок 2. Создание дерева в Prolog

“X является отцом Y”

```
father(X,Y):- parent(X,Y), person(X,'м',_).
?- father(X,Y).
X = 'Николай',
Y = 'Евгений' ;
X = 'Николай',
Y = 'Надежда' ;
X = 'Евгений',
Y = 'Сергей' ;
false.
```

Рисунок 3. Результат 1 действия

“X является бабушкой Y”

```
grandmother(X,Y):-parent(X,W),parent(W,Y), person(X,'ж',_).
?- grandmother(X,Y).
X = 'Вера',
Y = 'Сергей' ;
X = 'Надежда',
Y = 'Иван' ;
false.
```

Рисунок 4. Результат 2 действия

Т. 12. №6 2026
https://doi.org/10.33619/2414-2948/127

“X является сестрой Y”

```
sister(X,Y):-parent(W,X),parent(W,Y),person(X,'ж',_),X\=Y.
```

```
?- sister(X,Y).
X = 'Надежда',
Y = 'Евгений' ;
X = 'Катя',
Y = 'Коля' ;
false.
```

Рисунок 5. Результат 3 действия

“X является племянником Y”

```
nephewn(X,Y):-parent(W,X),parent(E,Y),parent(E,W),person(X,'м',_),Y\=W.
```

```
?- nephewn(X,Y).
X = 'Коля',
Y = 'Евгений' ;
X = 'Сергей',
Y = 'Надежда' ;
X = 'Сергей',
Y = 'Надежда' ;
X = 'Иван',
Y = 'Коля' ;
false.
```

Рисунок 6. Результат 4 действия

“X является племянницей Y”

```
nephewd(X,Y):-parent(W,X),parent(E,Y),parent(E,W),person(X,'ж',_),Y\=W.
```

```
?- nephewd(X,Y).
X = 'Катя',
Y = 'Евгений' ;
false.
```

Рисунок 7. Результат 5 действия

“X является родителем родителя Y”

```
roditel(X,Y):-parent(X,W),parent(W,Y).
```

```
?- roditel(X,Y).
X = 'Вера',
Y = 'Сергей' ;
X = 'Николай',
Y = 'Сергей' ;
X = 'Николай',
Y = 'Коля' ;
X = 'Николай',
Y = 'Катя' ;
X = 'Надежда',
Y = 'Иван' ;
false.
```

Рисунок 8. Результат 6 действия

“X является прадедушкой Y”

```
preded(X,Y):-parent(X,W),parent(W,E),parent(E,Y),person(X,'м',_).
```

```
?- preded(X,Y).
X = 'Николай',
Y = 'Иван' ;
false.
```

Рисунок 9. Результат 7 действия

Т. 12. №6 2026
https://doi.org/10.33619/2414-2948/127

“X является двоюродным братом Y”

```
cousin(X,Y):-parent(W,X),parent(E,Y),parent(O,W),parent(O,E),person(X,'м',_),e\=W.
```

```
?- cousin(X,Y).
X = Y, Y = 'Коля' ;
X = 'Коля',
Y = 'Катя' ;
X = 'Коля',
Y = 'Сергей' ;
X = 'Сергей',
Y = 'Коля' ;
X = 'Сергей',
Y = 'Катя' ;
X = Y, Y = 'Сергей' ;
X = Y, Y = 'Сергей' ;
X = Y, Y = 'Иван'.
```

Рисунок 10. Результат 8 действия

*Создание бинарного дерева:*

```
% Предикат для создания пустого дерева
create(nil).
% Предикат для добавления элемента в бинарное дерево
insert(X, nil, tree(nil, X, nil)).
insert(X, tree(L, V, R), tree(L1, V, R)) :- X < V, insert(X, L, L1).
insert(X, tree(L, V, R), tree(L, V, R1)) :- X > V, insert(X, R, R1).
```

*Обход дерева в глубину (DFS)*

```
% Предикат для обхода дерева в глубину (DFS)
dfs(nil).
dfs(tree(L, V, R)) :- dfs(L), write(V), write(" "), dfs(R).
```

*Обход дерева в ширину (BFS)*

```
% Предикат для обхода дерева в ширину (BFS)
bfs(nil).
bfs(tree(L, V, R)) :- queue_create(Q), queue_push(Q, tree(L, V, R)), bfs(Q).
bfs(Q) :- queue_empty(Q).
bfs(Q) :- queue_pop(Q, tree(nil, V, nil)),
  write(V), write(" "),
  queue_push(Q, nil),
  queue_push(Q, nil).
bfs(Q) :- queue_pop(Q, tree(nil, V, R)),
  write(V), write(" "),
  queue_push(Q, R),
  queue_push(Q, nil).
bfs(Q) :- queue_pop(Q, tree(L, V, nil)),
  write(V), write(" "),
  queue_push(Q, nil),
  queue_push(Q, L).
bfs(Q) :- queue_pop(Q, tree(L, V, R)),
  write(V), write(" "),
  queue_push(Q, L),
  queue_push(Q, R),
```

Т. 12. №6 2026
https://doi.org/10.33619/2414-2948/127

```
bfs(Q).
```

*Проверка, является ли дерево сбалансированным*

```
% Предикат для проверки, является ли дерево сбалансированным
balanced(nil).
balanced(tree(L, _, R)) :- height(L, HL), height(R, HR), abs(HL-HR) =< 1, balanced(L), balanced(R).
% Предикат для вычисления высоты дерева
height(nil,0).
height(tree(L, _, R), N) :- height(L, LH), height(R, RH), max(LH, RH, H), N is H+1.
% Предикат для вычисления максимального значения
max(X,Y,X) :- X > Y.
max(X,Y,Y) :- X =< Y.
```

*Заключение*

Деревья — это важная структура данных, которая находит применение в различных областях. В целом, язык Prolog имеет удобный и элегантный синтаксис для работы с деревьями. Благодаря своей логической природе, он позволяет использовать рекурсивные определения для описания структуры и поведения деревьев. Применение деревьев на языке Prolog может быть полезным для организации баз данных, поисковых алгоритмов и анализа текстовых данных. Код на языке Prolog для работы с деревьями имеет различные предикаты для создания, добавления, удаления и обхода деревьев. Prolog является мощным инструментом для работы с деревьями и представляет прекрасную основу для создания сложных алгоритмов и структур данных.

*Список литературы:*

1. Залогова Л. А. Формальное описание механизма логического вывода в Прологе // Вестник Пермского университета. Серия: Математика. Механика. Информатика. 2014. №4 (27). С. 84-91.

2. Здор Д. В., Горностаева Т. Н. Анализ способов завершения рекурсии в рекурсивных правилах на языке логического программирования Пролог // Программные системы и вычислительные методы. 2021. №4. С. 68-76.

3. Новиков А. Д. О современных определениях функции и её исследовании // Вестник Сургутского государственного педагогического университета. 2013. №6 (27). С. 95-101.

4. Половикова О. Н., Зенков А. В. Решение некоторого класса логических задач на языке Prolog декларированием генераторов состояний // Компьютерные инструменты в образовании. 2019. №1. С. 54-67.

5. Половикова О. Н. Анализ семантических ошибок и ошибок выполнения, встречающихся в коде программ на языке Пролог // Известия Алтайского государственного университета. 2010. №1-2. С. 128-130.

*References:*

1. Zalogova, L. A. (2014). Formal`noe opisanie mexanizma logicheskogo vy`voda v Prologe. *Vestnik Permskogo universiteta. Seriya: Matematika. Mexanika. Informatika,* (4 (27)), 84-91. (in Russian).

2. Zdor, D. V., & Gornostaeva, T. N. (2021). Analiz sposobov zaversheniya rekursii v rekursivny`x pravilax na yazy`ke logicheskogo programmirovaniya Prolog. *Programmny`e sistemy` i vy`chislitel`ny`e metody`,* (4), 68-76. (in Russian).

3. Novikov, A. D. (2013). O sovremenny`x opredeleniyax funkcii i eyo issledovanii. *Vestnik Surgutskogo gosudarstvennogo pedagogicheskogo universiteta,* (6 (27)), 95-101. (in Russian).

4. Polovikova, O. N., & Zenkov, A. V. (2019). Reshenie nekotorogo klassa logicheskix zadach na yazy`ke Prolog deklarirovaniem generatorov sostoyanij. *Komp`yuterny`e instrumenty` v obrazovanii,* (1), 54-67. (in Russian).

5. Polovikova, O. N. (2010). Analiz semanticheskix oshibok i oshibok vy`polneniya, vstrechayushhixsya v kode programm na yazy`ke Prolog. *Izvestiya Altajskogo gosudarstvennogo universiteta,* (1-2), 128-130. (in Russian).

*Поступила в редакцию*
*19.04.2026 г.*

*Принята к публикации*
*28.04.2026 г.*

---

*Ссылка для цитирования:*

Окулов М. Д., Денисенко В. К. Представление деревьев на языке Prolog // Бюллетень науки и практики. 2026. Т. 12. №6. С. 224-232. https://doi.org/10.33619/2414-2948/12728

*Cite as (APA):*

Okulov, M., & Denisenko, V. (2026). Representation of Trees in the Prolog Language. *Bulletin of Science and Practice, 12*(6), 224-232. (in Russian). https://doi.org/10.33619/2414-2948/127/28