ЛАБОРАТОРНАЯ РАБОТА №2
РЕКУРСИЯ И СТРУКТУРЫ ДАННЫХ

1 Цель и задачи

Целью работы является изучение особенностей реализации рекурсии на языке Пролог, освоение принципов решения типовых логических программ.

Для достижения поставленной цели требуется решить следующие задачи:

1) Изучить теоретический материал.

2) Создать правила в соответствии с вариантом задания и общей формулировкой задачи (п.3).

3) Проверить выполнение программы.

4) Составить отчет о выполнении работы.

5) Представить на проверку файл отчета и файл текста программы на языке GNU Prolog, решающей поставленные задачи.

Номер варианта и текст варианта задания должны быть представлены в форме комментариев в тексте программы. Номер группы и номер варианта должны присутствовать в имени файла с текстом программы.

2 Основные теоретические сведения

Рассмотрим несколько вариантов использования рекурсивного вызова на языке Пролог применительно к спискам.

Принадлежность списку. Сформулируем задачу проверки принадлежности данного терма списку.

Граничное условие:

Терм R содержится в списке [H|T], если R=H.

Рекурсивное условие:

Терм R содержится в списке [H|T], если R содержится в списке T.

Первый вариант записи определения на Прологе имеет вид:

содержится (R, L) :- L=[H | T], H=R.

содержится(R, L) :- L=[H | T], содержится (R, T).

Цель L=[H | T] в теле обоих утверждений служит для того, чтобы разделить список L на голову и хвост.

Можно улучшить программу, если учесть тот факт, что Пролог сначала сопоставляет с целью голову утверждения, а затем пытается согласовать его тело. Новая процедура, которую мы назовем "принадлежит", определяется таким образом:

принадлежит (R, [R | T]).

принадлежит (R, [H | T]) :- принадлежит (R, T).

На запрос

?- принадлежит(a, [a, Ь, c]).

будет получен ответ

да

на запрос

?- принадлежит(b, [a, b, c]).

- ответ

да

но на запрос

?- принадлежит(d, (a, b, c)).

Пролог дает ответ

нет

В большинстве реализации Пролога предикат «принадлежит» является встроенным.

Соединение двух списков. Задача присоединения списка Q к списку P, в результате чего получается список R, формулируется следующим образом:

Граничное условие:

Присоединение списка Q к [] дает Q.

Рекурсивное условие:

Присоединение списка Q к концу списка P выполняется так: Q присоединяется к хвосту P, а затем спереди добавляется голова P.

Определение можно непосредственно написать на Прологе:

соединить([],Q,Q).

соединить(P,Q,R) :- P=[HP | TP], соединить(TP, Q, TR), R=[HP | TR].

Однако, как и в предыдущем примере, воспользуемся тем, что Пролог сопоставляет с целью голову утверждения, прежде чем пытаться согласовать тело:

присоединить([] ,Q,Q).

присоединить(HP | TP], Q, [HP | TR]) :- присоединить (TP, Q, TR).

На запрос

?- присоединить [a, b, c], [d, e], L).

будет получен ответ

L = [a, b, c, d].

но на запрос

?- присоединить([a, b], [c, d], [e, f]).

ответом будет No

Часто процедура «присоединить» используется для получения списков, находящихся слева и справа от данного элемента:

присоединить (L [джим, р], [джек,.билл, джим, тим, джим, боб]) .

L = [джек, билл]

R = [тим, джим, боб]

другие решения (да/нет)? да

L=[джек, билл, джим, тим]

R=[боб]

другие решения (да/нет)? да

других решений нет

Индексирование списка. Задача получения N-го терма в списке определяется следующим образом:

Граничное условие:

Первый терм в списке [H | T] есть H.

Рекурсивное условие:

N-й терм в списке [H | T] является (N-1)-м термом в списке T.

Данному определению соответствует программа:

```
/* Граничное условие:
получить ([H | T], 1, H).
/* Рекурсивное условие:
получить([H | T], N, Y) :- M is N - 1, получить (T, M ,Y).
```

3 Общая формулировка задачи

Реализуйте выполнение задания с номером варианта, равным номеру бригады (для каждого варианта - по две задачи, одна – из Задания 1, вторая – из Задания 2).

Под заданиями приведены примеры для проверки решений. Рекомендуется во всех заданиях использовать рекурсивную обработку списка, с разделением его элементов на голову и хвост; можно определять/использовать вспомогательные предикаты.

4 Пример выполнения задания

```
% Программа проверки вхождения элемента в список
member(Elem, [Elem|_]).
member(Elem, [Head|Tail]) :- member(Elem, Tail).
% Проверка
?- member(b, [a, X, c]).
```

X = b

yes

5 Перечень заданий

[Задание 1, Списки]

1. Вставить число в упорядоченный список

?- list_insert(2, [1,2,3,4], X).

X = [1,2,2,3,4]

2. Разбить заданный список пополам, на списки с элементами с четными и нечетными порядковыми номерами

?- div_list([a,b,c,d,e], X, Y).

X = [a,c,e]

Y = [b,d]

3. Проверить, является ли заданный список "палиндромным" (симметричным)

?- palind_list([1,2,3,4,5,4,3,2,1])

Yes

4. Осуществить перевод числа из десятичной системы счисления в двоичную; результат представить в виде списка двоичных цифр, читаемых слева-направо

?- binary(10, X).

X = [0,1,0,1]

5. Соединить два списка целых чисел в один, исключая все повторения чисел.

?- qlue_lists([1,2,3,4], [3,4,5,6], X).

X = [1,2,3,4,5,6]

6. Найти все отрицательные элементы в исходном числовом списке.

?- pos_list_elem([1,-2,3,-4,-5,6], X).

X = [-2,-4,-5]

7. Реализовать слияние двух упорядоченных по возрастанию списков с получением также упорядоченного списка.

?- lists_union([1,2,3], [2,3,4], X).
X = [1,2,2,3,3,4]

8. Обратить числовой список (переставить его элементы в обратном порядке)
?- inverse_list([1,2,3,4], X).
X = [4,3,2,1]

9. Сложить два числа, заданных в виде списка составляющих их цифр.
?- big_sum([1,2,3,4], [5,6,7,8], X)
X = [6,9,1,2]

10. Вставить число в упорядоченный список
?- list_insert(2, [1,2,3,4], X).
X = [1,2,2,3,4]

[Задание 2, Деревья]

1. Создайте предикат, проверяющий, что дерево является двоичным справочником.
?- is_ordered(tr(2,tr(7, nil, nil),tr(3,tr(4, nil, nil), tr(1, nil, nil)))).
No

2. Модифицируйте предикат, превращающий бинарное дерево в список с сохранением порядка элементов, чтобы на выходе получалось два списка, содержащих, соответственно, положительные и отрицательные значения.
?- make_tree_lists(tr(5, tr(-2, tr(-4, nil, nil), tr(-1, nil, nil)), tr(7, tr(6, nil, nil), nil)), X, Y).
X = [5,7,6];
Y = [-2,-4,-1];

3. Напишите предикат, проверяющий, является ли заданное бинарное дерево двоичным справочником (в каждом узле - в левом поддереве - все элементы, меньшие узлового, в правом - большие либо равные узловому)
?- isBinaryDict(tr(5, tr(4, nil, nil), tr(8, tr(6, tr(3, nil, nil), nil), tr(9, nil, nil)))).
No.

4. Реализуйте предикат, возвращающий количество листьев заданного бинарного дерева (т.е. узлов, не имеющих потомков)
?- countLeafs(tr(5, tr(4, nil, nil), tr(8, tr(6, tr(3, nil, nil), nil), tr(9, nil, nil))), X).

X = 3.

5. Реализуйте предикат, ищущий максимальный элемент в заданном числовом бинарном дереве.

?- maxTreeElem(tr(5, tr(4, nil, nil), tr(8, tr(6, tr(3, nil, nil), nil), tr(9, nil, nil))), X).

X = 9.

6. Реализуйте предикат, возвращающий путь от корня до заданного уникального элемента в бинарном дереве.

?- findTreePath(tr(5, tr(4, nil, nil), tr(8, tr(6, tr(3, nil, nil), nil), tr(9, nil, nil))), 3, X).

X = [5,8,6,3]

7. В заданном числовом бинарном дереве, поменяйте местами значения у максимального и минимального узлов.

?- swap_tree_minmax(tr(5, tr(4, nil, nil), tr(8, tr(6, tr(3, nil, nil), nil), tr(9, nil, nil))), X).

X = (tr(5, tr(4, nil, nil), tr(8, tr(6, tr(9, nil, nil), nil), tr(3, nil, nil)))

8. В заданном бинарном дереве, удалите все отрицательные элементы из листьев.

?- del_neg_leafs(tr(5, tr(4, nil, nil), tr(8, tr(6, tr(-3, nil, nil), nil), tr(-9, nil, nil))), X).

X = tr(5, tr(4, nil, nil), tr(8, tr(6, nil, nil), nil)

9. В заданном дереве, подсчитайте кол-во узлов, имеющих ровно 1 потомка.

?- count_single_nodes(tr(5, tr(3, tr(1, nil, nil), nil), tr(8, tr(6, nil, nil), tr(9, nil, nil))), X).

X = 1

10. Посчитайте среднее арифметическое всех числовых элементов бинарного дерева.

?- avg_tree(tr(5, tr(3, tr(1, nil, nil), nil), tr(8, tr(6, nil, nil), nil)), X).

X = 6.4

6 Содержание отчета

1. Номер варианта и задание.

2. Описание порядка выполнения.

3. Текст программы с комментариями.

4. Примеры вызова соответствующих правил (вопросы) и результаты выполнения (скрины).

5. Выводы с описанием роли каждого члена бригады, а также встретившихся трудностей и найденных способов их решения.