Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 3 по дисциплине «Функциональное и логическое программирование»

Тема Списки

Студент Александров Э.И.

Группа ИУ7-53БВ

Преподаватель Строганов Ю.В.

Москва, 2024

Содержание

ВВЕДЕНИЕ

Целью данной работы является разработка программы на языке Prolog, которая включает в себя реализацию различных алгоритмов для работы с числами и списками. Задачи, поставленные в рамках работы, включают:

1) создание правил для поиска объектов в базе знаний,
2) нахождение максимальных значений
3) вычисление среднего арифметического,
4) нахождение общих элементов в списках двумя способами.

1 Аналитическая часть

Prolog — это логический язык программирования, который используется для решения задач, связанных с искусственным интеллектом и обработкой данных. В данной работе мы будем использовать Prolog для реализации различных алгоритмов, включая нахождение максимальных значений и вычисление среднего арифметического.

Основные операции, которые мы будем реализовывать, включают использование предикатов для работы со списками и термами, а также применение встроенных функций языка.

Вывод

В аналитической части мы рассмотрели основные аспекты языка Prolog и его применение для решения задач, связанных с обработкой данных.

2 Конструкторская часть

В данной части работы мы подробно рассмотрим алгоритмы, которые были разработаны для решения поставленных задач, а также архитектуру программы на Prolog. Основное внимание будет уделено структуре кода, логике работы алгоритмов и их взаимодействию.

1) Поиск продуктов по калориям, белкам, жирам и углеводам

Мы создали три предиката, использующих findall для поиска продуктов по различным критериям:

— `найти_продукты_по_калориям/2` — находит продукты с калориями выше заданного минимума.

— `найти_продукты_по_белкам/2` — находит продукты с белками выше заданного минимума.

— `найти_продукты_по_жиру_и_углеводам/3` — находит продукты с содержанием жиров и углеводов ниже заданных максимумов.

Каждый из этих предикатов использует `findall`, чтобы собрать все подходящие продукты в список.

2) Нахождение наибольшего из двух и трех чисел

Мы реализовали предикаты для нахождения наибольшего числа:

— `max2/3` и `max3/4` с использованием отсечений и без них. Эти предикаты сравнивают числа и возвращают наибольшее.

3) Нахождение наибольшего элемента в списках

Для нахождения наибольшего элемента в списках целых чисел и термов мы создали:

— Самописные предикаты `max_element/2` и `max_term/2`, которые рекурсивно находят максимальные значения.

— Встроенные предикаты `max_list/2` и `max_member/2`, которые используют стандартные библиотеки Prolog.

4) Нахождение наибольшего элемента в списках списков

Мы реализовали предикаты для нахождения наибольшего элемента в списках списков как для чисел, так и для термов, используя как самописные, так и встроенные методы.

5) Вычисление среднего арифметического

Предикат `average/2` вычисляет среднее арифметическое списка целых чисел, используя вспомогательный предикат `sum_list/2`.

6) Нахождение общих элементов в двух списках

Мы создали два предиката для нахождения общих элементов:

— Самописный предикат `common_elements/3`, который рекурсивно ищет общие элементы.

— Встроенный предикат `intersection/3`, который использует стандартные функции Prolog.

Вывод

В конструкторской части мы разработали алгоритмы для решения поставленных задач и определили их архитектуру. Каждый алгоритм был спроектирован с учетом логики работы языка Prolog, что позволило эффективно реализовать необходимые функции.

3 Технологическая часть

Для реализации программы использовался язык Prolog, а именно среда разработки SWI-Prolog. Программа включает следующие ключевые алгоритмы:

1) Поиск продуктов:

```
% Калории, Белки, Жиры, Углеводы
food("chiken", 165, 31, 3.6, 0.0).
food("broccoli", 34, 3.0, 0.28, 0.0).
food("salmon", 206, 22.0, 13.0, 0.0).
food("Eggs", 155, 13.0, 11.0, 1.1).
food("tofu", 144, 15.0, 8.0, 3.9).
food("rice", 130, 2.7, 0.3, 28.0).

% жарка, варка, запекание, пар
cooking("fried", 1.2, 0.95, 1.5, 0.9).
cooking("boiled", 0.85, 0.98, 0.8, 0.95).
cooking("baking", 1.1, 0.98, 1.2, 0.95).
cooking("steam", 0.9, 0.99, 0.85, 0.98).

%1 Найти продукты по калориям
найти_продукты_по_калориям(КалорииМинимум, Продукты) :-
findall(Продукт, (food(Продукт, Калории, _, _, _),
Калории > КалорииМинимум), Продукты).

% ?-найти_продукты_по_калориям(140, X).

%1 Найти продукты по белкам
найти_продукты_по_белкам(БелкиМинимум, Продукты) :-
findall(Продукт, (food(Продукт, _, Белки, _, _),
Белки > БелкиМинимум), Продукты).

% ?-найти_продукты_по_белкам(13, X).

%1 Найти продукты по жиру и углеводам
найти_продукты_по_жиру_и_углеводам(ЖирыМакс, УглеводыМакс, Продукты) :-
findall(Продукт, (food(Продукт, _, _, Жиры, Углеводы),
Жиры < ЖирыМакс, Углеводы < УглеводыМакс), Продукты).

% ?-найти_продукты_по_жиру_и_углеводам(13,3.9,X).
```

2) Нахождение наибольшего из двух и трех чисел с отсечением и без:

```
%2 С отсечением
max3(A, B, C, A) :- A >= B, A >= C, !.
max3(_, B, C, B) :- B >= C, !.
max3(_, _, C, C).
% ?- max3(5,2,3,R)

max2(A, B, A) :- A >= B, !.
max2(_, B, B).
% ?- max2(3,2,X).


%2 Без отсечения
max3(A,B,C,R):-A>=B, A>=C, R=A.
max3(A,B,C,R):-B>=A, B>=C, R=B.
max3(A,B,C,R):-C>=A, C>=B, R=C.
% ?- max3(2,1,3,R)

max2(A,B,R):-A>=B, R=A.
max2(A,B,R):-B>=A, R=B.
% ?- max2(3,2,X).
```

3) Нахождение наибольшего элемента в списке из целых чисел:

```
%3 нахождение наибольшего элемента в списке
из целых чисел

% Самописное
% Базовый случай: наибольший элемент в списке,
состоящем из одного элемента, это сам элемент.
max_element([X], X).
% Рекурсивный случай: сравниваем голову списка
с наибольшим элементом хвоста.
max_element([H|T], Max) :-
max_element(T, MaxTail),  % Находим наибольший элемент в хвосте
Max is max(H, MaxTail).    % Сравниваем его с головой и выбираем большее.
% ?- max_element([3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5], Max).
```

```
%3 Встроенное
:- use_module(library(lists)).
% Пример использования max_list/2
find_max(List, Max) :-
max_list(List, Max).
% ?- find_max([3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5], Max).
```

4) Нахождение наибольшего элемента в списке из термов

```
%4 нахождение наибольшего элемента в списке из термов

% Самописное
% Базовый случай: наибольший элемент в списке,
состоящем из одного терма, это сам терм.
max_term([X], X).
% Рекурсивный случай: сравниваем голову списка с наибольшим термом хвоста.
max_term([H|T], Max) :-
max_term(T, MaxTail),  % Находим наибольший терм в хвосте
(   H @> MaxTail        % Сравниваем термы
->  Max = H             % Если голова больше, то она - наибольшая
;   Max = MaxTail       % Иначе наибольший терм - это хвост
).
% ?- max_term([apple, banana, cherry, date], Max).
% ?- max_term([1, 2, 3, 4, 5], Max).

%4 Встроенное
:- use_module(library(lists)).
% Пример использования max_list/2 для нахождения наибольшего терма
find_max_term(List, Max) :-
max_member(Max, List).
% ?- find_max_term([apple, banana, cherry, date], Max).
```

5) Нахождение наибольшего элемента в списке из списков чисел

```
%5 Нахождение наибольшего элемента в списке из списков чисел

% Самописное
max_in_list([X], X).
max_in_list([Head|Tail], Max) :-
```

```
max_in_list(Tail, MaxTail),
Max is max(Head, MaxTail).

max_in_list_in_list([List], Max) :-
max_in_list(List, Max).

max_in_list_in_list([Head|Tail], Max) :-
max_in_list(Head, MaxHead),
max_in_list_in_list(Tail, MaxTail),
Max is max(MaxHead, MaxTail).
% ?- max_in_list_in_list([[1, 2, 3], [4, 5, 6], [7, 8, 9]], Max).

%5 Встроенное
:- use_module(library(lists)).
max_in_lists(ListOfLists, Max) :-
maplist(max_list, ListOfLists, MaxList),  % Find max in each sublist
max_list(MaxList, Max).                    % Find max in the list of maxes
% ?- max_in_lists([[1, 2, 3], [4, 5, 6], [7, 8, 9]], Max).
```

6) Нахождение наибольшего элемента в списке из списков термов

```
%6 нахождение наибольшего элемента в списке из списков термов

% Самописное
max_term_in_list([X], X).
max_term_in_list([H|T], Max) :-
max_term_in_list(T, MaxTail),
(H @> MaxTail -> Max = H;
Max = MaxTail).

max_term_in_list_of_lists([List], Max) :-
max_term_in_list(List, Max).

max_term_in_list_of_lists([Head|Tail], Max) :-
max_term_in_list(Head, MaxHead),
max_term_in_list_of_lists(Tail, MaxTail),
(MaxHead @> MaxTail -> Max = MaxHead ; Max = MaxTail).
% ?- max_term_in_list_of_lists
([[apple,banana],[veg,potato],[orange,soup]],Max).
```

```
%6 Встроенное
max_term_in_list_std(List, Max) :-
member(Max, List),
forall(member(X, List), Max @>= X).

max_term_in_list_of_lists_std(ListOfLists, Max) :-
maplist(max_term_in_list_std, ListOfLists, MaxList),
max_term_in_list_std(MaxList, Max).
% ?- max_term_in_list_of_lists_std
([[apple,banana],[veg,potato],[orange,soup]],Max).
```

7) Нахождение среднего арифметического в списке из целых чисел

```
%7 нахождение среднего арифметического в списке из целых чисел
% Основной предикат для нахождения среднего арифметического
average(List, Average) :-
sum_list(List, Sum),
length(List, Length),
Length > 0,  % Проверяем, что список не пустой
Average is Sum / Length.
% Предикат для нахождения суммы элементов списка
sum_list([], 0).  % Базовый случай: сумма пустого списка равна 0
sum_list([H|T], Sum) :-
sum_list(T, TailSum),  % Рекурсивно находим сумму хвоста списка
Sum is H + TailSum.
% ?- average([1, 2, 3, 4, 5], Average).
```

8) Нахождение общих элементов в двух списках

```
%8 нахождение общих элементов в двух списках

% Самописное
% Основной предикат для нахождения общих элементов
common_elements([], _, []).  % Если первый список пуст,
результат пустой
common_elements(_, [], []).  % Если второй список пуст,
результат пустой
common_elements([H|T1], L2, [H|Common]) :-
```

```
member(H, L2),  % Проверяем, есть ли голова первого списка
во втором списке
common_elements(T1, L2, Common).  % Рекурсивно ищем в хвосте
common_elements([H|T1], L2, Common) :-
\+ member(H, L2),  % Если голова не во втором списке
common_elements(T1, L2, Common).  % Рекурсивно ищем в хвосте
% ?- common_elements_std([apple,banana,potato],
[apple,orange,grape,potato],X).

%8 Встроенное
common_elements_std(List1, List2, Common) :-
intersection(List1, List2, Common).
% ?- common_elements_std([apple,banana,potato],
[apple,orange,grape,potato],X).
```

Тестовые данные:

1) Для проверки работы с числами были использованы различные наборы чисел (2 и 3 числа) для проверки правильности нахождения максимального значения.

2) Для работы со списками использовались списки чисел и термов.

Все тесты прошли успешно.

Вывод

В технологической части мы реализовали алгоритмы на Prolog и протестировали их, подтвердив корректность работы.

ЗАКЛЮЧЕНИЕ

В ходе работы была достигнута цель — разработка программы на Prolog для выполнения различных операций с числами и списками. Операций, таких как:

1) создание правил для поиска объектов в базе знаний,
2) нахождение максимальных значений,
3) вычисление среднего арифметического,
4) нахождение общих элементов в списках двумя способами.

Все поставленные задачи были успешно выполнены, тесты пройдены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сергиевский Г. М., Волченков Н. Г. Функциональное и логическое программирование: учебник. – Москва: Издательский центр Академия, 2010. – 320 с.
2. Документация swi-prolog — [https://www.swi-prolog.org/]
3. Введение в логическое программирование (Prolog) — [https://pro-prof.com].
4. Prolog — [https://en.wikipedia.org/wiki/Prolog]

Приложение А

Условие:

1) Создайте отчёт и программу на Prolog в созданной в прошлом проекте базе знаний написать правил, использующих findall, для поиска всех объектов, подходящих под ограничения (не меньше 3 правил, в каждом правиле от 1 до 3 ограничений),
кроме того в работе необходимо реализовать:

2) нахождение наибольшего из 2х чисел, из 3х чисел (без использования отсечений и с использованием отсечений);

3) нахождение наибольшего элемента в списке из целых чисел (2 способа: самописное правило, стандартное правило из документации);

4) нахождение наибольшего элемента в списке из термов (2 способа: самописное правило, стандартное правило из документации);

5) нахождение наибольшего элемента в списке из списков чисел (2 способа: самописное правило, стандартное правило из документации);

6) нахождение наибольшего элемента в списке из списков термов (2 способа: самописное правило, стандартное правило из документации);

7) нахождение среднего арифметического в списке из целых чисел;

8) нахождение общих элементов в двух списках (2 способа);

Код программы:

```
% Калории, Белки, Жиры, Углеводы
food("chiken", 165, 31, 3.6, 0.0).
food("broccoli", 34, 3.0, 0.28, 0.0).
food("salmon", 206, 22.0, 13.0, 0.0).
food("Eggs", 155, 13.0, 11.0, 1.1).
food("tofu", 144, 15.0, 8.0, 3.9).
food("rice", 130, 2.7, 0.3, 28.0).

% жарка, варка, запекание, пар
cooking("fried", 1.2, 0.95, 1.5, 0.9).
cooking("boiled", 0.85, 0.98, 0.8, 0.95).
cooking("baking", 1.1, 0.98, 1.2, 0.95).
cooking("steam", 0.9, 0.99, 0.85, 0.98).

% списки
%1 Найти продукты по калориям
найти_продукты_по_калориям(КалорииМинимум, Продукты) :-
findall(Продукт, (food(Продукт, Калории, _, _, _),
Калории > КалорииМинимум), Продукты).
```

```
% ?-найти_продукты_по_калориям(140, X).

%1 Найти продукты по белкам
найти_продукты_по_белкам(БелкиМинимум, Продукты) :-
findall(Продукт, (food(Продукт, _, Белки, _, _),
Белки > БелкиМинимум), Продукты).
% ?-найти_продукты_по_белкам(13, X).

%1 Найти продукты по жиру и углеводам
найти_продукты_по_жиру_и_углеводам(ЖирыМакс, УглеводыМакс, Продукты) :-
findall(Продукт, (food(Продукт, _, _, Жиры, Углеводы),
Жиры < ЖирыМакс, Углеводы < УглеводыМакс), Продукты).
% ?-найти_продукты_по_жиру_и_углеводам(13,3.9,X).

%2
% Наибольшее из трех чисел с отсечением
max3(A,B,C,A):-A>=B,A>=C,!.
max3(_,B,C,B):-B>=C,!.
max3(_,_,C,C).
% ?- max3(2,1,3,R)

%2
% Наибольшее из двух чисел с отсечением
max2(A,B,A):-A>=B,!.
max2(_,B,B).
% ?- max2(3,2,X).

%2
% Наибольшее из трех чисел без отсечения
max3(A,B,C,R):-A>=B, A>=C, R=A.
max3(A,B,C,R):-B>=A, B>=C, R=B.
max3(A,B,C,R):-C>=A, C>=B, R=C.
% ?- max3(2,1,3,R)

%2
% Наибольшее из двух чисел без отсечения
max2(A,B,R):-A>=B, R=A.
max2(A,B,R):-B>=A, R=B.
% ?- max2(3,2,X).
```

```
%3 нахождение наибольшего элемента в списке из целых чисел
% Самописное
% Базовый случай: наибольший элемент в списке,
состоящем из одного элемента, это сам элемент.
max_element([X], X).
% Рекурсивный случай: сравниваем голову списка с наибольшим элементом хвоста.
max_element([H|T], Max) :-
max_element(T, MaxTail),  % Находим наибольший элемент в хвосте
Max is max(H, MaxTail).    % Сравниваем его с головой и выбираем большее.
% ?- max_element([3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5], Max).

%3 Встроенное
:- use_module(library(lists)).
% Пример использования max_list/2
find_max(List, Max) :-
max_list(List, Max).
% ?- find_max([3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5], Max).


%4 нахождение наибольшего элемента в списке из термов
% Самописное
% Базовый случай: наибольший элемент в списке,
состоящем из одного терма, это сам терм.
max_term([X], X).
% Рекурсивный случай: сравниваем голову списка с наибольшим термом хвоста.
max_term([H|T], Max) :-
max_term(T, MaxTail),  % Находим наибольший терм в хвосте
(   H @> MaxTail        % Сравниваем термы
->  Max = H            % Если голова больше, то она - наибольшая
;   Max = MaxTail      % Иначе наибольший терм - это хвост
).
% ?- max_term([apple, banana, cherry, date], Max).
% ?- max_term([1, 2, 3, 4, 5], Max).

%4 Встроенное
:- use_module(library(lists)).
% Пример использования max_list/2 для нахождения наибольшего терма
```

```
find_max_term(List, Max) :-
max_member(Max, List).
% ?- find_max_term([apple, banana, cherry, date], Max).


%5 Нахождение наибольшего элемента в списке из списков чисел
% Самописное
max_in_list([X], X).
max_in_list([Head|Tail], Max) :-
max_in_list(Tail, MaxTail),
Max is max(Head, MaxTail).

max_in_list_in_list([List], Max) :-
max_in_list(List, Max).

max_in_list_in_list([Head|Tail], Max) :-
max_in_list(Head, MaxHead),
max_in_list_in_list(Tail, MaxTail),
Max is max(MaxHead, MaxTail).
% ?- max_in_list_in_list([[1, 2, 3], [4, 5, 6], [7, 8, 9]], Max).

%5 Встроенное
:- use_module(library(lists)).
max_in_lists(ListOfLists, Max) :-
maplist(max_list, ListOfLists, MaxList),  % Find max in each sublist
max_list(MaxList, Max).                    % Find max in the list of maxes
% ?- max_in_lists([[1, 2, 3], [4, 5, 6], [7, 8, 9]], Max).


%6 нахождение наибольшего элемента в списке из списков термов
% Самописное
max_term_in_list([X], X).
max_term_in_list([H|T], Max) :-
max_term_in_list(T, MaxTail),
(H @> MaxTail -> Max = H;
Max = MaxTail).

max_term_in_list_of_lists([List], Max) :-
max_term_in_list(List, Max).
```

```
max_term_in_list_of_lists([Head|Tail], Max) :-
max_term_in_list(Head, MaxHead),
max_term_in_list_of_lists(Tail, MaxTail),
(MaxHead @> MaxTail -> Max = MaxHead ; Max = MaxTail).
% ?- max_term_in_list_of_lists([[apple,banana],[veg,potato],
[orange,soup]],Max).

%6 Встроенное
max_term_in_list_std(List, Max) :-
member(Max, List),
forall(member(X, List), Max @>= X).

max_term_in_list_of_lists_std(ListOfLists, Max) :-
maplist(max_term_in_list_std, ListOfLists, MaxList),
max_term_in_list_std(MaxList, Max).
% ?- max_term_in_list_of_lists_std([[apple,banana],[veg,potato],
[orange,soup]],Max).


%7 нахождение среднего арифметического в списке из целых чисел
% Основной предикат для нахождения среднего арифметического
average(List, Average) :-
sum_list(List, Sum),
length(List, Length),
Length > 0,  % Проверяем, что список не пустой
Average is Sum / Length.
% Предикат для нахождения суммы элементов списка
sum_list([], 0).  % Базовый случай: сумма пустого списка равна 0
sum_list([H|T], Sum) :-
sum_list(T, TailSum),  % Рекурсивно находим сумму хвоста списка
Sum is H + TailSum.
% ?- average([1, 2, 3, 4, 5], Average).


%8 нахождение общих элементов в двух списках
% Самописное
% Основной предикат для нахождения общих элементов
common_elements([], _, []).  % Если первый список пуст, результат пустой
```

```
common_elements(_, [], []).  % Если второй список пуст, результат пустой
common_elements([H|T1], L2, [H|Common]) :-
member(H, L2),  % Проверяем, есть ли голова первого списка во втором списке
common_elements(T1, L2, Common).  % Рекурсивно ищем в хвосте
common_elements([H|T1], L2, Common) :-
\+ member(H, L2),  % Если голова не во втором списке
common_elements(T1, L2, Common).  % Рекурсивно ищем в хвосте
% ?- common_elements_std([apple,banana,potato],[apple,orange,grape,potato],X).

%8 Встроенное
common_elements_std(List1, List2, Common) :-
intersection(List1, List2, Common).
% ?- common_elements_std([apple,banana,potato],[apple,orange,grape,potato],X).
```