Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных технологий

Кафедра математики и цифровых технологий

**ОТЧЕТ**

по лабораторной работе № 6

по дисциплине «Функционально-логическое программирование»

ОГУ 02.03.02. 4025. 583 О

Руководитель

Старший преподаватель кафедры МЦТ,

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_И.В. Минина

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г.

Студент группы 22ФИИТ(б)РАИС

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_И.Д. Евдокимов

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2025 г.

Оренбург 2025

**Пузырьковая сортировка**

Идея этого метода заключается в следующем. На каждом шаге сравниваются два соседних элемента списка. Если оказывается, что они стоят неправильно, то есть предыдущий элемент меньше следующего, то они меняются местами. Этот процесс продолжаем до тех пор, пока есть пары соседних элементов, расположенные в неправильном порядке. Это и будет означать, что список отсортирован.

Реализуем *пузырьковую сортировку* посредством двух предикатов. Один из них, назовем его permutation, будет сравнивать два первых элемента списка и в случае, если первый окажется больше второго, менять их местами. Если же первая пара расположена в правильном порядке, этот предикат будет переходить к рассмотрению хвоста.

Основной предикат bubble будет осуществлять *пузырьковую сортировку* списка, используя вспомогательный предикат permutation.

permutation([X,Y|T],[Y,X|T]):–

X>Y,!.

/\* переставляем первые два элемента, если первый больше

второго \*/

permutation([X|T],[X|T1]):–

permutation(T,T1).

/\*переходим к перестановкам в хвосте\*/

bubble(L,L1):–

permutation(L,LL), /\* вызываем предикат,

осуществляющий перестановку \*/

!,

bubble(LL,L1). /\* пытаемся еще раз отсортировать

полученный список \*/

bubble(L,L). /\* если перестановок не было, значит список

отсортирован \*/

**Код:**

% Пузырьковая сортировка

% Идея этого метода заключается в следующем.

% На каждом шаге сравниваются два соседних элемента списка.

% Если оказывается, что они стоят неправильно, то есть предыдущий элемент меньше следующего,

% то они меняются местами. Этот процесс продолжаем до тех пор,

% пока есть пары соседних элементов, расположенные в неправильном порядке.

% Это и будет означать, что список отсортирован.

% В этой лабе под пузырьком я подразумеваю элемент, который должен изменить свое положение

% Предикат оболочка для пузырьковой сортировки

bubblesort(List):-

sorting(List, Result),

write(Result).

% Предикат сортировки

% Если после попытки поднять один пузырек наверх ничего не изменилось - массив уже отсортирован

sorting(List, List):-

bubbleup(List, List).

% Если после поднимания пузырька массив изменился (т.е. какой-то пузырек поднялся максимально возможно,

% а, значит, отсортированный фрагмент концовки удлинился на 1), то снова пытаемся отсортировать массив

sorting(List, Result):-

bubbleup(List, Result1),

not(List = Result1),

sorting(Result1, Result).

% Предикат поднимания одного пузырька максимально возможно вверх

% Если длина меньше 2, то пузырек уже точно не может подняться

bubbleup(List,List):-

length(List, Length),

Length < 2.

% Если первый элемент больше второго, то он пузырек.

% Результатом будет склейка второго элемента и рекурсивного поднятия пузырька в хвосте

bubbleup([H1,H2|T], Result):-

H1 > H2,

bubbleup([H1|T], Result1),

append([H2], Result1, Result).

% Если первый элемент меньше или равен второго, то он не пузырек. Продолжаем попытку поднимания пузырька в хвосте.

bubbleup([H1,H2|T], Result):-

H1 =< H2,

bubbleup([H2|T], Result1),

append([H1], Result1, Result).

**Тестировочные кейсы:**

?- bubblesort([]).

[]

?- bubblesort([10]).

[10]

?- bubblesort([5, 1, 4, 2, 8]).

[1, 2, 4, 5, 8]

?- bubblesort([3, 2, 1]).

[1, 2, 3]

?- bubblesort([10, -5, 7, 0, 3]).

[-5, 0, 3, 7, 10]

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

[1, 2, 3, 4, 5]

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

[1, 2, 3, 4, 5]

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

[-5, -4, -3, -2, -1]

?- bubblesort([2.5, 1.2, 3.8, -1.1, 0.5]).

[-1.1, 0.5, 1.2, 2.5, 3.8]

?- bubblesort([1000, 500, 250, 125, 50]).

[50, 125, 250, 500, 1000]

?- bubblesort([1, 3, 3, 2, 1]).

[1, 1, 2, 3, 3]

?- bubblesort([7, 7, 7, 7]).

[7, 7, 7, 7]

?- bubblesort([10, -1, -1, 10]).

[-1, -1, 10, 10]

?- bubblesort([2147483647, -2147483648, 0]).

[-2147483648, 0, 2147483647]

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

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

**Сортировка вставкой**

Сортировка вставкой основана на том, что если хвост списка уже отсортирован, то достаточно поставить первый элемент списка на его место в хвосте, и весь список будет отсортирован.

**Код:**

% Сортировка вставкой

% Сортировка вставкой основана на том, что если хвост списка уже отсортирован,

% то достаточно поставить первый элемент списка на его место в хвосте,

% и весь список будет отсортирован.

% Предикат оболочка сортировки вставкой

insertionsort(List):-

sorting(List, Result),

write(Result).

% Предикат сортировки

% Базовый случай - список длиной 0 или 1 уже отсортирован

sorting(List, List):-

length(List, Length),

Length < 2.

% Рекурсивный случай: результатом сортировки является вставка головы в рекурсивно отсортированный хвост

sorting([H|T], Result):-

length([H|T], Length),

Length > 1,

sorting(T, Result1),

insertion(Result1, H, Result).

% Предикат вставки элемента на необходимую позицию в отсортированный список

% Вставка в пустой список есть список из элемента

insertion([], Element, [Element]).

% Если элемент меньше головы, то результат вставки выглядит как <Элемент|список>

insertion([H|T], Element, Insertedlist):-

Element =< H,

append([Element], [H|T], Insertedlist).

% Если элемент больше головы, то вставлять нужно после нее. То есть, результат <Голова|рекурсивная вставка в хвост>

insertion([H|T], Element, Insertedlist):-

Element > H,

insertion(T, Element, Insertedtail),

append([H], Insertedtail, Insertedlist).

**Тестировочные кейсы:**

?- insertionsort([]).

[]

?- insertionsort([7]).

[7]

?- insertionsort([9, 3, 7, 2, 8]).

[2, 3, 7, 8, 9]

?- insertionsort([12, 11, 15, 10, 9]).

[9, 10, 11, 12, 15]

?- insertionsort([42, 35, 11, 13, 29, 1]).

[1, 11, 13, 29, 35, 42]

?- insertionsort([0, -5, -10, 15, 5]).

[-10, -5, 0, 5, 15]

?- insertionsort([8, 4, 2, 1, 16, 8, 4]).

[1, 2, 4, 4, 8, 8, 16]

?- insertionsort([-100, 0, 100, -200, 200]).

[-200, -100, 0, 100, 200]

?- insertionsort([7.5, 3.2, 4.8, 1.1]).

[1.1, 3.2, 4.8, 7.5]

?- insertionsort([55, 44, 33, 22, 11]).

[11, 22, 33, 44, 55]

?- insertionsort([18, 22, 5, 13, 9, 3]).

[3, 5, 9, 13, 18, 22]

?- insertionsort([99, 98, 97, 96, 95]).

[95, 96, 97, 98, 99]

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

[-5, -4, -3, -2, -1]

?- insertionsort([1000, 500, -500, 0, -1000]).

[-1000, -500, 0, 500, 1000]

?- insertionsort([1, 2, 3, 4, 5, 6, 0]).

[0, 1, 2, 3, 4, 5, 6]

**Сортировка выбором**

Идея алгоритма *сортировки**выбором* очень проста. В списке находим минимальный элемент. Удаляем его из списка. Оставшийся список сортируем. Приписываем минимальный элемент в качестве головы к отсортированному списку. Так как этот элемент был меньше всех элементов исходного списка, он будет меньше всех элементов отсортированного списка. И, следовательно, если его поместить в голову отсортированного списка, то порядок не нарушится.

**Код:**

% Сортировка выбором

% Идея алгоритма сортировки выбором очень проста.

% В списке находим минимальный элемент. Удаляем его из списка.

% Оставшийся список сортируем. Приписываем минимальный элемент в качестве головы к отсортированному списку.

% Так как этот элемент был меньше всех элементов исходного списка, он будет меньше всех элементов отсортированного списка.

% И, следовательно, если его поместить в голову отсортированного списка, то порядок не нарушится.

% Предикат оболочка для сортировки выбором

selectionsort(List):-

sorting(List, Res),

write(Res).

% Основной предикат сортировки

% Базовый случай для пустого списка или списка из 1 элемента (уже отсортирован)

sorting(List, List):-

length(List, Length),

Length < 2.

% Рекурсивный случай

sorting(List, Result):-

length(List, Length),

Length > 1,

minimum(List, Minimum),

deleteonce(List, Minimum, Cutedlist),

sorting(Cutedlist, Sortedcutedlist),

append([Minimum], Sortedcutedlist, Result).

% Предикат получения наименьшего элемента в списке

% Минимум списка длины 2 есть минимум среди его элементов

minimum([H1,H2|T], Minimum):-

length([H1,H2|T], 2),

Minimum is min(H1,H2).

% Минимум списка есть минимум среди головы и результата рекурсии от хвоста

minimum([H|T], Minimum):-

length([H|T], Length),

Length > 2,

minimum(T, Minimum1),

Minimum is min(H, Minimum1).

% Предикат удаления одного вхождения элемента в список

% Удаление из пустого списка есть пустой список

deleteonce([], \_, []).

% Удаление, если голова не элемент, есть склейка головы и рекурсии удаления из хвоста

deleteonce([H|T], Element, Res):-

length([H|T], Length),

Length >= 1,

not(H = Element),

deleteonce(T, Element, Res1),

append([H], Res1, Res).

% Удаление, если голова - элемент, есть хвост

deleteonce([H|T], Element, Res):-

length([H|T], Length),

Length >= 1,

H = Element,

Res = T.

**Тестировочные кейсы:**

?- selectionsort([]).

[]

?- selectionsort([1]).

[1]

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

[1, 2, 3, 4, 5]

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

[1, 2, 3, 4, 5]

?- selectionsort([3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]).

[1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]

?- selectionsort([0, -1, -2, 10, 7]).

[-2, -1, 0, 7, 10]

?- selectionsort([10, 10, 10, 10]).

[10, 10, 10, 10]

?- selectionsort([0, -1, -1, 0]).

[-1, -1, 0, 0]

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

[-5, -4, -3, -2, -1]

?- selectionsort([1000, 500, -100, -1000, 0]).

[-1000, -100, 0, 500, 1000]

?- selectionsort([1.5, 2.3, -1.2, 0.7]).

[-1.2, 0.7, 1.5, 2.3]

?- selectionsort([5, 1, 4, 2, 8]).

[1, 2, 4, 5, 8]

?- selectionsort([1, 2, 3, -999999999]).

[-999999999, 1, 2, 3]

?- selectionsort([2147483647, -2147483648, 0]).

[-2147483648, 0, 2147483647]

?- selectionsort([3, 3, 3, 2, 2, 1, 1, 1]).

[1, 1, 1, 2, 2, 3, 3, 3]

**Быстрая сортировка**

Идея метода следующая. Выбирается некоторый «барьерный» элемент, относительно которого мы разбиваем исходный список на два подсписка. В один мы помещаем элементы, меньшие барьерного элемента, во второй — большие либо равные. Каждый из этих списков мы сортируем тем же способом, после чего приписываем к списку тех элементов, которые меньше барьерного, вначале сам барьерный элемент, а затем — список элементов не меньших барьерного. В итоге получаем список, состоящий из элементов, стоящих в правильном порядке.

**КОД:**

% Оболочка для вывода

quicksort(List):-

sorting(List, Result),

write(Result).

% Базовый случай для пустого списка

sorting([], []).

% Базовый списко для списка из одного элемента

sorting(List, List):-

length(List, 1).

% Рекурсивный случай

sorting(List, Result):-

length(List, Length),

Length>1,

getpivotnumber(List, Pivotnumber),

getpivot(List, Pivot),

lesssublist(List, Pivot, Sublist1),

moresublist(List, Pivot, Pivotnumber, Sublist2),

sorting(Sublist1, Sortedsublist1),

sorting(Sublist2, Sortedsublist2),

append(Sortedsublist1, [Pivot], Result1),

append(Result1, Sortedsublist2, Result).

% Получение номера опорного элемента

getpivotnumber(List, Pivotnumber):-

length(List, Length),

Pivotnumber is Length div 2.

% Получение опорного элемента для списка нечетной длины (строго медианный)

getpivot(List, Pivot):-

length(List, Length),

1 is Length mod 2,

append(Left, [Mid|Right], List),

length(Left, Leftlength),

length(Right, Leftlength),

Pivot is Mid.

% Получение опорного элемента для списка четной длины (правый из медианных)

getpivot(List, Pivot):-

length(List, Length),

0 is Length mod 2,

append(Left, [Mid|Right], List),

length(Left, Leftlength),

length(Right, Rightlength),

Leftlength is Rightlength + 1,

Pivot is Mid.

% Предикат получения элементов меньше опорного

% Список элементов из пустого списка, которые меньше опорного, равны пустому списку

lesssublist([], \_, []).

% Список элементов из списка, в котором голова больше опорного, меньших опорного есть аналогичный список из хвоста

lesssublist([H|T], Pivot, Sublist):-

H >= Pivot,

lesssublist(T, Pivot, Sublist).

% Список элементов из списка, в котором голова меньше опорного, меньших опорного, есть склейка головы и рекурсии от хвоста

lesssublist([H|T], Pivot, Sublist):-

H < Pivot,

lesssublist(T, Pivot, Sublist1),

append([H], Sublist1, Sublist).

% Предикат получения элементов больше опорного

% Из пустого списка больше опорного - пустой список

moresublist([], \_, \_, []).

% Если голова меньше, то элементы >= опорного аналогичны рекурсии от хвоста

moresublist([H|T], Pivot, Pivotnumber, Sublist):-

H < Pivot,

Pivotnumber1 is Pivotnumber - 1,

moresublist(T, Pivot, Pivotnumber1, Sublist).

% Если достигнут опорный (счетчик опорного снизился до нуля), то элементы >= опорного аналогичны рекурсии от хвоста

moresublist([H|T], Pivot, Pivotnumber, Sublist):-

H >= Pivot,

Pivotnumber is 0,

Pivotnumber1 is Pivotnumber - 1,

moresublist(T, Pivot, Pivotnumber1, Sublist).

% Если элемент больше либо равен опорному и это не ПОЗИЦИЯ опорного элемента - то добавляем его в результат

moresublist([H|T], Pivot, Pivotnumber, Sublist):-

H >= Pivot,

not(Pivotnumber is 0),

Pivotnumber1 is Pivotnumber - 1,

moresublist(T, Pivot, Pivotnumber1, Sublist1),

append([H], Sublist1, Sublist).

**Тестовые кейсы:**

?- quicksort([]).

[]

?- quicksort([42]).

[42]

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

[1, 2, 3, 4, 5]

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

[1, 2, 3, 4, 5]

?- quicksort([7, 7, 7, 7, 7]).

[7, 7, 7, 7, 7]

?- quicksort([2, 3]).

[2, 3]

?- quicksort([3, 2]).

[2, 3]

?- quicksort([5, 1, 5, 3, 5]).

[1, 3, 5, 5, 5]

?- quicksort([-3, -1, -7, 0, 2]).

[-7, -3, -1, 0, 2]

?- quicksort([0, 0, 0, 0]).

[0, 0, 0, 0]

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

[-5, -2, 1, 3, 4]

?- quicksort([7, 2, 5, 3, 1]).

[1, 2, 3, 5, 7]

?- quicksort([8, 3, 6, 1]).

[1, 3, 6, 8]

?- quicksort([12, 4, 5, 2, 8, 1, 7, 6, 3, 11, 9, 10]).

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

?- quicksort([3.1, 1.5, 2.8, 3.3, 1.1]).

[1.1, 1.5, 2.8, 3.1, 3.3]

**Сортировка слияниями**

Идея этого метода заключается в следующем. Разобьем список, который нужно упорядочить, на два подсписка. Упорядочим каждый из них этим же методом, после чего сольем упорядоченные подсписки обратно в один общий список.

**Код:**

% Сортировка слияниями

% Идея этого метода заключается в следующем.

% Разобьем список, который нужно упорядочить, на два подсписка.

% Упорядочим каждый из них этим же методом, после чего сольем упорядоченные подсписки обратно в один общий список.

% Функция оболочка для корректного вывода

mergesort(List):-

sorting(List, Res),

write(Res).

% Базовый случай: пустой список является отсортированным

sorting([], []).

% Базовый случай: список из 1 элемента является отсортированным

sorting(List, List):-

length(List,1).

% Рекурсивный случай: делим список пополам, сортируем половины, соединяем

sorting(List, Sortedlist):-

length(List, Listlength),

Listlength > 1,

spliting(List, List1, List2),

sorting(List1, Sortedlist1),

sorting(List2, Sortedlist2),

merging(Sortedlist1, Sortedlist2, Sortedlist).

% Предикат деления списка пополам

% Cлучай четного массива

spliting(List, List1, List2):-

append(List1, List2, List),

length(List, Len), 0 is Len mod 2,

length(List1, X),

length(List2, X).

% Cлучай нечетного массива

spliting(List, List1, List2):-

append(List1, List2, List),

length(List, Len), 1 is Len mod 2,

length(List1, X1),

length(List2, X2),

X1 is X2 + 1.

% Предикат слияния двух отсортированных списков в третий

% Первый список пуст

merging([], List2, List2).

% Второй список пуст

merging(List1, [], List1).

% Списки непусты, голова первого больше головы второго

merging([H1|T1], [H2|T2], List):-

H1 >= H2,

merging([H1|T1], T2, Taillist),

append([H2], Taillist, List).

% Списки непусты, голова первого меньше головы второго

merging([H1|T1], [H2|T2], List):-

H1 < H2,

merging(T1, [H2|T2], Taillist),

append([H1], Taillist, List).

**Тестировочные кейсы:**

?- mergesort([]).

[]

?- mergesort([5]).

[5]

?- mergesort([1, 2]).

[1, 2]

?- mergesort([2, 1]).

[1, 2]

?- mergesort([3, 1, 2]).

[1, 2, 3]

?- mergesort([4, 1, 4, 2, 1]).

[1, 1, 2, 4, 4]

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

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

?- mergesort([7, 6, 5, 4, 3, 2, 1]).

[1, 2, 3, 4, 5, 6, 7]

?- mergesort([2, 2, 2, 2, 2]).

[2, 2, 2, 2, 2]

?- mergesort([3, 0, 2, 1]).

[0, 1, 2, 3]

?- mergesort([-3, -1, -2, 0]).

[-3, -2, -1, 0]

?- mergesort([-1, 3, -2, 2, 0]).

[-2, -1, 0, 2, 3]

?- mergesort([1000, 2000, -3000, 500, 0]).

[-3000, 0, 500, 1000, 2000]

?- mergesort([0]).

[0]

?- mergesort([4, 2, 0, 3, 1]).

[0, 1, 2, 3, 4]

?- mergesort([10, 9, 8, 7, 6, 5, 4, 3, 2, 1]).

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

?- mergesort([-5, -3, 0, -4, -1]).

[-5, -4, -3, -1, 0]

?- mergesort([8, 3, 7, 1, 5, 6, 2, 4]).

[1, 2, 3, 4, 5, 6, 7, 8]

?- mergesort([5, 5, 5, 5, 5, 5, 5, 5, 5]).

[5, 5, 5, 5, 5, 5, 5, 5, 5]

?- mergesort([15, -8, 0, 7, 1, 100, -3, -200, 50]).

[-200, -8, -3, 0, 1, 7, 15, 50, 100]