**День 1:**

Слово – конечный непустой список букв фиксированного конечного алфавита. Текст – конечный непустой список слов. Построить логическую программу, которая …

1. … для заданного текста L вычисляет одно из слов X, имеющее максимальную длинну среди всех тех слов текста L, которые встречаются в нем наиболее часто. Запрос к программе должен иметь вид ?C(L, X).

Решение:

Сначала будет найден список наиболее частых слов текста L. Потом из него будет выбрано слово минимальной длины. Для нахождения списка наиболее частых слов текста L будет сначала сформирован (бесповторный) список его слов, затем для этого списка слов будет сформирован список частот его встречаемости, будет найдена максимальная частота встречаемости, потом будет найден список слов, частота которых равна максимальной. Заметим, что в этой задаче, вообще говоря, не обязательно формировать список слов исходного текста (вместо него можно использовать сам исходный текст) – это ни на что, кроме объема вычислений, не повлияет. Однако в некоторых задачах может понадобиться бесповторный список слов (кроме того, может просто понадобиться сделать список бесповторным), так что эта процедура будет здесь разобрана.

Кроме того, потребуются некоторые вспомогательные функции. А именно, len(x, n), которая определяет длину n слова x, eq(x, y), которая проверяет два слова на равенство, neq(x, y), которая проверяет два слова на неравенство, а также el(L, x) и nel(L, x), которые имеют успешное вычисление только в том случае, когда x является (соответственно, не является) элементом списка L.

Длина слова:

?len(x, n):

len(nil, 0) <-;

len(a.x, n) <- len(x, m), n is m + 1;

Равенство двух слов:

?eq(x, y):

eq(nil, nil) <-;

eq(a.x, a.y) <- eq(x, y);

Неравенство двух слов:

?neq(x, y):

neq(a.x, b.y) <- a <> b;

neq(a.x, a.y) <- neq(x, y);

x является элементом списка L:

?el(L, x):

el(x.L, x) <-;

el(y.L, x) <- el(L, x);

x не является элементом списка L:

?nel(L, x):

nel(nil, x) <-;

nel(y.L, x) <- y <> x, nel(L, x);

Настало время более интересных функций. Составим список L1 всех слов текста L. Запрос ?words(L, L1):

words(nil, nil) <-;

words(x.L, x.L1) <- nel(L, x), !, words(L, L1);

words(x.L, L1) <- words(L, L1);

Заметим, что эта программа просто составляет бесповторный список элементов списка L. Можно ее использовать и для этих целей. Правда, она не сохраняет порядок первых вхождений (поскольку удобнее оставлять последнее вхождение), однако в этой задаче, как и во многих других, это не важно.

Для нашей задачи требуется вычислить частоту n вхождения слова x в текст L, то есть сколько данное слово встретилось в тексте. Запрос ?freq(L, x, n):

freq(nil, x, 0) <-;

freq(x.L, x, n) <- freq(L, x, m), n is m + 1;

freq(y.L, x, n) <- neq(x, y), freq(L, x, n);

Теперь по списку слов L1 вычислим список количеств вхождений Lf этих слов в текст L. Запрос ?flist(L, L1, Lf):

flist(L, nil, nil) <-;

flist(L, x.L1, n.Lf) <- freq(L, x, n), flist(L, L1, Lf);

Далее, вычислим максимум n элементов списка натуральных чисел L. Эта программа нам потребуется для вычисления максимальной частоты вхождения элемента в список. Запрос ?maxl(L, n):

maxl(n.nil, n) <-;

maxl(n.L, n) <- maxl(L, m), n >= m;

maxl(n.L, m) <- maxl(L, m), m > n;

Теперь по списку слов L и списку частот Lf найдем список слов L1, которые встречаются с определенной частотой n. Если в качестве n подставить максимальную частоту, то мы получим список наиболее часто встречающихся слов. Запрос freqeqn(L, Lf, L1, n):

freqeqn(nil, nil, nil, n) <-;

freqeqn(x.L, n.Lf, x.L1, n) <- freqeqn(L, Lf, L, n);

freqeqn(x.L, m.Lf, L1, n) <- m <> n, !, freqeqn(L, Lf, L1, n);

Теперь составим программу нахождения наиболее часто встречающихся слов Lw (список таких слов) в тексте L. Запрос ?mfwords(L, Lw):

mfwords(L, Lw) <- words(L, L1), flist(L, L1, Lf), maxl(Lf, n), freqeqn(L1, Lf, Lw, n);

Остается только написать программу, которая выбирает из списка слов L самое длинное (одно из самых длинных) слово x. Запрос ?longest(L, x):

longest(x.nil, x) <-;

longest(x.L, x) <- longest(L, y), len(x, n), len(y, m), n >= m;

longest(x.L, y) <- longest(L, y), len(x, n), len(y, m), n < m;

Вся программа для решения задачи имеет вид:

с(L, x) <- mfwords(L, Lw), longest(Lw, x);

1. … для заданных текстов L1 и L2 вычисляет одно из слов X, содержащееся в тексте L1 и имееющее своим префиксом одно из слов максимальной длины из текста L2. Запрос к программе должен иметь вид ?D(L1, L2, X).

Решение:

Сначала требуется найти список самых длинных слов текста L2. Он вычисляется аналогично предыдущей задаче (только вместо частоты используется длина, что позволяет все упростить). Функции, написанные в предыдущей задаче, будут использоваться без дополнительных комментариев. Кроме того, потребуется написать функции для работы с префиксами.

Найдем максимальную длину n слова из списка L. Запрос ?maxlen(L, n):

maxlen(x.nil, n) <- len(x, n);

maxlen(x.L, n) <- len(x, n), maxlen(L, m), n >= m;

maxlen(x.L, n) <- len(x, m), maxlen(L, n), n > m;

Теперь найдем список слов L1, который состоит из слов списка L, имеющих длину n. Запрос ?leneqn(L, L1, n):

leneqn(nil, nil, n) <-;

leneqn(x.L, x.L1, n) <- len(x, n), !, leneqn(L, L1, n);

leneqn(x.L, L1, n) <- len(x, m), m <> n, leneqn(L, L1, n);

Осталось найти все слова максимальной длины списка L. Запрос ?mlwords(L, Lw):

mlwords(L, Lw) <- words(L, L1), maxlen(L1, n), leneqn(L1, Lw, n);

Теперь необходимо решить ту часть задачи, которая связана с префиксами. Для начала напишем программу, проверяющую, является ли слово x префиксом слова y. Запрос ?prefix(x, y):

prefix(nil, y) <-;

prefix(a.x, a.y) <- prefix(x, y);

Теперь нужно написать программу, которая проверяет, содержит ли список слов L слово, которое является префиксом слова x. Запрос ?prefixamong(L, x):

prefixamong(y.L, x) <- prefix(y, x), !;

prefixamong(y.L, x) <- prefixamong(L, x);

Переходим к нахождению любого слова x из списка L, которое имеет хотя бы один префикс среди слов списка L1 (реально он найдет первое такое слово). Запрос ?wpa(L, L1, x):

wpa(x.L, L1, x) <- prefixamong(L1, x), !;

wpa(y.L, L1, x) <- wpa(L, L1, x);

Соответственно, решение задачи будет выглядеть так:

d(L1, L2, x) <- mlwords(L2, L), wpa(L1, L, x);

1. … для заданных текстов L1 и L2 вычисляет суммарное количество X вхождений в текст L1 всех наиболее часто встречающихся слов из текста L2. Запрос к программе должен иметь вид ?E(L1, L2, X).

Решение:

Сначала вычисляется список наиболее часто встречающихся слов в тексте L2. Это уже было сделано в первой задаче, здесь будет просто вызов функции. Далее, этот список разбирается поэлементно, и для каждого слова считается количество его вхождений в список L1. Функция, которая считает количество вхождений, также была написана в первой задаче. Здесь же остается написать функцию, которая по списку слов L (бесповторному, вот где нужен бесповторный список наиболее часто встречающихся слов!) и тексту L1 посчитает суммарное количество вхождений s слов из L в текст L1. Запрос ?sumfreq(L, L1, s):

sumfreq(nil, L1, 0) <-;

sumfreq(x.L, L1, s) <- freq(L1, x, q), sumfreq(L, L1, p), s is p + q;

Теперь пишем ту программу, которая требуется в условии задачи:

e(L1, L2, x) <- mfwords(L2, L), sumfreq(L, L1, x);

1. … для заданных текстов L1 и L2 вычисляет одно из слов, содержащееся в тексте L2 такое, что частота его вхождения L2 больше частоты его вхождения в L1. Запрос к программе должен иметь вид ?E(L1, L2, X).

Решение:

Для начала напишем вспомогательную функцию, у которой будет 4 аргумента и которая будет находить слово x в списке L, частота вхождения которого в L2 больше, чем частота его вхождения в L1. Запрос ?f(L, L1, L2, x):

f(x.L, L1, L2, x) <- freq(L1, x, n1), freq(L2, x, n2), n2 > n1, !;

f(x.L, L1, L2, y) <- f(L, L1, L2, y);

Остается только в качестве списка слов использовать текст L2, и мы получим решение задачи:

e(L1, L2, x) <- f(L2, L1, L2, x);

**День 2:**

Построить логическую программу, которая …

1. … для двух заданных конечных последовательностей натуральных чисел, представленных списками L1 и L2, вычисляет наименьшее из чисел X, которые являются элементами L1 и превосходят любой элемент последовательности L2. Запрос к программе должен должен иметь вид ?G(L1,L2,X).

Решение:

Сначала вычисляется максимальный элемент списка L2, затем от списка L1 оставляются только те элементы, которые превосходят найденный максимум. После этого из них выбирается минимальный элемент.

Нахождение минимального и максимального элемента списка (запросы minl(L, x) и maxl(L, x) соответственно) рассматривалось ранее и поэтому сейчас рассмотрено не будет. Рассмотрим второй этап – выбор только тех элементов, которые превосходят максимум. Запрос listgt(L1, L2, n) формирует список L2 тех элементов списка L1, которые превосходят n. Программа:

?listgt(L1, L2, n):

listgt(nil, nil, n) <-;

listgt(x.L1, x.L2, n) <- x > n, !, listgt(L1, L2, n);

listgt(x.L1, L2, n) <- x <= n, !, listgt(L1, L2, n);

Теперь собственно решение задачи:

g(L1, L2, x) <- maxl(L2, m), listgt(L1, L, m), minl(L, x);

1. … для заданной конечной последовательности натуральных чисел L вычисляет сумму X всех тех элементов L, квадрат которых превосходит среднее арифметическое всех элементов последовательности L. Запрос к программе должен иметь вид ?G(L,X).

Решение:

Сначала вычисляется среднее арифметическое последовательности, потом вычисляется сумма требуемых элементов. При этом элемент включается или не включается в сумму в зависимости от того, удовлетворяет ли он заданному условию или нет.

Программы для вычисления суммы и длины списка уже встречались ранее. Здесь я напишу только программу для среднего арифметического и для решения задачи.

Вычисление среднего арифметического:

?avg(L, a):

avg(L, a) <- sum(L, s), len(L, n), a is s / n;

Вычисление суммы s элементов L1, квадрат которых превосходит величину a (если таких элементов нет, то их сумма считается равной нулю):

?sumsga(L, s, a):

sumsga(nil, 0) <-;

sumsga(x.L, s) <-

1. …для двух заданных конечных последовательностей натуральных чисел, представленных списками L1 и L2, вычисляет бесповторный список L3 из всех элементов последовательности L1, которые не превосходят наименьшего элемента последовательности L2. Запрос к программе должен иметь вид ?G(L1, L2, L3).

Решение:

Сначала находится наименьший элемент последовательности L2. Затем определяются те элементы последовательности L1, которые не превосходят его, и из них составляется бесповторный список. Поскольку не требуется сохранять порядок вхождения, удобнее оставлять последний элемент. Для решения задачи будут использованы программы minl(L, x) нахождения минимального элемента списка и nel(L, x), которая имеет успешное вычисление только в том случае, когда x – не элемент списка L.

Программа, которая по списку L1 и числу n формирует бесповторный список L элементов из L1, которые не превосходят n.

?numsltn(L1, L, n):

numsltn(nil, nil, n) <-;

numsltn(x.L1, x.L, n) <- x <= n, nel(L, x), !, numsltn(L1, L, n);

numsltn(x.L1, L, n) <- numsltn(L1, L, n);

Собственно решение задачи:

g(L1, L2, L3) <- minl(L2, n), numsltn(L1, L3, n);