###### ПРАКТИКУМ

# Программирование символьных вычислений

## Цель работы

Изучить базовые функциональные средства языка программирования Лисп, ориентированного на символьные вычисления и задачи искусственного интеллекта. Освоить основные приемы обработки сложных структур символьных данных, разработать и реализовать лисп-программу для определенного варианта преобразования символьных выражений.

### Общие требования

1. В качестве домашней работы необходимо запрограммировать решение ***исходной задачи*** по выбранному ***основному варианту*** (см. ниже).
2. В ходе разработки программы необходимо продумать и затем использовать ***внутреннее представление*** символьных структур, адекватное проводимым символьным вычислениям.
3. В ходе сдачи разработанной программы (на занятии в компьютерном классе) потребуется внести изменения в программу, расширяющие ее функциональность до некоторого ***окончательного варианта***. При выставлении оценки учитывается не только качество итоговой программы, но и способность студента ее модифицировать. Учитывается также обоснованность внутреннего представления обрабатываемых символьных структур и эффективность их преобразования.
4. Для отладки программы необходимо составить ***набор тестов***, демонстрирующий ее работоспособность на разнообразных входных данных.
5. Текст **отчета** (примерно 2-3 стр.) должен включать следующее:
6. ФИО, номер группы, год.
7. Описание проделанного варианта задания (***окончательный вариант***).
8. Текст лисп-программы.
9. Описание лисп-программы (допускается описание в виде ***комментариев*** *к основным функциям программы*):

* представление исходных данных программы и вид результирующих данных;
* внутреннее представление обрабатываемых символьных выражений;
* спецификации основных функций программы.

1. Набор ***тестов***, использованных при отладке программы.

**Методические указания**

Для упрощения разработки лисп-программ предполагается, что исходные сложные символьные объекты (полиномы, формулы, уравнения, списки дорог или проходов в лабиринте) заключены в объемлющие круглые скобки – тем самым они становятся лисповскими выражениями, и облегчается их ввод. В дальнейшем при проведении преобразований целесообразно эти скобки сохранить и убрать их только при выводе результата.

## 1. Интерпретатор выражений в чистом лямбда-исчислении (средняя сложность - максимальная оценка ХОР, можно усложнить до ОТЛ)

Рассматриваются лямбда-выражения, в которых используются только переменные, скобки, операции абстракции и аппликации. В качестве имен переменных берутся латинские буквы, операция абстракции записывается как слэш **/**, а вместо точки используется знак **>**, знаки операции абстракции и переменные абстракции разделяются пробелами, а само выражение заключается в круглые скобки (чтобы сделать его лисповским списком и тем самым облегчить его ввод и обработку), например:

((/ x > / y > x y w)(/ z > z)).

Требуется реализовать вычисление (редукцию) заданного лямбда-выражения, используя правило бета-редукции и приводя выражение к нормальной форме на основе нормального порядка редукции. Результатом редукции лямбда-выражения, приведенного выше, будет выражение (/ y > y w).

В процессе редуцирования может понадобиться альфа-преобразование в случае, когда есть конфликт имён, например, при редуцировании такого выражения:  
((/ x > / y > + x y) y 10).

Реализация интерпретатора, умеющего работать только с выражениями, в которых нет конфликтов имён и нет необходимости проводить альфа-преобразование – на ХОР. Дополнительная реализация альфа преобразования (переименования переменных) в процессе редуцирования в случае необходимости – на ОТЛ.

## 2. Проверка логических формул (простое - максимальная оценка УДОВЛ)

Обрабатываются формулы алгебры логики, в которых используются логические константы true и false, логические переменные и логические операции: отрицания (¬), конъюнкции (&), дизъюнкции (∨), импликации (⊃), а также круглые скобки. В качестве имен переменных могут быть взяты произвольные латинские буквы. К примеру, логической формулой является запись a ⊃ ¬(b & ¬c ∨ d) ≡ (c & g).

Порядок выполнения операций в формуле алгебры логики определяется согласно общепринятому приоритету логических операций и записанным скобкам. Логическая операция ⊃ имеет меньший приоритет, чем логическое отрицание, конъюнкция и дизъюнкция.

В логической формуле могут быть опущены незначащие скобки (пара скобок считается незначащей, если после ее удаления формула остается эквивалентной исходной).

*Дизъюнктивной нормальной формой* (ДНФ) называется логическая формула, представляющая собой дизъюнкцию из нескольких формул – элементарных конъюнкций. Элементарной конъюнкцией является конъюнкция нескольких переменных или их отрицаний.

ДНФ называется *правильной*, если каждая ее элементарная конъюнкция не содержит повторных вхождений одной переменной. Правильная ДНФ называется *совершенной*, если каждая элементарная конъюнкция содержит вхождения всех переменных булевской функции.

Аналогично определяются понятия *конъюнктивной нормальной формы* (КНФ), правильной и совершенной КНФ. Незначащие скобки в нормальных формах не ставятся.

Исходная задача: проверить, является ли заданная логическая формула ДНФ. Исходная формула заключена в объемлющие скобки, т.е. задается в виде лисповского списка. Все составные элементы формулы (константы, переменные, операции) записываются через пробел, например: для заданной формулы (a & b ∨  b & ¬ c ∨  true)  результатом проверки будет nil.

## 3. Преобразование произвольной формулы алгебры логики в дизъюнктивную нормальную форму (высокая сложность - максимальная оценка ОТЛ)

Преобразовать в *дизъюнктивную нормальную форму* произвольную формулу алгебры логики, в которой используются логические константы, однобуквенные переменные и операции логического отрицания, конъюнкции, дизъюнкции и импликации. Результирующая дизъюнктивная нормальная форма должна быть *правильной*, т.е. каждая ее элементарная конъюнкция не должна содержать повторных вхождений одной переменной. Полученная формула не должна содержать избыточных скобок.

Например, (¬ a ∨ d ∨ false) ⊃ ¬(b & ¬ c ∨ d) преобразуется к виду (a & ¬ d) ∨ (¬ b & ¬ d) ∨ (c & d) или a & ¬ d ∨ ¬ b & ¬ d ∨ c & d .

Порядок выполнения операций в формуле алгебры логики определяется согласно общепринятому приоритету логических операций и записанным скобкам.

При записи исходной логической формулы знаки логических операций, символы констант и переменных разделяются хотя бы одним пробелом (если между ними не стоит другой разделитель – круглая скобка). Сама же формула заключается в объемлющие скобки – с тем, чтобы сделать ее лисповским выражением и тем самым облегчить ее ввод. В дальнейшем при проведении преобразований целесообразно эти скобки сохранить и убрать их только при выводе результирующей дизъюнктивной нормальной формы.

Решение задачи с помощью таблицы истинности - УДОВЛ. Решение задачи с помощью символьных преобразований - ОТЛИЧНО

## 4. Преобразование дизъюнктивной нормальной формы в конъюнктивную нормальную форму (средняя высокая сложность - максимальная оценка ОТЛ)

По заданной дизъюнктивной нормальной форме некоторой булевской функции построить ее совершенную конъюнктивную нормальную форму, а также сокращенную конъюнктивную нормальную форму. Как исходная дизъюнктивная нормальная форма, так и результирующие конъюнктивные нормальные формы предполагаются *правильными*, т.е. все переменные, встречающиеся в каждой элементарной конъюнкции/дизъюнкции, различны. В отличие от сокращенной, в *совершенной* конъюнктивной нормальной форме каждая элементарная дизъюнкция содержит вхождения всех переменных булевской функции. Полученные конъюнктивные нормальные формы не должны содержать избыточных скобок.

Например, для дизъюнктивной формы a&b&¬c ∨  a&b&d  ∨ b&c&d получается сокращенная конъюнктивная нормальная форма b &(a ∨ ¬b ∨ c)&(¬c ∨ d) .

Порядок выполнения логических операций определяется согласно общепринятому приоритету операций отрицания, конъюнкции и дизъюнкции и записанным скобкам.

На вход программы построения конъюнктивных форм подается лисповский список, в котором записаны через пробел символы переменных и знаки логических операций. В ходе этого преобразования необходимо проверить правильность записи исходной дизъюнктивной формы. Решение задачи требуется проводить с помощью символьных преобразований.

## 5. По регулярной грамматике построить соответствующий детерминированный конечный автомат (высокая сложность - максимальная оценка ОТЛ)

По заданному тексту *регулярной* (леволинейной или праволинейной) грамматики построить соответствующий детерминированный конечный автомат (*диаграмму состояний*). Автомат реализует распознавание слов языка, определенного грамматикой.

Грамматика задается как конечный набор правил, состоящих из левой и правой частей, например: T=aN|bN. Нетерминальные символы грамматики записываются большими латинскими буквами, а терминальные – маленькими. Начальный нетерминальный символ грамматики обозначается буквой S.

Построенный автомат представляет собой ориентированный и помеченный граф: вершины графа соответствуют состояниям автомата и помечены нетерминальными символами грамматики; в множество вершин входит начальное состояние H и заключительное состояние S. Ребра графа соответствуют переходам между состояниями автомата и помечены терминальными символами грамматики. Граф записывается в виде списка входящих в него ребер, каждое ребро представлено трехэлементным списком вида (метка\_вершины метка\_ребра метка\_вершины).

При записи грамматики нужно заключить в круглые скобки каждое ее правило, а также левые и правые части правил – тем самым получаем лисповский список правил. Для запаси терминальных символов и символа | использовать символы Лиспа, например:

(((S) = (#\a N #\| #\b N)) ((N) = ( #\c N #\| #\d))).

В этом списке терминальные и нетерминальные символы разделены пробелами.

## 6. По конечному автомату восстановить текст соответствующей регулярной грамматики (высокая сложность - максимальная оценка ОТЛ)

По заданному конечному автомату восстановить текст соответствующей *регулярной* леволинейной (или праволинейной) грамматики, включающий алфавиты (множества) терминальных и нетерминальных символов и набор правил грамматики, записанных в общепринятой форме.

Конечный автомат представлен в виде ориентированного и помеченного графа, т.е. *диаграммы состояний*. Вершины этого графа соответствуют состояниям автомата и помечены нетерминальными символами грамматики; в множество вершин входит начальное состояние H и заключительное состояние S. Ребра графа соответствуют переходам между состояниями автомата и помечены терминальными символами грамматики. Граф записан как лисповский список входящих в него ребер, каждое ребро представлено трехэлементным списком вида (метка\_вершины метка\_ребра метка\_вершины).

Исходный конечный автомат может быть как детерминированным, так и недетерминированным, в последнем случае необходимо, кроме грамматики, построить эквивалентный ему детерминированный автомат – т.е. детерминированный автомат, определяющий тот же самый язык.

## 7. Анализ и упрощение КС-грамматики (средняя сложность - максимальная оценка ХОР)

По заданной произвольной контекстно-свободной грамматике построить эквивалентную ей *приведенную* неукорачивающую грамматику, удалив недостижимые и бесплодные нетерминальные символы грамматики, а также правила с пустой правой частью. Грамматика задаётся как конечный набор правил вида

*нетерминальный символ = цепочка терминальных и нетерминальных символов*

|*цепочка терминальных и нетерминальных символов*

| ...

состоящих в общем случае из нескольких альтернатив, например: T = А N b | xy C | $ . Альтернативы в правых частях правил могут быть пустыми и обозначаются символом $. Нетерминальные символы грамматики записываются заглавными (большими) латинскими буквами, а терминальные – строчными (маленькими). Начальный символ грамматики обозначается буквой S.

В ходе построения приведенной грамматики следует выводить на печать каждый совершаемый шаг с указанием исключаемых символов (недостижимых или бесплодных), удаляемых правил с пустой правой частью, а также грамматики, получающейся на этом шаге (возможно, во внутреннем ее представлении). Итоговая приведенная грамматика должна быть выведена в общепринятом представлении (см. пример выше). В случае, когда исходная грамматика уже является приведенной, достаточно вывести только сообщение об этом.

При записи исходной грамматики нужно заключить в круглые скобки каждое её правило и составить из них лисповский список, в котором все символы грамматики разделены пробелами, а знак **|** и строчные буквы записаны согласно правилам соответственно для особых и обычных символов языка Лисп. Например:

((T = A N #\b \| #\x #\y C \| $ ) (N = #\c N \| #\d)).

## 8. Решение линейного уравнения (простое - максимальная оценка УДОВЛ)

Рассматриваются линейные уравнения от одной переменной, именем которой служит однобуквенный идентификатор. Обе части уравнения представляют собой сумму/разность нескольких слагаемых, каждое слагаемое – произведение из целых чисел и переменной, причем знак операции умножения в записи уравнения опускается, например: Y – 7Y2 +5 = 16Y –1.

Уравнение называется *приведенным*, если его левая часть содержит только одну переменную с некоторым числовым множителем, а правая – число, например, 3X = 9.

Исходная задача: преобразовать заданное уравнение в эквивалентное приведенное уравнение. Уравнение задано в виде лисповского списка, в котором все составляющие уравнения (алгебраические операции, переменная, знак равенства и числа) записываются через пробел, например: (Y – 7 Y 2 + 5 = 16 Y – 1).

**9. Решение системы линейных уравнений методом исключений (высокая сложность - максимальная оценка ОТЛ)**

Дана система M линейных уравнений с N неизвестными, N и M ≥ 1, например (M=2,N=2):

Необходимо решить заданную систему линейных уравнений *методом исключения переменных*. Решение системы включает в себя следующие шаги преобразований уравнений:

1. в одном из уравнений некоторая переменная выражается через другие переменные, например: X =(4+Y)/2 = 4/2 + Y/2 ;
2. полученное для этой переменной выражение подставляется в остальные уравнения системы вместо всех вхождений указанной переменной.

В результате выполнения этих шагов получается формула для вычисления значения одной переменной системы по значениям других переменных (шаг 1), а также новая равносильная система уравнений, содержащая на одно уравнение и на одну переменную меньше, чем предыдущая. Далее шаги 1-2 повторяются для новой системы уравнений, и так продолжается до тех пор, пока не останется одно уравнение или пока не обнаружится противоречие.

Противоречие (неверное равенство) возникает, когда система не имеет решения (например, в ходе преобразований получено уравнение 6‑4=0), и в этом случае в качестве ответа должно быть выдано соответствующее сообщение. Если же в процессе преобразований одно из уравнений превратилось в тождество (например, 0=0), то исходная система уравнений была избыточна, и это уравнение можно исключить из решаемой системы.

В случае, когда в ходе преобразований системы получено одно уравнение с одной переменной, то из этого уравнения вычисляется значение этой переменной. Значения остальных переменных вычисляются по полученным в ходе преобразований формулам, и итоговое решение системы уравнений выводится в виде строк вида *имя\_переменной*=*значение*. Для системы уравнений, приведённой выше в качестве примера, ответом будет: X=1 и Y=-2.

Если же в результате преобразований остаётся одно уравнение, и оно содержит более одной переменной (например, Y-Z-2=0), то это означает, что исходная система имеет бесконечное множество решений. В этом случае необходимо вывести на печать имена *свободных* переменных системы (которые могут принимать любые значения) и формулы для вычисления остальных переменных по значениям свободных. Например, для системы уравнений:

Результат может быть выдан в виде: X=-Z-2, Y=Z+2, Z – свободная переменная.

Исходная система уравнений задается как список строк, каждая строка соответствует уравнению и записывается как список входящих в уравнение компонент (знаков операций, чисел и переменных), которые разделяются пробелами, например:

((2 \* X - Y = 4) (X + 3 \* Y = -5)) .

## 10. Построение префиксного дерева (средняя сложность - максимальная оценка ХОР)

Префиксное дерево (бор, *tier*) – структура для оптимального хранения словарей и ассоциативных списков и работы с ними. Требуется построить префиксное дерево для заданного набора слов (не менее 20), записанных в латинице, и реализовать функции работы с этим деревом.

В префиксном дереве дуги помечаются буквами слов (по одной букве на дуге), и у любого узла дерева буквы на исходящих из него дугах попарно различны. Слова хранятся как последовательности букв на ветвях префиксного дерева, идущих от корня, при этом склеиваются общие начала слов – им соответствует одна и также последовательность дуг (как например, для слов *ten* и *tea* две общие дуги с буквами *t* и *e*). Для любого хранимого в дереве слова {\displaystyle S} существует путь из корня дерева в конечный его узел (лист дерева) или же в некоторый промежуточный узел (если это слово вкладывается в другое), такой что конкатенация букв на этом пути образует это слово. В случаях вложения слова в другое (например: *use* и *useful*) конец более короткого слова должен быть помечен специальной меткой. При хранении в дереве регистр букв не важен.

Кроме построения префиксного дерева по заданному набору слов необходимо реализовать:

* функцию-предикат, проверяющую наличие конкретного слова в дереве-словаре;
* функцию вставки в дерево нового слова и функцию исключения из него заданного слова;
* функцию, осуществляющую структурный вывод построенного дерева до заданной глубины N (N ≥ 2), реализуя при выводе переход на более глубокий уровень с помощью пробелов-отступов (при этом дуги одного уровня печатаются по строкам друг под другом в алфавитном порядке).

Исходный набор слов может быть задан как список, элементами которого являются списки букв слов, например: *((t e n ) ( t e a) (u s e) ...)* .

## 11. Поиск на мультиграфе, отображающем систему дорог (высокая сложность - максимальная оценка ОТЛ)

Система дорог – это мультиграф без петель, в котором вершины соответствуют городам, а ребра – дорогам между городами. Вершины помечены названиями городов, а ребрам приписаны положительные целые числа – длины (протяженности) соответствующих дорог.

Не для всех пар городов существует дорога между ними, в тоже время одна и та же пара городов может быть связана более чем одной дорогой (в общем случае, с разными длинами). Расстояние между двумя городами – это длина минимального пути между этими городами.

Все дороги допускают двустороннее движение. Известно также, что система дорог позволяет добраться из каждого города в любой другой.

Исходная задача: найти все пути без циклов между двумя заданными городами. Исходная система дорог (связный мультиграф) представляется как лисповский список входящих в него дорог (ребер), причем каждая дорога (ребро) записывается в виде трехэлементного списка, включающего названия городов, связанных этой дорогой (ребром), и ее длину. К примеру, возможная система дорог:

((А В 45) (В С 7) (В D 12) (A B 34) (D A 49)).

## 12. Поиск в лабиринте (средняя сложность - максимальная оценка ХОР, можно усложнить до ОТЛ)

Лабиринт состоит из входа, выхода и комнат, соединенных проходами (у каждой комнаты имеется не менее одного и не более трех проходов в соседние комнаты). Вход и выход также соединен с не более чем тремя комнатами.

Предполагается, что лабиринт проходим, т.е. существует хотя бы один путь по проходам лабиринта, соединяющий вход и выход лабиринта.

При записи лабиринта его комнаты идентифицируются номерами, вход и выход кодируются знаками IN и OUT, а также указываются все пары комнат, соединенных проходом.

Исходная задача: найти путь в заданном лабиринте, ведущий от входа к выходу и не имеющий циклов (т.е. проходов через одну и ту же комнату). Лабиринт представляется как лисповский список из проходов между комнатами (включая вход и выход).

Например, для лабиринта ((1 2) (2 4) (IN 1) (IN 3) (1 3) (4 5) (5 OUT)) один из возможных путей – (IN 3 1 2 4 5 OUT).

## 13. Работа с данными.

Даны таблицы с данными. Все поля имеют названия. Названия являются уникальными внутри одной таблицы, но могут повторяться в разных таблицах. К полю таблицы можно обратиться (Имя\_Таблицы Имя\_Поля). Такое обращение уникально.

Одно из полей таблицы может быть ключевым, т.е. значения этого поля являются уникальными. Типы всех полей заданы заранее. Остальные поля могут иметь любые значения, соответствующие типу поля. В ячейке таблицы может храниться либо целое число, либо слово, состоящее из латинских букв.

Между любыми двумя таблицами можно построить связь, т.е. одно из полей таблицы ссылается на ключ другой таблицы (типы обоих полей должны при этом совпадать). С помощью построенных связей можно получать наборы связных записей из разных таблиц.

1. Предложить способ хранения описанных таблиц в виде Лисповских списков.
2. Реализовать следующие функции для работы с этими списками:
   1. **SELECT**
      1. *(SELECT ( <name1> <name2> … ) FROM (<table>))* - вывести значения указанных полей указанной таблицы; (SELECT (\*) FROM (<table>)) – вывести значения всех полей указанной таблицы
      2. *(SELECT DISTINCT ( <name1> <name2> … ) FROM (<table>))* – без повторяющихся данных
      3. *(SELECT ( <name1> <name2> … ) FROM (<table>) WHERE (<condition>))*, только те, которые удовлетворяют указанному условию. В качестве условия могут быть: сравнение текста; сравнение численных значений.
   2. **INSERT**
      1. *(INSERT INTO (<table>) VALUES (<value1> <value2> <value3>…))* – добавляет в таблицу указанные значения. Предполагается, что порядок указанных значений соответствует порядку полей таблицы, но соответствие типов требуется проверять, как и уникальность ключевых полей.
      2. *(INSERT INTO (<table>) (<name1> <name2> <name3> …) VALUES (<value1> <value2> <value3> …))* – добавляет в таблицу указанные значения в указанные поля (поля могут идти в произвольном порядке, несовпадающем с порядком полей в таблице), если значение какого-то поля не указано, то добавляется nil.
   3. **DELETE**
      1. *(DELETE FROM (<table>) WHERE (<condition>))* - используется для удаления данных, удовлетворяющих указанному условию, из таблицы.
      2. каскадное удаление: если в других таблицах есть данные, ссылающиеся на удаляемые, то те записи тоже должны быть удалены (и так дальше в рекурсии).
   4. **UPDATE**
      1. *(UPDATE (<table>) SET ((<name1> <value1>) (<name2> <value2>) ...) WHERE (<condition>))* – обновить данные в тех записях, которые удовлетворяют указанному условию;
      2. каскадное обновление: если в других таблицах есть данные, ссылающиеся на изменяемые значения, то соответствующие значения тех записей тоже должны быть изменены (и так дальше в рекурсии).

На удовлетворительную оценку требуется реализовать первые функции из каждого раздела. Реализация функций с условиями WHERE повышает оценку до ХОРОШО. Реализация каскадного обновления и удаления повышает оценку до ОТЛИЧНО. Выбранный набор требуется сначала обсудить с преподавателем и закрепить за собой получившийся вариант.

## 14. Геометрия

Точки на плоскости заданы с помощью Лисповского списка: ((x1 y1) (x2 y2)…).

* 1. **Найти все прямые, порождаемые этими точками, предложить удобную для последующей обработки структуру хранения этих прямых. Требуется:**
     1. найти параллельные и перпендикулярные прямые;
     2. найти прямые, содержащие более двух исходных точек;
     3. найти все точки пересечения получившихся прямых.
  2. **По двум наборам из трех точек построить две параболы**
     1. определить, пересекают ли параболы ось Х, если да, то найти эти точки;
     2. определить, пересекаются ли параболы, если да, найти эти точки пересечения;
     3. построить функцию, которая в любой точке Х принимает значение, равное среднему арифметическому значений заданных парабол в этой точке.
  3. **Дан набор точек на плоскости**
     1. найти две самые удаленные и самые близкие друг к другу точки;
     2. построить окружность минимального радиуса таким образом, что все заданные точки попадут внутрь окружности или окажутся на ее границе;
     3. если такое возможно, среди заданных найти точки, линия, проведенная через которые, делит оставшиеся на два равных множества (точки, оказавшиеся по разные стороны от прямой). Если таких наборов несколько – найти все.
  4. **По двум наборам из трех точек построить окружности**
     1. определить, пересекаются ли получившиеся окружности, найти эти точки пересечения;
     2. найти координаты центров обеих окружностей, определить, находится ли центр одной из них внутри другой;
     3. Если заданные окружности не пересекаются и одна не лежит внутри другой, то построить третью окружность с минимальным радиусом, которая будет касаться двух заданных и найти эти точки касания.

На удовлетворительную оценку требуется реализовать функции из любых двух разделов. Реализация функций трех разделов повышает оценку до ХОРОШО. Реализация полного списка функций повышает оценку до ОТЛИЧНО. Выбранный набор требуется сначала обсудить с преподавателем и закрепить за собой получившийся вариант.