Лабораторная работа №1

Тема: распознавание образов

Цель работы: Получение практических навыков работы с простейшими алгоритмами распознавания объектов с качественными характеристиками.

Задание: На любом языке программирования реализовать программу, определяющую степень сходства образцов, поданных на вход. Использовать при этом все формулы сходства и проанализировать полученные результаты. В качестве образцов используйте данные своего варианта. Сформулируйте признаки, по которым будет осуществляться сравнение (не менее 5).

Отчет по лабораторной работе должен содержать:

1. Фамилию и номер группы учащегося, задание
2. Краткое описание алгоритма работы программы
3. Полный текст программы и протокол ее работы для трех прогонов
4. Анализ предложенных формул сходства.

Основные теоретические сведения

В большинстве случаев образы и отдельные изображения характеризуются с помощью *количественных* характеристик: геометрических размеров, веса, площади, объема и т. д. В этих случаях количественные изменения характеристик конкретного изображения обычно не сразу ведут к изменению образа, к которому относится распознаваемое изображение. Только достигнув определенных для каждого образа границ, количественные изменения вызывают качественный скачок - переход к другому образу. Образы и конкретные изображения могут характеризоваться не только количественными, но и *качественными* характеристиками (свойствами, признаками, атрибутами). Эти признаки не могут быть описаны (или обычно не описываются) количественно, например, цвет, вкус, ощущение, запах. Образы либо обладают какими-то качественными характеристиками, либо не обладают.

Между качественными и количественными характеристиками образов есть существенное различие, однако это различие во многих случаях нельзя абсолютизировать, поскольку каждому качественному атрибуту присущи и определенные интервалы изменения количественных характеристик, за пределами которых меняется и качественный атрибут.

Например, определенному цвету изображения соответствует конкретный диапазон длин электромагнитных волн, за пределами которого цвет изменится.

Существуют различные подходы к распознаванию изображений с качественными характеристиками. В данной лабораторной работе рассмотрим один из них, основанный на двоичном кодировании наличия или отсутствия какого-либо качественного признака. В рассматриваемом подходе конкретное изображение X_k некоторого образа с качественными характеристиками представляется в виде двоичного вектора

$$X_k = (x_{k1}, x_{k2}, \ldots, x_{kn})$$

где n — размерность пространства признаков.

Если изображение X_k обладает j-м признаком, то $x_{kj}=1$, а если нет, то $x_{kj}=0$, т. е. здесь отождествляется объект и двоичный вектор, его описывающий.

Рассмотрим в качестве примера четыре объекта (вишня, апельсин, яблоко, дыня), каждый из которых имеет три признака: цвет, наличие косточки или семечек (табл. 1). В табл. 2 приведены числовые значения признаков для рассматриваемого примера после их двоичного кодирования.

Наиболее простой метод решения задач распознавания объектов с качественными характеристиками после двоичного кодирования атрибутов — свести решение исходной задачи к решению задачи распознавания объектов с количественными характеристиками в n-мерном векторном пространстве. Для этого необходимо для каждого качественного признака ввести в n-мерном векторном пространстве ось. Если для рассматриваемого объекта признак существует, то на оси откладывается единица, если нет — то нуль. В результате получается многомерное двоичное пространство признаков, где можно использовать различные расстояния, применяемые для распознавания объектов с количественными характеристиками.

Таблица 1

	Вектор признаков	Желтый цвет	Оранжевый цвет	Красный цвет	Есть косточка	Есть семечки
Вишня	X_1	нет	нет	да	да	нет
Апельсин	X_2	нет	да	нет	нет	да
Яблоко	X_3	да	нет	да	нет	да
Дыня	X_4	да	нет	нет	нет	да

Таблица 2

	Вектор признаков	Желтый цвет	Оранжевый цвет	Красный цвет	Есть косточка	Есть семечки
Вишня	X_1	$x_{11} = 0$	$x_{12}=0$	$x_{13}= 1$	$x_{14}= 1$	$x_{15}=0$
Апельсин	X_2	$x_{21}= 0$	$x_{22}= 1$	$x_{23}=0$	$x_{24}=0$	$x_{25}= 1$

Яблоко	X_3	x_{31}= 1	x_{32}=0	x_{33}= 1	x_{34}=0	x_{35}= 1
Дыня	X_4	x_{41}= 1	x_{42}=0	x_{43}=0	x_{44} =0	x_{45}= 1

В рассматриваемом примере в результате введения количественных характеристик вместо качественных признаков (табл. 2) получается пятимерное двоичное пространство, где можно применять расстояния по Евклиду (1), по Минковскому (2), расстояние, использующее сумму модулей разностей между соответствующими компонентами n-мерных векторов (3):

$$L_1(S_i, X_j) = \sqrt{\sum_{k=1}^{n} (s_{ik} - x_{jk})^2} \quad (1)$$

$$L_2(S_i, X_j) = \sqrt[\lambda]{\sum_{k=1}^{n} (s_{ik} - x_{jk})^\lambda} \quad (2)$$

$$L_3(S_i, X_j) = \sum_{k=1}^{n} |s_{ik} - x_{jk}| \quad (3)$$

где $L_p(S_i, X_j)$, $p = \overline{1,3}$ - соответствующее расстояние между входным изображением $S_i = (s_{i1}, ..., s_{in})$ и эталонным изображением $X_i = (x_{i1}, ..., x_{in})$ j-го образа; λ - целое положительное число, большее двух.

Расстояния (1) — (3) могут использоваться также и с весовыми коэффициентами.

При двоичном кодировании качественных признаков может применяться и расстояние по Хеммингу, которое вводится для любых двоичных векторов. Расстояние по Хеммингу между двумя двоичными векторами равно числу несовпадающих двоичных компонент векторов. Если вектора имеют все одинаковые компоненты, то расстояние между ними равно нулю, если вектора не имеют совпадающих компонент, то расстояние равно размерности векторов.

Более тонкая классификация объектов с качественными признаками получается при введении для каждой пары объектов Xj, Xn, для которых введено двоичное кодирование качественных признаков, переменных, характеризующих их общность или различие с помощью табл. 3.

Таблица 3

X_j	X_i	
	1	0
1	a	h
0	g	b

Переменная a в табл. 3 предназначена для подсчета числа общих признаков объектов X_j, и X_i. Она может быть вычислена с помощью соотношения

$$a = \sum_{k=1}^{n} x_{jk} x_{ik} ,$$

где x_{jk}, x_{ik} - двоичные компоненты векторов, описывающих объекты X_j, и X_i.

С помощью переменной b подсчитывается число случаев, когда объекты X_j, и X_i. не обладают одним и тем же признаком, $b = \sum_{k=1}^{n} (1 - x_{jk})(1 - x_{ik})$. Переменные g и h предназначены соответственно для подсчета числа признаков, присутствующих у объекта X_i и отсутствующих у объекта X_j, и, присутствующих у объекта X_j и отсутствующими у объекта $X_{i,}$, $g = \sum_{k=1}^{n} x_{ik}(1 - x_{jk})$, $h = \sum_{k=1}^{n} (1 - x_{ik}) x_{jk}$.

Из анализа переменных a, b, g, h следует, что, чем больше сходство между объектами X_j, и Xi, тем больше должна быть переменная a, т.е. мера близости объектов или функция сходства должна быть возрастающей функцией от a, функция сходства должна быть симметричной относительно переменных g и h. Относительно переменной b однозначный вывод сделать не удается, поскольку, с одной стороны, отсутствие одинаковых признаков у объектов может свидетельствовать об их сходстве, однако, с другой стороны, если у объектов общим является только отсутствие одинаковых признаков, то они не могут относиться к одному классу.

Наиболее часто применяются следующие функции сходства:

$$S_1(X_i, X_j) = \frac{a}{a+b+g+h} = \frac{a}{n} \text{ (функция сходства Рассела и Рао),}$$

$$S_2(X_i, X_j) = \frac{a}{n-b} \text{ (функция сходства Жокара и Нидмена),}$$

$$S_3(X_i, X_j) = \frac{a}{2a+g+h} \text{ (функция сходства Дайса),}$$

$$S_4(X_i, X_j) = \frac{a}{a+2(g+h)} \text{ (функция сходства Сокаля и Снифа),}$$

$$S_5(X_i, X_j) = \frac{a+b}{n} \text{ (функция сходства Сокаля и Мишнера),}$$

$$S_6(X_i, X_j) = \frac{a}{g+h} \text{ (функция сходства Кульжинского),}$$

$$S_7(X_i, X_j) = \frac{ab-gh}{ab+gh} \text{ (функция сходства Юла).}$$

Варианты заданий

Вариант	Эталоны
1	Linux, Windows, Unix, FreeBSD
2	Pascal, C++, PHP, JavaScript, Assembler
3	Boolean, double, char, spring
4	MatLab, MatCad, AutoCad, Statistica
5	Семантическая сеть, Фреймы, Продукции
6	локальная сеть, Интернет, интранет
7	WordPad, Блокнот, MS Word, Open Word
8	Paint, PhotoShop, CorelDraw
9	Mac, Intel, AMD
10	CD, DVD, Blu-ray, flash
11	класс, объект, массив, переменная, константа
12	MySQL, PostgreSQL, InterBase, MS SQL
13	Word, Excel, Access, PowerPoint
14	структурный, логический, имитационный, эволюционный (подходы к исследованию в области искусственного интеллекта)
15	Prolog, Lisp, РЕФАЛ
16	свободный вариант, студент сам предлагает вариант

Лабораторная работа №2

Тема: онтологии

Цель работы: получение практических навыков построения онтологий.

Задание: используя программу Protege:

1) создать онтологию согласно полученному варианту, онтология должна содержать:
   - иерархию классов (не менее 15);
   - назначенные классам простые свойства Data Properties (не менее 10);
   - назначенные классам свойства-отношения Object Properties (не менее 5) с указанием вида связи между индивидами (функциональная, симметричная и т.д.);
   - индивиды Individuals (не менее 10), с заполненными значениями свойств унаследованного класса;
   - аксиомы, наложенные на свойства и классы в Equivalent to и др. (не менее 5);
   - правила (не менее 5);
2) используя плагин OntoGraf (вкладка Window/Tabs/OntoGraf), получить визуальное отображение онтологии в виде графа;
3) онтология должна охватывать всю предметную область (требование полноты), и быть достаточно детализированной;
4) запустить Reasoner Pellet (скинуть плагин в папку плагинов и затем установить его при запуске Protege) и проверить онтологию на согласованность, если есть несогласованности, исправить;
5) используя плагин SWRLTab, написать правила (не менее 3), синхронизировать (вкладка Reasoner) и посмотреть результаты (прогнозируемые элементы онтологии);
6) используя плагин SQWRLTab, построить запросы к онтологии (не менее 3).

Отчет по лабораторной работе должен содержать:

1. Фамилию и номер группы учащегося, задание
2. Описание онтологий:
   a. классы, свойства, индивиды;
   b. запросы и их результаты (принтскрин);
   c. правила и прогнозируемые на их основе новые элементы (принтскрин);
3. Графическое отображение онтологий.

Основные теоретические сведения

В последние годы разработка онтологий - формальных явных описаний терминов предметной области и отношений между ними (Gruber 1993) – переходит из мира лабораторий по искусственному интеллекту на рабочие столы экспертов по предметным областям. Во всемирной паутине онтологии стали обычным явлением. Онтологии в сети варьируются от больших таксономий, категоризирующих веб-сайты (как на сайте Yahoo!), до категоризаций продаваемых товаров и их характеристик (как на сайте Amazon.com). Консорциум WWW (W3C) разрабатывает RDF (Resource Description Framework) (Brickley and Guha 1999), язык кодирования знаний на веб-страницах, для того, чтобы сделать их понятными для электронных агентов, которые осуществляют поиск информации. Управление перспективных исследований и разработок министерства обороны США (The Defense Advanced Research Projects Agency, DARPA) в сотрудничестве с W3C разрабатывает Язык Разметки для Агентов DARPA (DARPA Agent Markup Language, DAML), расширяя RDF более выразительными конструкциями, предназначенными для облегчения взаимодействия агентов в сети (Hendler and McGuinness 2000).

Онтология – формальное явное описание понятий в рассматриваемой предметной области (**классов** (иногда их называют **понятиями**)), свойств каждого понятия, описывающих различные свойства и атрибуты понятия (**слотов** (иногда их называют **ролями** или **свойствами**)), и ограничений, наложенных на слоты (**фацетов** (иногда их называют **ограничениями ролей**)).

Онтология вместе с набором индивидуальных **экземпляров** классов образует **базу знаний**. В действительности, трудно определить, где кончается онтология и где начинается база знаний.

Классы описывают понятия предметной области. Например, класс вин представляет все вина. Конкретные вина – экземпляры этого класса. Вино в конкретном бокале – это экземпляр класса вин. Класс может иметь подклассы, которые представляют более конкретные понятия, чем надкласс. Например, мы можем разделить класс всех вин на красные, белые и розовые вина. В качестве альтернативы мы можем разделить класс всех вин на игристые и не игристые вина.

Слоты описывают свойства классов и экземпляров: вино крепкое, оно производится на винном заводе Абрау-Дюрсо. У нас есть два слота, которые описывают вино в этом примере: слот крепость со значением «крепкое» и слот производитель со значением «винный завод Абрау-Дюрсо. Мы можем сказать, что на уровне класса у экземпляров класса Вино есть слоты, которые описывают вкус, крепость, уровень сахара, производителя вина и т.д.

Все экземпляры класса Вино и его подкласс Красное вино имеют слот производитель, значение которого является экземпляром класса Винный завод. Все экземпляры класса Винный завод имеют слот производит, относящийся ко всем винам (экземплярам класса Вино и его подклассов), которые производятся на этом заводе.

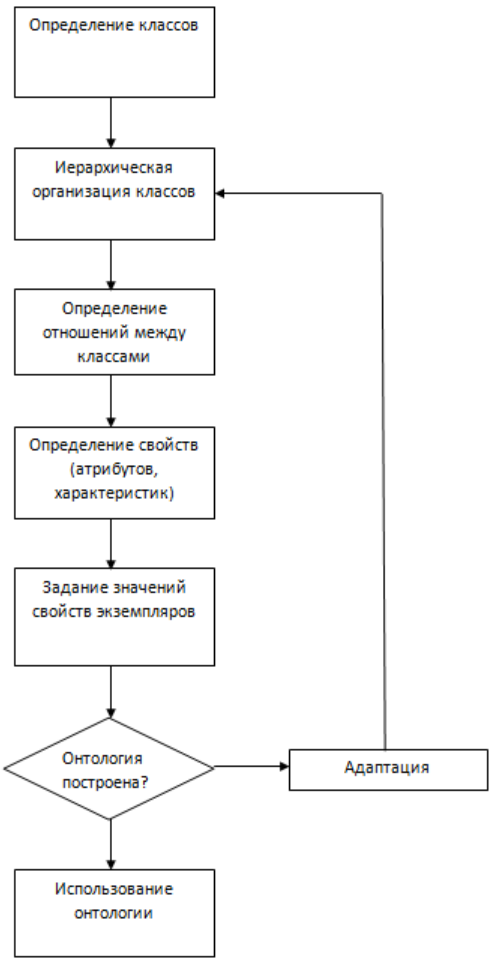
Правила построения онтологий.

1) Не существует единственного правильного способа моделирования предметной области – всегда существуют жизнеспособные альтернативы. Лучшее решение почти всегда зависит от предполагаемого приложения и ожидаемых расширений.

2) Разработка онтологии – это обязательно итеративный процесс.

3) Понятия в онтологии должны быть близки к объектам (физическим или логическим) и отношениям в интересующей вас предметной области. Скорее всего, это существительные (объекты) или глаголы (отношения) в предложениях, которые описывают вашу предметную область.

Схема процесса создания онтологии



Варианты заданий

1. Информационные системы
2. Интеллектуальные информационные системы
3. Компьютерные сети
4. Языки программирования
5. Модели представления знаний
6. Классификация нейронных сетей
7. Классификация экспертных систем
8. Классификация систем поддержки принятия решений
9. Электронно-вычислительные машины (компьютеры)
10. Направления искусственного интеллекта
11. Операционные системы
12. Классификация наук
13. Направления исследований в информатике
14. Протоколы Интернет
15. Свободный вариант, студент сам предлагает предметную область (связанна с информатикой) и согласовывает с преподавателем.

Характеристики свойств объектов

Функциональные свойства (Functional) – если свойство является функциональным, то для данного индивида (экземпляра) может существовать не более одного индивида, который имеет отношение к первому индивиду через это свойство.

Обратные функциональные свойства (Inverse functional) – если свойство является обратным функциональному свойству, то это значит, что свойство является обратным функциональным.

Транзитивные свойства (Transitive) – если свойство транзитивное и свойство связывает индивида a и индивида b, а также индивида b связывает с индивидом c, то мы можем вывести, что индивид a связан с индивидом c через это свойство.

Симметричные свойства (Simmetric) – если свойство p симметричное, и свойство связывает индивида a с индивидом b, то индивид b связан также с индивидом a через свойство p.

Асимметричные свойства (Asimmetric) – если свойство p асимметричное, и свойство связывает индивида a с индивидом b, то индивид b не может быть связан с индивидом a через свойство p.

Рефлексивные свойства (Reflexive) – свойство p называется рефлексивным, когда индивид a должен быть связан с собой.

Иррефлексивные свойства (Irreflexive) – если свойство p иррефлексивное, то оно может быть охарактеризовано как свойство, которое связывает индивида a с индивидом b, где индивид a и индивид b обязательно разные.

Список команд для построения аксиом (можно применять в Equivalent to)

Команда	Пример	Значение
some	hasPet some Dog	Things that have a pet that is a Dog This is the most common type of class expression. Also known as, an "SomeValueFrom restriction" or an "Existential Restriction". This kind of class expression consists of a property (in this case hasPet) and a class expression that is known as a filler (in this case the filler is Dog). Individuals that are instances of this class expression have a relationship along the hasPet property to an individual that is an instance of class Dog.
value	hasPet value Tibbs	Things that have a pet that is Tibbs. Here, Tibbs is a specific individual. Intuitively this would describe Tibb's owners. Note that this is a short cut for, and logically equivalent to, (hasPet some {Tibbs}), where the curly brackets describe a class of specific individuals - in this case, a class of one individual that is Tibbs. Also known as a "HasValue restriction"
only	hasPet only Cat	Things that have pets that are only Cats. Note that this does not mean that these things must have a Cat, but if they do have a pet then it will be a Cat. Also known as an "AllValuesFrom restriction" or a "Universal restriction"
min	hasPet min 3 Cat	Things that have at least three pets that are Cats. Things that have at least three pets that are Cats. Also known as a "Min cardinality restriction"
max	hasPet max 5 Dog	Things that have at most five pets that are Dogs. Note that there may be more than five pets in total e.g. three Cats and five Dogs, but the number of Dogs will not be more than five. This class expression also means that there may be no pets at all. Also known as a "Max cardinality restriction"
exactly	hasPet exactly 2 GoldFish	Things that have exactly 2 GoldFish as pets. Note that there may be more than two pets, but two of the pets are GoldFish - no more and no less. Also known as an "Exact cardinality restriction"
and	Person and (hasPet some Cat)	People that have a pet that's a Cat. Here we have a class name and a complex class expression combined with the and keyword. The brackets are optional in this particular case, but have been included for clarity. Also known as an "Intersection" or a "Conjunction". We also say that each class expression in the conjunction is a "conjunct".
or	(hasPet some Cat) or (hasPet some Dog)	Things that have a pet that's a Cat or have a pet that's a Dog Note that the or is not exclusive. That is, this class includes the things that have a Cat as a pet, or have a Dog as a pet, or have both a pet Dot and a pet Cat. Note that we could have also written this example as (hasPet some (Cat or Dog)) Also known as a "Union" or a "Disjunction". We also say that each class expression in the disjunction is a "disjunct".
not	not (hasPet some Dog)	Things that do not have a pet that's a dog. This includes things that either do not have any pet, or if they have a pet then it's not a Dog. Note that this is logically equivalent to (hasPet only (not Dog)) Also known as a "negation".

Список команд Built-Ins (можно применять в правилах)

Команда	Описание
for Comparisons	
swrlb:equal (from XQuery op:numeric-equal, op:compare, op:boolean-equal op:yearMonthDuration-equal, op:dayTimeDuration-equal, op:dateTime-equal, op:date-equal, op:time-equal, op:gYearMonth-equal, op:gYear-equal, op:gMonthDay-equal, op:gMonth-equal, op:gDay-equal, op:anyURI-equal)	Satisfied iff the first argument and the second argument are the same.
swrlb:notEqual (from swrlb:equal)	The negation of swrlb:equal.
swrlb:lessThan (from XQuery op:numeric-less-than, op:compare, op:yearMonthDuration-less-than, op:dayTimeDuration-less-than, op:dateTime-less-than, op:date-less-than, op:time-less-than)	Satisfied iff the first argument and the second argument are both in some implemented type and the first argument is less than the second argument according to a type-specific ordering (partial or total), if there is one defined for the type. The ordering function for the type of untyped literals is the partial order defined as string ordering when the language tags are the same (or both missing) and incomparable otherwise.
swrlb:lessThanOrEqual (from swrlb:lessThan, swrlb:equal)	Either less than, as above, or equal, as above.
swrlb:greaterThan (from XQuery op:numeric-greater-than, op:compare, op:yearMonthDuration-greater-than, op:dayTimeDuration-greater-than, op:dateTime-greater-than, op:date-greater-than, op:time-greater-than)	Similarly to swrlb:lessThan
swrlb:greaterThanOrEqual (from swrlb:greaterThan, swrlb:equal)	Similarly to swrlb:lessThanOrEqual.
Math	
swrlb:add (from XQuery op:numeric-add)	Satisfied iff the first argument is equal to the arithmetic sum of the second argument through the last argument.
swrlb:subtract (from XQuery op:numeric-subtract)	Satisfied iff the first argument is equal to the arithmetic difference of the second argument minus the third argument.
swrlb:multiply (from XQuery op:numeric-multiply)	Satisfied iff the first argument is equal to the arithmetic product of the second argument through the last argument.
swrlb:divide (from XQuery op:numeric-divide)	Satisfied iff the first argument is equal to the arithmetic quotient of the second argument divided by the third argument.
swrlb:integerDivide (from XQuery op:numeric-integer-divide)	Satisfied if the first argument is the arithmetic quotient of the second argument idiv the third argument. If the numerator is not evenly divided by the divisor, then the quotient is the xsd:integer value obtained, ignoring any remainder that results from the division (that is, no rounding is performed).
swrlb:mod (from XQuery op:numeric-mod)	Satisfied if the first argument represents the remainder resulting from dividing the second argument, the dividend, by the third argument, the divisor. The operation a mod b for operands that are xsd:integer or xsd:decimal, or types

	derived from them, produces a result such that (a idiv b)*b+(a mod b) is equal to a and the magnitude of the result is always less than the magnitude of b. This identity holds even in the special case that the dividend is the negative integer of largest possible magnitude for its type and the divisor is -1 (the remainder is 0). It follows from this rule that the sign of the result is the sign of the dividend
swrlb:pow	Satisfied iff the first argument is equal to the result of the second argument raised to the third argument power.
swrlb:unaryPlus (from XQuery op:numeric-unary-plus)	Satisfied iff the first argument is equal to the second argument with its sign unchanged.
swrlb:unaryMinus (from XQuery op:numeric-unary-minus)	Satisfied iff the first argument is equal to the second argument with its sign reversed.
swrlb:abs (from XQuery fn:abs)	Satisfied iff the first argument is the absolute value of the second argument.
swrlb:ceiling (from XQuery fn:ceiling)	Satisfied iff the first argument is the smallest number with no fractional part that is greater than or equal to the second argument.
swrlb:floor (from XQuery fn:floor)	Satisfied iff the first argument is the largest number with no fractional part that is less than or equal to the second argument
swrlb:round (from XQuery fn:round)	Satisfied iff the first argument is equal to the nearest number to the second argument with no fractional part.
swrlb:roundHalfToEven (from XQuery fn:round-half-to-even)	Satisfied iff the first argument is equal to the second argument rounded to the given precision. If the fractional part is exactly half, the result is the number whose least significant digit is even.
swrlb:sin	Satisfied iff the first argument is equal to the sine of the radian value the second argument.
swrlb:cos	Satisfied iff the first argument is equal to the cosine of the radian value the second argument.
swrlb:tan	Satisfied iff the first argument is equal to the tangent of the radian value the second argument.
for Boolean Values	
swrlb:booleanNot (from XQuery fn:not)	Satisfied iff the first argument is true and the second argument is false, or vice versa.
for Strings	
swrlb:stringEqualIgnoreCase	Satisfied iff the first argument is the same as the second argument (upper/lower case ignored)
swrlb:stringConcat (from XQuery fn:concat)	Satisfied iff the first argument is equal to the string resulting from the concatenation of the strings the second argument through the last argument.
swrlb:substring (from XQuery fn:substring)	Satisfied iff the first argument is equal to the substring of optional length the fourth argument starting at character offset the third argument in the string the second argument.
swrlb:stringLength (from XQuery fn:string-length)	Satisfied iff the first argument is equal to the

	length of the second argument.
swrlb:normalizeSpace (from XQuery fn:normalize-space)	Satisfied iff the first argument is equal to the whitespace-normalized value of the second argument.
swrlb:upperCase (from XQuery fn:upper-case)	Satisfied iff the first argument is equal to the upper-cased value of the second argument.
swrlb:lowerCase (from XQuery fn:lower-case)	Satisfied iff the first argument is equal to the lower-cased value of the second argument.
swrlb:translate (from XQuery fn:translate)	Satisfied iff the first argument is equal to the second argument with occurrences of characters contained in the third argument replaced by the character at the corresponding position in the string the fourth argument.
swrlb:contains (from XQuery fn:contains)	Satisfied iff the first argument contains the second argument (case sensitive).
swrlb:containsIgnoreCase	Satisfied iff the first argument contains the second argument (case ignored).
swrlb:startsWith (from XQuery fn:starts-with)	Satisfied iff the first argument starts with the second argument.
swrlb:endsWith (from XQuery fn:ends-with)	Satisfied iff the first argument ends with the second argument.
swrlb:substringBefore (from XQuery fn:substring-before)	Satisfied iff the first argument is the characters of the second argument that precede the characters of the third argument.
swrlb:substringAfter (from XQuery fn:substring-after)	Satisfied iff the first argument is the characters of the second argument that follow the characters of the third argument.
swrlb:matches (from XQuery fn:matches)	Satisfied iff the first argument matches the regular expression the second argument.
swrlb:replace (from XQuery fn:replace)	Satisfied iff the first argument is equal to the value of the second argument with every substring matched by the regular expression the third argument replaced by the replacement string the fourth argument.
swrlb:tokenize (from XQuery fn:tokenize)	Satisfied iff the first argument is a sequence of one or more strings whose values are substrings of the second argument separated by substrings that match the regular expression the third argument.
for Date, Time and Duration	
swrlb:yearMonthDuration (from XQuery xdt:yearMonthDuration)	Satisfied iff the first argument is the xsd:duration representation consisting of the year the second argument and month the third argument.
swrlb:dayTimeDuration (from XQuery xdt:dayTimeDuration)	Satisfied iff the first argument is the xsd:duration representation consisting of the days the second argument, hours the third argument, minutes the fourth argument, and seconds the fifth argument.
swrlb:dateTime	Satisfied iff the first argument is the xsd:dateTime representation consisting of the year the second argument, month the third argument, day the fourth argument, hours the fifth argument, minutes the sixth argument,

	seconds the seventh argument, and timezone the eighth argument.
swrlb:date	Satisfied iff the first argument is the xsd:date representation consisting of the year the second argument, month the third argument, day the fourth argument, and timezone the fifth argument.
swrlb:time	Satisfied iff the first argument is the xsd:time representation consisting of the hours the second argument, minutes the third argument, seconds the fourth argument, and timezone the fifth argument.
swrlb:addYearMonthDurations (from XQuery op:add-yearMonthDurations)	Satisfied iff the yearMonthDuration the first argument is equal to the arithmetic sum of the yearMonthDuration the second argument through the yearMonthDuration the last argument.
swrlb:subtractYearMonthDurations (from XQuery op:subtract-yearMonthDurations)	Satisfied iff the yearMonthDuration the first argument is equal to the arithmetic difference of the yearMonthDuration the second argument minus the yearMonthDuration the third argument.
swrlb:multiplyYearMonthDuration (from XQuery op:multiply-yearMonthDuration)	Satisfied iff the yearMonthDuration the first argument is equal to the arithmetic product of the yearMonthDuration the second argument multiplied by the third argument.
swrlb:divideYearMonthDurations (from XQuery op:divide-yearMonthDuration)	Satisfied iff the yearMonthDuration the first argument is equal to the arithmetic remainder of the yearMonthDuration the second argument divided by the third argument.
swrlb:addDayTimeDurations (from XQuery op:add-dayTimeDurations)	Satisfied iff the dayTimeDuration the first argument is equal to the arithmetic sum of the dayTimeDuration the second argument through the dayTimeDuration the last argument.
swrlb:subtractDayTimeDurations (from XQuery op:subtract-dayTimeDurations)	Satisfied iff the dayTimeDuration the first argument is equal to the arithmetic difference of the dayTimeDuration the second argument minus the dayTimeDuration the third argument.
swrlb:multiplyDayTimeDurations (from XQuery op:multiply-dayTimeDuration)	Satisfied iff the dayTimeDuration the first argument is equal to the arithmetic product of the dayTimeDuration the second argument multiplied by the third argument.
swrlb:divideDayTimeDuration (from XQuery op:divide-dayTimeDuration)	Satisfied iff the dayTimeDuration the first argument is equal to the arithmetic remainder of the dayTimeDuration the second argument divided by the third argument.
swrlb:subtractDates (from XQuery op:subtract-dates)	Satisfied iff the dayTimeDuration the first argument is equal to the arithmetic difference of the xsd:date the second argument minus the xsd:date the third argument.
swrlb:subtractTimes (from XQuery op:subtract-times)	Satisfied iff the dayTimeDuration the first argument is equal to the arithmetic difference of the xsd:time the second argument minus the xsd:time the third argument.
swrlb:addYearMonthDurationToDateTime (from	Satisfied iff the xsd:dateTime the first argument is equal to the arithmetic sum of the

XQuery op:add-yearMonthDuration-to-dateTime)	xsd:dateTime the second argument plus the yearMonthDuration the third argument.
swrlb:addDayTimeDurationToDateTime (from XQuery op:add-dayTimeDuration-to-dateTime)	Satisfied iff the xsd:dateTime the first argument is equal to the arithmetic sum of the xsd:dateTime the second argument plus the dayTimeDuration the third argument.
swrlb:subtractYearMonthDurationFromDateTime (from XQuery op:subtract-yearMonthDuration-from-dateTime)	Satisfied iff the xsd:dateTime the first argument is equal to the arithmetic difference of the xsd:dateTime the second argument minus the yearMonthDuration the third argument.
swrlb:subtractDayTimeDurationFromDateTime (from XQuery op:subtract-dayTimeDuration-from-dateTime)	Satisfied iff the xsd:dateTime the first argument is equal to the arithmetic difference of the xsd:dateTime the second argument minus the dayTimeDuration the third argument.
swrlb:addYearMonthDurationToDate (from XQuery op:add-yearMonthDuration-to-date)	Satisfied iff the xsd:date the first argument is equal to the arithmetic sum of the xsd:date the second argument plus the yearMonthDuration the third argument.
swrlb:addDayTimeDurationToDate (from XQuery op:add-dayTimeDuration-to-date)	Satisfied iff the xsd:date the first argument is equal to the arithmetic sum of the xsd:date the second argument plus the dayTimeDuration the third argument.
swrlb:subtractYearMonthDurationFromDate (from XQuery op:subtract-yearMonthDuration-from-date)	Satisfied iff the xsd:date the first argument is equal to the arithmetic difference of the xsd:date the second argument minus the yearMonthDuration the third argument.
swrlb:subtractDayTimeDurationFromDate (from XQuery op:subtract-dayTimeDuration-from-date)	Satisfied iff the xsd:date the first argument is equal to the arithmetic difference of the xsd:date the second argument minus the yearMonthDuration the third argument.
swrlb:addDayTimeDurationToTime (from XQuery op:add-dayTimeDuration-to-time)	Satisfied iff the xsd:time the first argument is equal to the arithmetic sum of the xsd:time the second argument plus the dayTimeDuration the third argument.
swrlb:subtractDayTimeDurationFromTime (from XQuery op:subtract-dayTimeDuration-from-time)	Satisfied iff the xsd:time the first argument is equal to the arithmetic difference of the xsd:time the second argument minus the dayTimeDuration the third argument.
swrlb:subtractDateTimesYieldingYearMonthDuration (from XQuery fn:subtract-dateTimes-yielding-yearMonthDuration)	Satisfied iff the yearMonthDuration the first argument is equal to the arithmetic difference of the xsd:dateTime the second argument minus the xsd:dateTime the third argument.
swrlb:subtractDateTimesYieldingDayTimeDuration (from XQuery fn:subtract-dateTimes-yielding-dayTimeDuration)	Satisfied iff the dayTimeDuration the first argument is equal to the arithmetic difference of the xsd:dateTime the second argument minus the xsd:dateTime the third argument.
for URIs	
swrlb:resolveURI (from XQuery op:resolve-uri)	Satisfied iff the URI reference the first argument is equal to the value of the URI reference the second argument resolved relative to the base URI the third argument.
swrlb:anyURI	Satisfied iff the first argument is a URI reference consisting of the scheme the second argument, host the third argument, port the fourth argument, path the fifth argument, query the sixth argument, and fragment the seventh argument.
for Lists	
swrlb:listConcat (from Common Lisp append)	Satisfied iff the first argument is a list

	representing the concatenation of the lists the second argument through the last argument.
swrlb:listIntersection	Satisfied iff the first argument is a list containing elements found in both the list the second argument and the list the third argument.
swrlb:listSubtraction	Satisfied iff the first argument is a list containing the elements of the list the second argument that are not members of the list the third argument.
swrlb:member	Satisfied iff the first argument is a member of the list the second argument.
swrlb:length (from Common Lisp list-length)	Satisfied iff the first argument is the length of the list the second argument (number of members of the list).
swrlb:first (from rdf:first)	Satisfied iff the first argument is the first member of the list the second argument.
swrlb:rest (from rdf:rest)	Satisfied iff the first argument is a list containing all members of the list the second argument except the first member (the head).
swrlb:sublist	Satisfied iff the list the first argument contains the list the second argument.
swrlb:empty (from rdf:nil)	Satisfied iff the list the first argument is an empty list.

Лабораторная работа №3

Тема: экспертные системы

Цель работы: получение практических навыков применения специализированных оболочек для построения продукционных экспертных систем.

Задание: На языке CLIPS реализовать экспертную систему согласно полученному варианту. Ориентировочное количество продукций базы знаний не менее 25 шт. В качестве справочного материала по оболочке CLIPS используйте «Самоучитель по CLIPS», расположенный в этой же папке. Разрешается делать вариант двоим, при условии, что база знаний будет содержать не менее 50 правил.

Отчет по лабораторной работе должен содержать:

1. Фамилию и номер группы учащегося, задание
2. Краткое описание предметной области и решаемой задачи (каков результат работы системы, что является входными данными)
3. Описание примененного подхода к формализации знаний с необходимыми рисунками, диаграммами (деревья И/ИЛИ, деревья решений) и т.д.
4. Полный текст базы знаний и протокол ее работы для трех консультаций

Основные теоретические сведения

Экспертные системы – это сложные программные комплексы, аккумулирующие знания специалистов в конкретных предметных областях и тиражирующие эти знания для консультации менее квалифицированных специалистов[1].

Оболочка ЭС – «пустая» экспертная система[2].

Оболочка ЭС - инструментальное средство для проектирования и создания экспертных систем. В состав оболочки входят средства проектирования баз знаний с различными формами представления знаний и выбора режима работы решателя задач. Для конкретной предметной области инженер по знаниям определяет нужное представление знаний и стратегии решения задач, а затем, вводя их в оболочку, создает конкретную экспертную систему.

Применение оболочки ЭС позволяет достаточно быстро и с минимальными затратами создать исследовательскую, демонстрационную или промышленную ЭС. EMYCIN – первая оболочка, основанная на MYCIN[3].

---

[1] Ясницкий Л.Н. Введение в искусственный интеллект. - М.: Издательский центр «Академия», 2005. – 20.
[2] Словарь терминов // http://www.buk.ru/dict_view/662/ 07.04.07 г.

Современные оболочки ЭС предлагают следующие **возможности** (в каждой конкретной оболочке представлены частично):

- гибридное представление знаний (EsWin);
- выбор из нескольких стратегий вывода (G2, CLIPS);
- подключение библиотек и других систем (ACTIVATION FRAMEWORK);
- архитектура на основе «доски объявлений» (HEARSAY-III);
- архитектура «клиент-сервер» (JESS);
- интеграция в Интернет/Интранет (Egg2Lite, Exsys Corvid);
- графический интерфейс (WindExS, WxCLIPS);
- подсистема моделирования (G2);
- модульное построение системы (ReThink, G2);
- визуализация структуры баз знаний (W.E.S.T.) и т.д.

Создание ЭС с использованием оболочки - это процесс заполнения оболочки ЭС данными, знаниями, алгоритмами решения, настройка механизма вывода, разработка пользовательского интерфейса, отвечающего требованиям и стандартам конкретной предметной область.

CLIPS (C Language Integrated Production System) является одним из распространенных инструментальных средств разработки экспертных систем (ЭС). Представляя собой логически полную среду, содержащую встроенный редактор и средства отладки, CLIPS является оболочкой ЭС. Разработчиком CLIPS является Национальное Аэрокосмическое Агентство США. Первая версия системы вышла в 1984 году, текущая версия -6.1.

CLIPS использует продукционную модель представления знаний и поэтому содержит три основных элемента:

1. список фактов
2. базу знаний
3. блок вывода

Принципиальным отличием данной системы от аналогов является то, что она полностью реализована на языке С. Причем исходные тексты ее программ опубликованы в сети Интернет.

В CLIPS используется оригинальный LIPS-подобный язык программирования, ориентированный на разработку ЭС. Кроме того, CLIPS поддерживает еще две парадигмы программирования: объектно-ориентированную и процедурную.

---

[3] Одна из первых ЭС для медицинской диагностики. Разработана группой по инфекционным заболеваниям Стенфордского университета, ее разработка была начата в 1974 г.

Варианты заданий

1. Определение конфигурации персонального компьютера в зависимости от потребностей пользователя
2. Выбор оптимального способа подключения к интернет
3. Диагностика неисправностей компьютера
4. Рекомендация информационной системы в зависимости от потребностей пользователя
5. Рекомендация учебников или книг (по информатике) в зависимости от потребностей пользователя
6. Рекомендация средств разработки ПО (среды, языки, платформы) в зависимости от требований к будущей системе
7. Выбор метода решения задачи по ее описанию (известные алгоритмы, методы искусственного интеллекта и т.д.)
8. Рекомендации по конфигурированию локальной сети
9. Рекомендации по выбору периферийных устройств
10. Рекомендации по формату хранения информации разного типа
11. Свободный вариант, студент сам предлагает предметную область, связанную с информатикой

Лабораторная работа №4

Тема: принятие решений

Цель работы: получение практических навыком применения методов принятия решений.

Задание: На любом языке программирования напишите программу, позволяющую решать задачу вашего варианта при произвольных входных данных.

Отчет по лабораторной работе должен содержать:

1. Фамилию и номер группы учащегося, задание
2. Описание алгоритма принятия решения с необходимыми рисунками, диаграммами (деревья И/ИЛИ, деревья решений) и т.д.
3. Листинг программы и протокол ее работы для трех запусков с разными данными

Основные теоретические сведения

Принятие решения – это особый вид человеческой деятельности, направленный на выбор лучшей из имеющихся альтернатив. Главной задачей, которую приходится разрешать при принятии решения, «является выбор альтернативы, наилучшей для достижения некоторой цели, или ранжирование множества возможных альтернатив по степени их влияния на достижение этой цели»[1].

Интеллектуальные игры — это одна из областей искусственного интеллекта, где оптимистические прогнозы ученых 50-х годов прошлого века, хотя и с большим опозданием, но полностью сбылись. В 1998 г. в Нью-Йорке в матче Deep Blue против Гарри Каспарова компьютер впервые победил чемпиона мира по шахматам. Матч состоял из шести партий и завершился со счетом 3,5 на 2,5 в пользу компьютера.

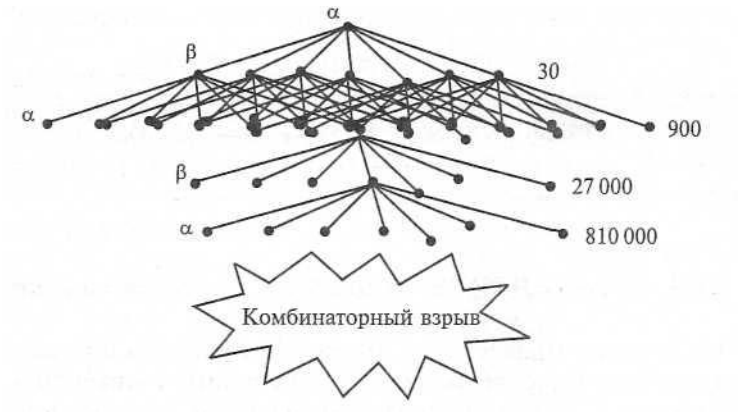
Понятие «игра» имеет более широкое значение. Игрой можно считать многие экономические, политические, военные и другие конфликты.

И в играх и при принятии решений в различных предметных областях используют одинаковые методы. Принципы работы, предложенные разработчиками, опираются на исследования дерева возможных продолжений игры. Корневая вершина дерева возможностей представляет собой текущее положение фигур на шахматной доске, а работа программы состоит в выборе очередного хода.

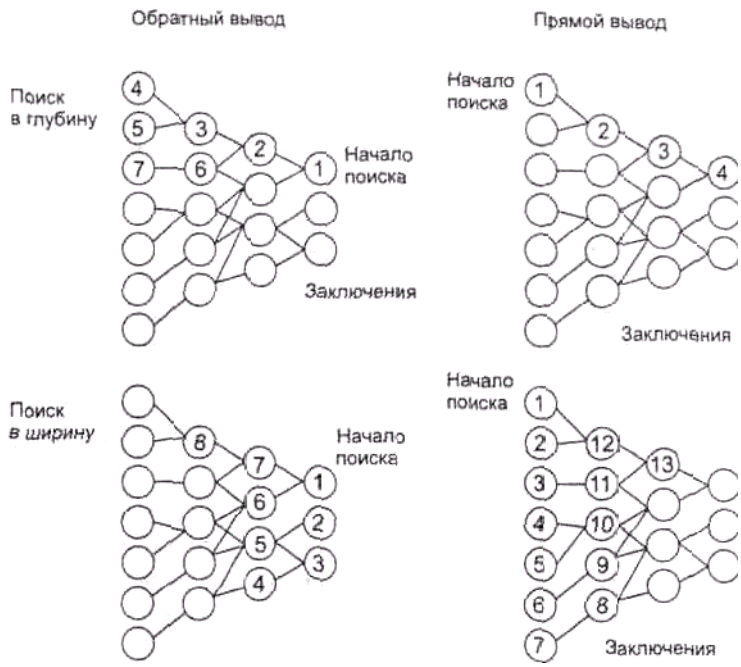
В середине партии у игрока обычно имеется около 30 возможных вариантов следующего хода. Возникающие в результате их перебора конфигурации

[1]Тоценко В. Системы поддержки принятия решений - ваш инструмент для правильного выбора // Компьютерра, № 34, 1998. http://www.computerra.ru/offline/1998/262/1520/

представляются как дочерние вершины для данной корневой вершины. В каждой из дочерних вершин возможно около 30 ответов противника, так что для изображения результирующих конфигураций потребуется еще около 900 вершин и т.д. Дерево быстро разрастается, что приводит к комбинаторному взрыву.



Дерево возможных продолжений шахматной игры



Стратегии вывода

Все вершины могут быть двух типов. В одних очередной ход предстоит делать компьютеру, в других — его противнику. Первые называют *альфа-вершинами,* вторые — *бета-вершинами.* Таким об разом, дерево возможностей представляет собой чередующиеся слои альфа- и бета-вершин.

Если бы дерево можно было обследовать полностью, т.е. вплоть до листьев, представляющих собой все возможные окончания в данной игре, то имелась бы возможность выбрать ход, обеспечивающий для компьютера выигрыш независимо от реакций противника. Такая возможность имеется в простейших играх, таких как крестики—нолики. В интеллектуальных играх типа шахмат удается построить и

просмотреть лишь небольшую часть дерева возможностей. В этом случае говорят, что дерево возможностей подвергается подрезке, а конечные вершины, ниже которых дерево отсечено, называют *терминальными* вершинами.

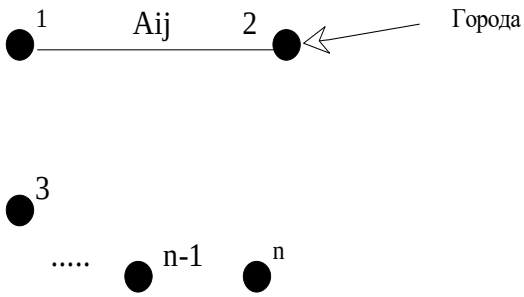
Поиск наилучшего решения в дереве может осуществляться по различным стратегиям. Для выбора стратегии необходимо ответить на 2 вопроса:

- Какую точку в пространстве состояний принять в качестве исходной? От выбора этой точки зависит и метод осуществления поиска — в прямом или обратном направлении.
- Какими методами можно повысить эффективность поиска решения? Эти методы определяются выбранной стратегией перебора — глубину, в ширину, по подзадачам или иначе.

Варианты заданий

1) *Задача коммивояжера*

A_{ij} - стоимость проезда между городами i и j. $A = \left\| A_{ij} \right\|_{n \times n}$ - таблица тарифов.

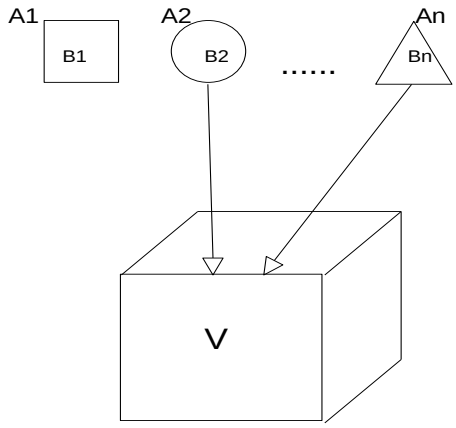


Найти маршрут для к $i_0 \to i_1 \to i_2 \to ... \to i_{l-1} \to i_l = i_0$ такой, что $\forall k$, p $i_k \neq i_p$ и

$$\sum_{m=0}^{l} A_{i_m i_{m+1}} \to \min$$

2) *Задача о рюкзаке*

A_n - объем предмета, B_n - стоимость предмета. Рюкзак имеет объем V



Требуется заполнить рюкзак так, чтобы:

$$\sum_{k=1}^{m} B_{i_k} \to \max,$$

$$\sum_{k=1}^{m} A_{i_k} \le V,$$

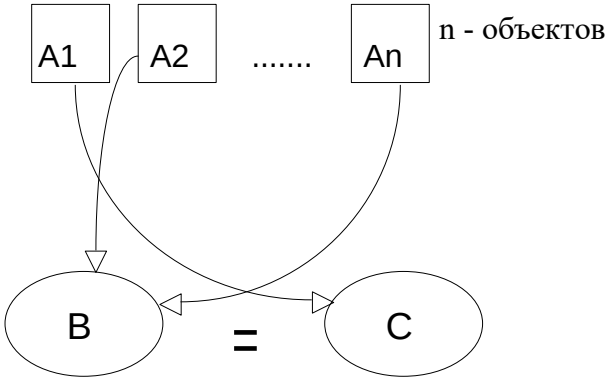
$$m \le n$$

3) *Задача о разбиении*

Разбиение - разделение множества $A=\{A_1, A_2, ...A_n\}$ на два подмножества B и C:

$$B=\left\{ A_{i_1}, A_{i_2}, \ldots, A_{i_m} \right\}$$

$$C=\left\{ A_{j_1}, A_{j_2}, \ldots, A_{j_l} \right\},$$



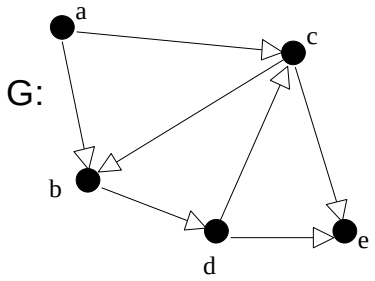
где m+l=n и B∩C=0, A=B∪C .

Требуется найти такой вариант, когда $\sum_{k=1}^{m} A_{i_k} = \sum_{p=1}^{l} A_{j_p}$.

4) *Задача об ориентированном графе*

Имеется ориентированный граф G, у которого заданы длины всех дуг.

Построить все ациклические пути из вершины **a** в вершину **b** на графе G и упорядочить их по возрастанию длин путей.



5) *Игра «Пятнашки»*

Разработать программу, решающую головоломку “Пятнашки” (решает программа, а не пользователь).

6) *Игра в крестики-нолики*

Написать программу игры “Крестики-нолики” на неограниченном поле. Игра ведется между пользователем и вашей программой.

7) *Числовой ребус*

D O N A L D

+

G E R A L D

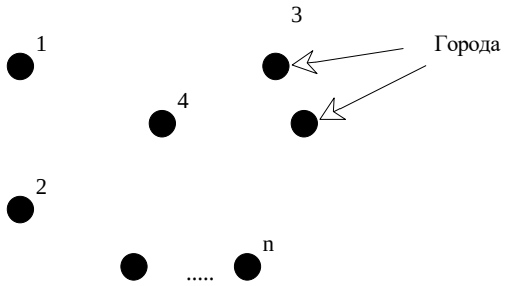
R O B E R T

Заменить буквы D, O, N и т.д. на цифры таким образом, чтобы сумма была правильной. Разным буквам соответствуют разные цифры.

8) *Задача о кратчайшем связывающем дереве*

Пусть a_{ij} - расстояния между i-м и j-м городом.

Требуется построить систему дорог между городами так, чтобы $L_{sum} \rightarrow \min$ и из любого города можно было бы попасть в любой другой город.



9) *Игра «Реверсы»*

Написать программу игры “Реверсы”. Игра ведется между пользователем и вашей программой.

10) *«Японский кроссворд»*

Написать программу решающую японские кроссворды (минимальная размерность кроссворда для сдачи10x10).

11) *Игра «Быки и коровы»*

Пользователь загадывает число из 4 цифр, каждая из которых от 1 до 6, причем все цифры различны. Написать программу, которая угадывает число по следующим правилам: выводится число и пользователь сообщает, сколько в нем "быков" и "коров", т.е. сколько цифр стоит на своих местах и сколько цифр содержится в обоих числах, но совпадают лишь по значению. Например, пусть загадано число 1264, спрошено 1256. В этом случае 2 быка (1,2) и одна корова (6)

12) Свободный вариант, студент сам предлагает задачу.