2. ЛАБОРАТОРНАЯ РАБОТА № 2
«ПОИСК РЕШЕНИЙ CSP ЗАДАЧ»

2.1. Цель работы

Изучение особенностей задач удовлетворения ограничений (**CSP** — **C**onstraint **S**atisfaction **P**roblem) и исследование основных методов поиска их решений средствами языка Пролог.

2.2. Краткие теоретические сведения

2.2.1. Общая формулировка задачи CSP

В системах искусственного интеллекта используют несколько способов представления задач [1]. Ранее рассматривалось представление задач в пространстве состояний, в соответствии с которым поиск решения задачи сводился к нахождению пути на графе состояний. В том случае, когда целью поиска является непосредственно конечное состояние задачи ~~и~~ сам ~~путь~~ не представляет интереса, может использоваться представление в виде CSP задачи. Любая задача CSP определяется совокупностью трех составляющих [1,2,8]:

1) множеством переменных $X_1, X_2, \ldots, X_n$;
2) областью определения каждой переменной $D_1, D_2, \ldots, D_n$;
3) множеством ограничений (отношений) $C_1, C_2, \ldots, C_m$, каждое из которых включает некоторое подмножество переменных и задает допустимые комбинации значений для этого подмножества.

Состояние задачи определяется путем присваивания значений некоторым или всем переменным. Присваивание, которое не нарушает никаких ограничений, называется *совместимым*. *Полным* называется присваивание, в котором участвует каждая переменная, а *решением* задачи CSP является полное присваивание, которое удовлетворяет всем ограничениям.

Уточним сказанное на примере задачи раскрашивания плоской карты (рис.2.1). Необходимо раскрасить карту, используя только 3 цвета. Соседние регионы на карте не должны раскрашиваться одним и тем же цветом.

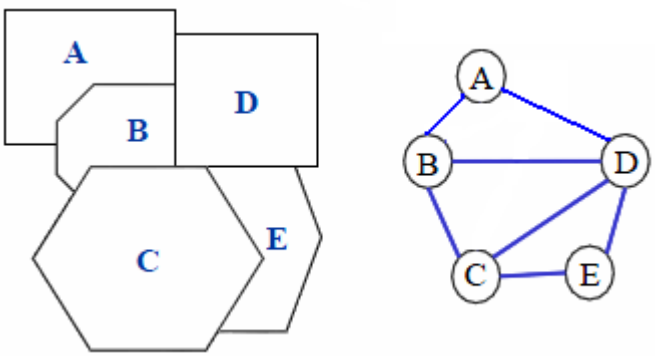

Рисунок 2.1 — Задача раскрашивания карты и граф ограничений

Чтобы определить эту задачу в виде задачи CSP в качестве переменных будем использовать буквенные обозначения регионов на карте: *A, B, C, D, E*. Областью определения каждой переменной является множество цветов D_i={*red, green, blue*}. Ограничения требуют, чтобы соседние регионы имели разные цвета, например, допустимые комбинации цветов для регионов *A* и *B*: { *(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)*}. В общем случае ограничения этой задачи можно записать в виде: $Color(X_i) \neq Color(X_j)$, где X_i и X_j переменные, обозначающие соседние регионы. Решение задачи заключается в том, чтобы всем переменным (*A, B, C, D, E*) присвоить совместимые значения.

Задачу CSP удобно представлять в виде *графа ограничений*, узлы которого представляют переменные задачи, а дуги — ограничения (см. рис. 2.1).

Переменные задачи могут быть дискретными или непрерывными, с конечными или бесконечными областями определений [8]. Ниже рассматриваются только простые CSP задачи *с дискретными переменными*, характеризуемыми *конечной областью определения*. Если максимальный размер области определения равен *d*, то в худшем случае количество возможных полных присваиваний оценивается величиной $O(d^n)$. Поэтому время решения задачи экспоненциально зависит от количества переменных. Однако в большинстве практических случаев алгоритмы CSP общего назначения позволяют решать задачи на несколько порядков более крупные, например, по сравнению алгоритмами поиска решений в пространстве состояний [8].

Ограничения (отношения) могут быть унарными, бинарными и ограничениями высокого порядка в зависимости от количества переменных. *Унарные* ограничения ограничивают значение одной переменной. Унарное ограничение можно устранить, удалив соответствующее значение из области определения переменной, нарушающее это ограничение. *Бинарное ограничение* связывает между собой две переменные. CSP задача, в которой используются только бинарные ограничения, может быть представлена графом ограничений. В *ограничениях высокого порядка* участвуют три и больше переменных. Ограничение высокого порядка может сведено к бинарным ограничениям путем введения вспомогательных переменных [8].

2.2.2. Общие методы решения CSP задач

Рассмотрим 4 метода решения CSP задач: «генерируй и тестируй», поиск с возвратами, поиск с предварительной проверкой, поиск с распространением ограничения [8].

2.2.2.1. Метод «генерируй и тестируй» (generate and test). Метод также называют методом «образуй и проверь» [2,11]. В соответствии с этим методом генерируются все возможные полные присваивания переменным, и каждое из них тестируется (проверяется) на совместимость. Структура соответствующей программы весьма проста и выглядит в виде вложенных циклов по каждой переменной. В самом внутреннем цикле выполняется проверка каждого ограничения. Если они все выполняются, то текущее полное присвоение — решение задачи.

В большинстве случае «слепая» генерация всех полных присваиваний весьма неэффективна, т.к. приводит к разрастанию дерева поиска.

2.2.2.2. Поиск с возвратами (backtracking search). По сути, это тот же метод «генерируй и тестируй», но организованный в виде поиска в глубину, в котором присваивается значение очередной переменной, проверяются ограничения и выполняется возврат, если присвоение не допустимо. ***Таким образом, проверка ограничений как бы погружается в процесс генерации решения, что позволяет ограничить разрастание дерева поиска.***

Эффективность метода зависит от порядка выбора переменных. Наиболее часто для определения порядка выбора переменных используют две эвристики: степенную эвристику и MRV-эвристику (**MRV — M**inimum **R**emaining **V**alues). *Степенная* эвристика позволяет уменьшить степень ветвления за счет выбора переменной, которая участвует в наибольшем количестве ограничений. *MRV-эвристика* предусматривает выбор переменной с наименьшим количеством оставшихся допустимых значений. Такая переменная с наибольшей вероятностью, вскоре приведет к неудаче, усекая тем самым дерево поиска. Степенная эвристика обычно используется для начального выбора переменных, а MRV-эвристика в ходе дальнейших присваиваний.

В процессе поиска с возвратами множество переменных, которым уже присвоены значения, на каждом шаге расширяется на очередную переменную, если выполняются соответствующие ограничения. Процесс завершается, если получено полное присваивание.

2.2.2.3. Метод предварительной (опережающей) проверки (forward checking). В случае поиска с возвратами ограничения, в которых участвует некоторая переменная, учитываются непосредственно в момент, когда происходит назначение значения этой переменной. Но выполняя опережающую проверку некоторых ограничений на предшествующих этапах поиска, можно резко сократить пространство поиска.

В соответствии с методом предварительной проверки в момент присваивания значения переменной *X* просматривается каждая переменная *Y*, которой не присвоены значения и которая связана с *X* некоторым ограничением. При этом из области определения Y удаляется любое значение, которое несовместимо со значением, присвоенным переменной *X*.

Вернемся к задаче раскрашивания карты. В соответствии с графом ограничений (см. рис. 2.1) после присваивания *A*=red и *C*=green области определения переменных *B* и *D* сокращаются до единственного значения (рис.2.2).

Переменные	***A***	***B***	***C***	***D***	***E***
Начальная обл. определения	R G B	R G B	R G B	R G B	R G B
После присваивания ***A***=red	**red**	G B	R G B	G B	R G B
После присваивания ***C***=green	**red**	B	**green**	B	R B
После присваивания ***E***=blue	**red**	B	**green**	-	**blue**

Рисунок 2.2 — Поиск с предварительной проверкой

Таким образом, ветвление, связанное с поиском значений для этих переменных, полностью устраняется (за счет распространения вперед информации о присвоенных значениях для переменных *A* и *C)*. После присваивания *E*=blue область определения *D* становится пустой, т.е. предварительная проверка обнаруживает, что частичное присваивание {*A*=red , *C*=green, *E*=blue} несовместимо с ограничениями задачи, и алгоритм выполнит возврат, не предпринимая попыток присваивания значений оставшимся переменным.

2.2.2.4. Метод распространения ограничения. Предварительная проверка не позволяет обнаруживать все несовместимости. Распространение ограничения — это общее название методов обнаружения потенциальных несовместимостей на ранних этапах решения задачи за счет распространения последствий применения некоторого ограничения к одной из переменных.

Для быстрого распространения ограничений выполняется проверка *совместимости дуг*. Дуга (*X*, *Y*) называется *совместимой*, если для каждого значения *x* из области определения переменной *X* существует некоторое значение *y* из области определения переменной *Y*, которое удовлетворяет бинарному ограничению между этими переменными. Обратим внимание на то, что понятие совместимости дуг является направленным, т.е. если дуга (*X*, *Y*) совместима, то это не означает, что обратная дуга (*Y, X*) также совместима.

Например, в третьей строке (рис. 2.2) текущими областями определения переменных *D* и *E* являются {blue} и {red, blue}. При *D*= blue существует совместимое присваивание для *E*, а именно *E*=red. Поэтому дуга от *D* до *E* совместима. С другой стороны, обратная дуга от *E* до *D* несовместима, так как для *E*=blue не существует совместимого присваивания. Эту дугу можно сделать совместимой, удалив значение blue из области определения *E*.

Проверку совместимости дуг можно использовать либо в качестве этапа предварительной обработки перед началом процесса поиска, либо в качестве этапа распространения ограничения после каждого присваивания во время поиска Проверка совместимости дуг требует дополнительных затрат времени, которые в наихудшем случае составляют $O(n^2 d^3)$. По этой причине в большинстве случаев при решении простых CSP задач ограничиваются применением поиска с возвратами или поиска с предварительной проверкой.

2.2.2.5. Обработка специальных ограничений. Встречаются некоторые типы ограничений, которые могут обрабатываться более эффективно с помощью специальных алгоритмов, а не общих алгоритмов, рассмотренных выше. Например, ограничение *Alldiff*, которое указывает, что все переменные задачи должны иметь разные значения. Один из простых способов проверки несовместимости для ограничения *Alldiff* сводится к правилу: если в ограничении участвуют *m* переменных и все они вместе взятые имеют *n* возможных значений, то при $m>n$ ограничение не может быть удовлетворено.

2.2.3. Решение CSP задач на языке Пролог

Так как в Пролог встроен механизм поиска с возвратами, который лежит в основе рассмотренных выше общих методов решения CSP задач, то на нём достаточно просто могут быть реализованы эти методы. Рассмотрим примеры написания соответствующих программ.

2.2.3.1. Прямая реализация метода «генерируй и тестируй». На Прологе метод реализуется простой конъюнкцией двух целей, одна из которых выполняет генерацию возможных решений (полных присваиваний), а вторая проверяет, удовлетворяют ли эти решения ограничениям задачи:

```
решить(X) :- генерировать_решение(X), проверить_ограничения(X).
```

Если проверка завершается неудачей, то происходит возврат к цели **генерировать_решение,** которая генерирует новое решение. Процесс продолжается до тех пор, пока не будет найдено решение, удовлетворяющее ограничениям, или генератор не исчерпает все альтернативные варианты предполагаемых решений.

В качестве основы генератора решений, часто используют запросы к базам данных или предикаты работы со списками: **принадлежит, удалить, перестановка**. Предикат **принадлежит** (встроенный предикат **member)** можно использовать не только для проверки того, что данный элемент входит в список, но и для того, чтобы получать (генерировать) элементы списка:

```
:- принадлежит( X, [a,b,c] ).
X = a ;
X = b ;
X = c
```

Предикат **удалить(X, L, L1)** проверяет, входит ли элемент **X** в список **L**, удаляет его и возвращает список **L1** без элемента **X**. Определение предиката:

```
удалить( X, [X|T], T ).
удалить( X, [H|T], [H|T1] ):-удалить(X, T, T1).
```

Предикат удаляет только одно вхождение **X**. Предикат можно использовать для извлечения элементов из списка (поэтому соответствующий встроенный предикат именуется как **select**):

```
:- удалить(X,[1,2,3],L1).
X = 1
L1 = [2, 3] ;
X = 2
L1 = [1, 3] ;
X = 3
L1 = [1, 2]
```

Его можно также использовать для вставки элемента в список. Например:

```
:- удалить(a, L, [1,2,3]).
L = [a, 1, 2, 3] ;
L = [1, a, 2, 3] ;
L = [1, 2, a, 3] ;
L = [1, 2, 3, a]
```

Для вставки элемента в список разумно определить отдельный предикат: **вставить(X,L1,L):-удалить(X,L,L1).**

Для генерации вариантов решений CSP задач, также полезным может оказаться предикат **перестановка(L, P)** с двумя аргументами-списками, один из которых является перестановкой элементов другого. Предикат определяется следующим образом:

```
перестановка( [ ],[ ] ).
перестановка( [X|L], P ):-перестановка(L,L1), вставить(X, L1, P).
```

Если первый список пустой, то и второй список должен быть пустым. Если первый список не пуст, то находим перестановку **L1** для хвоста списка **L**, а затем выполняем вставку головы списка **X** в произвольную позицию **L1**. Пример вызова предиката:

```
:-перестановка([red, blue, green], P).
P = [red, blue, green];
P = [blue, red, green];
P = [blue, green, red];
P = [red, green, blue];
P = [green, red, blue];
P = [green, blue, red]
```

Как и предполагалось, построены все шесть возможных перестановок.

Применим метод «генерируй и тестируй» к решению задачи раскрашивания карты, изображенной на рис. 2.1. При этом решение **X** будем представлять списком цветов вершин графа ограничений в порядке алфавита, т.е **[A,B,C,D,E].** Списки цветов будем генерировать с помощью предиката **перестановка**, а затем проверять, чтобы смежные вершины имели разные цвета. Программа получается довольно простой:

```
решить(X) :- генерировать_решение(X),  проверить_ограничения(X).

генерировать_решение(Список_цветов) :-
   перестановка([red, green, blue, red, green], Список_цветов).

проверить_ограничения(Список_цветов) :-
   Список_цветов = [A,B,C,D,E],
   A\==B,A\==D,          %цвет A отличается от цвета B и цвета D
   B\==D, B\==C,         %цвет B отличается от цвета D и цвета C
   C\==D, C\==E,         %цвет C отличается от цвета D и цвета E
   D\==E.                %цвет A отличается от цвета E
```

При запросе **решить(X)** получим следующий ответ: **X = [red, green, red, blue, green].**

Чтобы показать, что такой подход к построению программ является достаточно общим, рассмотрим логическую задачу. Известны следующие сведения о людях с именами: Иван, Нина, Сергей, Аня. Иван старше Нины. Сергей самый молодой. Аня не самая старшая. При заданных ограничениях, получить список

имен, упорядоченный по возрасту, т.е. **[И1, И2, И3, И4],** где **И1** самый молодой, т.е. Сергей. Построим программу по образцу предыдущей:

```
решить1(X) :-  генерировать_решение(X),  проверить_ограничения(X).

%формирование очередного решения  путем  перестановки имен в списке
генерировать_решение(Список_имен) :-
   перестановка([иван, нина, сергей, аня], Список_имен).

%проверка выполнения ограничений
проверить_ограничения(Список_имен) :-
   Список_имен = [И1, И2, И3, И4],
   предшествует(нина, иван, Список_имен),        %Иван старше Нины
   И1 = сергей,                                   %Сергей самый молодой
   принадлежит(аня, [И1,И2,И3]).                  %Аня не самая старшая

%отношение предшествует(X,Y,L) верно, если  X следует в списке L  раньше  Y
предшествует(X,Y,[X|T]) :-принадлежит(Y,T).
предшествует(X,Y,[_|T]) :- предшествует(X,Y,T).
```

Программа находит два решения: **X = [сергей, нина, аня, иван]** и **X = [сергей, аня, нина, иван].**

В большинстве случаев программы, реализующие прямой метод «генерируй и тестируй», неэффективны. Чтобы повысить эффективность процесса поиска решения, следует проверку ограничений выполнять по мере присваивания значений переменным. Ранняя проверка ограничений возможна при использовании поиска с возвратами, когда процесс проверки ограничений погружается в процесс генерации решений.

2.2.3.2. Реализация поиска с возвратами и предварительной проверкой ограничений. Вернемся к задаче раскрашивания карты. Представим карту (см. рис. 2.1.) в виде списка дуг **[A::B, A::D, B::D, B::C, D::C, D::E, C::E],** где символ **::** — инфиксный оператор (объявляемый в программе), который будет обозначать ребро графа. Обратим внимание, что ребра связывают переменные задачи, которым необходимо присвоить согласованные значения. Будем поочередно обрабатывать элементы этого списка, назначая цвета (из списка цветов) очередной паре переменных и проверять, чтобы эти цвета не совпадали. При этом, если какие-либо переменные получат значения, например для первого элемента **A=red** и **B=green**, то эти же значения автоматически назначаются и в других парах, т.е. распространяются вдоль всего списка, ограничивая (сокращая) области определения связанных переменных, в примере **D и C**. Следовательно, автоматически будет выполняться предварительная проверка ограничений.

Для назначения цвета переменной будем использовать предикат **принадлежит(X,L)**, который также будет порождать точки возврата. В случае возврата сделанные назначения автоматически стираются. Определим предикат **раскрасить1(Карта, Список_цветов)** рекурсивно:

```
%объявление инфиксного оператора
:- op(100,xfy,'::').

раскрасить1([X1::X2|Ост], Список_цветов):-
```

```
        принадлежит(X1, Список_цветов),          %назначение цвета X1
        принадлежит(X2, Список_цветов),          %назначение цвета X2
        X1\=X2,                                  %проверка ограничения
        раскрасить1(Ост, Список_цветов).         %раскрасить оставшуюся часть
раскрасить1([ ], _).
```

К недостаткам этого определения следует отнести, то что предикат **принадлежит** при первом вызове назначает переменным **X1** и **X2** одинаковые цвета, что создает очевидный возврат после проверки ограничения **X1\=X2.** Эта проблема легко преодолевается, если для назначения цвета **X1** использовать предикат **удалить(X1, Список_цветов, Оставшиеся_цвета)**, а цвет переменной **X2** выбирать из списка оставшихся цветов.

```
раскрасить2([X1::X2|Ост], Список_цветов):-
        удалить(X1,Список_цветов, Оставшиеся_цвета),  %назначение цвета X1
        принадлежит(X2, Оставшиеся_цвета),            %назначение цвета X2
        раскрасить2(Ост,Список_цветов).
раскрасить2([ ], _).
```

При запросе **раскрасить2([A::B, A::D, B::D, B::C, D::C, D::E, C::E], [red,green,blue])** получим следующий ответ:

```
A = C, C = red,
B = E, E = green,
D = blue.
```

Можно сравнить варианты решения задачи раскрашивания карты с использованием прямой реализации метода «генерируй и тестируй» и с использованием поиска с возвратами и предварительной проверкой. Для этого можно воспользоваться встроенным предикатом **time(Цель),** который возвращает статистику общего числа логических выводов при выполнении предиката **Цель** и процессорное время. Чтобы получить устойчивые результаты будем решение задачи выполнять 1000 раз:

```
:-time(повторять(Цель,1000)).

% цикл повторения выполнения Цели заданное число раз (N)
повторять(Цель,1):-Цель.
повторять(Цель,N):-
            not(not(Цель)),                  %стирание предыдущих подстановок
            M is N-1,повторять(Цель,M).
```

В результате вызова **time(повторять(решить(X),1000))** получим:

```
% 10,673,997 inferences, 2.137 CPU in 2.137 seconds (100% CPU, 4994352 Lips)
X = [red, green, red, blue, green]
```

Вызов **time(повторять(раскрасить2([A::B, A::D, B::D, B::C,D::C, D::E, C::E], [red, green, blue]),1000))** вернет то же решение. Однако поиск его происходит намного быстрее:

```
% 91,051 inferences, 0.047 CPU in 0.046 seconds (102% CPU, 1945522 Lips)
A = C, C = red,
```

B = E, E = green,
D = blue

В приложении Б.1 приведен еще один вариант решения задачи раскрашивания карты, который выполняется 2 раза быстрее **раскрасить2.**

Эффективный вариант реализации можно получить и для примера простейшей логической задачи рассмотренной выше. Оказывается, чтобы упорядочить список имен по возрасту методом поиска с возвратом и предварительной проверкой ограничений, ничего менять в основных определениях не надо. Достаточно просто сначала проверить выполнение ограничений, а затем дополнить полученный шаблон решения некоторыми не присвоенными значениями путем генерации перестановок. Иными словами, необходимо поменять местами цели в предикате **решить1(X).** Тогда получим следующее определение:

решить2(X):- проверить_ограничения(X), генерировать_решение(X).

Такое изменение порядка следования целей возможно, так как цель **проверить_ограничения(X)** может не только проверять ограничения, но и выполнять генерацию частичных решений. В таком случае мы добиваемся того, что проверка ограничений максимально погружается в процесс генерации потенциальных решений.

Вызов **решить2(X)** вернет решение аналогичное вызову **решить1(X).** Но если при выполнении предиката **решить1(X)** генерируется 4!=24 перестановки, то при выполнении предиката **решить2(X)** ограничение «Сергей самый молодой» сразу исключает 18 перестановок, а ограничение «Аня не самая старшая» — еще 2 перестановки. Сравнение времени поиска показывает, что вызов **решить2(X)** выполняется в 3 раза быстрее:

:-time(повторять(решить1(X),1000)).
% 104,997 inferences, 0.047 CPU in 0.043 seconds (109% CPU, 2243511 Lips)
X = [сергей, нина, аня, иван]

:-time(повторять(решить2(X),1000)).
% 41,997 inferences, 0.016 CPU in 0.016 seconds (98% CPU, 2692098 Lips)
X = [сергей, нина, аня, иван]

В приложении Б.2 приведен пример решения более сложной логической задачи с ограничениями типа *Alldiff*, для которой смена порядка следования целей «генерировать» и «тестировать» даёт еще больший выигрыш. В приложении Б.3. приведено решение головоломки Эйнштейна, для которой вариант с предварительной генерацией полных присваиваний совершенно не приемлем, т.к. потребует очень больших временных затрат.

2.3. Варианты заданий

Решить логическую задачу [7] с обязательным использованием методов поиска решений CSP задач на языке Пролог, рассмотренных в п.2.2.

1. Известно, что Единорог лжет по понедельникам, вторникам и средам и говорит правду во все остальные дни недели. Он может сказать: "Вчера я лгал. После завтрашнего дня я буду лгать два дня подряд". В какой день Единорог произнес эту фразу.

2. Три друга — Петр, Роман и Сергей — учатся на математическом, физическом и химическом факультетах. Если Петр — математик, то Сергей не физик. Если Роман не физик, то Петр — математик. Если Сергей не математик, то Роман — химик. Какая специальность у Сергея?

3. Четыре юных филателиста: Митя, Толя, Петя и Саша — купили почтовые марки. Каждый из них покупал марки только одной страны, причем двое из них купили российские марки, один — болгарские и один — чешские. Известно, что Митя и Толя купили марки двух разных стран. Марки разных стран купили Митя с Сашей, Петя с Сашей, Петя с Митей и Толя с Сашей. Кроме того, известно, что Митя купил не болгарские марки. Кто купил болгарские марки?

4. В чашке, стакане, кувшине и банке находятся молоко, лимонад, квас и вода. Известно, что вода и молоко не в чашке; сосуд с лимонадом стоит между кувшином и сосудом с квасом; в банке не лимонад и не вода; а стакан стоит между банкой и сосудом с молоком. Где находится вода.

5. Боря, Витя, Гриша и Егор встретились на олимпиаде. Ребята приехали из разных городов: один — из Твери, другой — из Омска, третий — из Томска, четвертый — из Казани. Известно, что Боря жил в одной комнате с мальчиком из Казани и ни один из них никогда не был ни в Твери, ни в Томске. Гриша играл в одной команде с мальчиком из Твери, а против них обычно сражался приятель из Казани. Егор и мальчик из Твери увлекались игрой в шахматы. Определить город, из которого приехал Егор.

6. На столе лежат в ряд четыре фигуры: треугольник, ромб, круг и квадрат. Квадрат, круг, ромб и треугольник вырезаны из белой, синей, красной и зеленой бумаги. Известно, что: круг не белый и не зеленый; синяя фигура лежит между ромбом и красной фигурой; треугольник не синий и не зеленый; квадрат лежит между треугольником и белой фигурой. Из бумаги какого цвета вырезан ромб?

7. Пятеро школьников из пяти различных городов приехали в Смоленск для участия в олимпиаде по математике. "Откуда вы, ребята?" — спросили их. Вот что они ответили: Андреев: "Я приехал из Ярославля, а Григорьев живет в Гагарине". Борисов: "В Гагарине живет Васильев, я прибыл из Вязьмы". Васильев: "Из Ярославля приехал я, а Борисов - из Ельни". Григорьев: Я приехал из Гагарина, а Данилов из Ярцева". Данилов: "Да, я действительно из Ярцева, Андреев живет в Вязьме". В каждом из высказываний одно утверждение правильное, а другое ложное. Откуда приехал Данилов?

8. Три девушки Аня, Варя и Клава ходили на демонстрацию. Одна из них была в красном платье, другая — в белом, третья — в синем. На вопрос, какое было платье на каждой из девушек они дали ответ: Аня была в красном, Варя — не в красном, Клава — не в синем. В этом ответе из трёх частей одна верна, две неверны. В каком платье была каждая из девушек ?

9. Четыре марсианки, оказавшиеся на Земле в 2222 году, на вопрос об их возрасте дали ответы: а) Ми — 22 года, Ме — 21 год; б) Мо — 19 лет, Ми — 21 год; в) Ма — 21 год, Мо — 18 лет. Все марсианки — разных возрастов, притом только таких — 18, 19, 21 и 22. В каждом ответе одна часть истинна, а другая - ложна. Сколько лет каждой из марсианок?

10. В велогонках приняли участие 5 школьников. После гонок 5 болельщиков заявили: 1) Коля занял 1-е место, а Ваня — 4; 2) Сережа занял 2-е место, а Ваня — 4; 3) Сережа занял 2-е место, а Коля — 3; 4) Толя занял 1-е место, а Надя — 4; 5) Надя заняла 3-е место, а Толя — 5. Зная, что одно из показаний каждого болельщика верное, а другое — ложное. Определить какое место занял Толя.

11. Шестеро юношей наблюдали прохожего, затем каждый из них ответил на вопрос, какого цвета его волосы, глаза и костюм, сколько ему лет. Андрей: рыжий, глаза голубые, костюм серый, 34. Вова: блондин, глаза карие, костюм синий, 30. Борис: рыжий, глаза карие, костюм коричневый, 32. Григорий: брюнет, глаза голубые, костюм, во всяком случае, не коричневый, 30. Дмитрий: шатен, глаза чёрные, костюм серый, 28. Евгений: блондин, глаза карие, костюм

синий, 32. Каждый из них трижды ошибся. Но из шести ответов на каждый из вопросов, по меньшей мере, один был верен. Каковы приметы прохожего?

12. Четверо ребят — Алексей, Борис, Иван и Григорий — соревновались в беге. На следующий день на вопрос, кто какое место занял, они ответили так :

Алексей : «Я не был ни первым, ни последним».

Борис : «Я не был последним».

Иван : «Я был первым».

Григорий : «Я был последним».

Известно, что три из этих ответов правильные, а один — неверный.

Кто сказал неправду, и кто был первым на самом деле?

13. Три купчихи — Олимпиада, Матрена и Евдокия — пили чай. Если бы Олимпиада выпила на 5 чашек больше, то она выпила бы столько, сколько две другие вместе. Если бы Матрена выпила бы на 9 чашек больше, то она выпила бы столько, сколько две другие вместе. Сколько каждая из трех купчих выпила чашек и у кого из них какое отчество, если известно, что: Уваровна пила чай вприкуску; количество чашек чая, выпитых Титовной, кратно трем; Карповна выпила 11 чашек.

14. В одном классе учатся три брата: Алексей, Леонид и Александр. Учитель заметил, что:

— если кто-то из них получает подряд две четверки или две тройки, то дальше он учится кое-как и получает тройку;

— если он получает подряд две пятерки, то совсем перестает заниматься и получает двойку;

— если он получает две разные оценки, то следующей будет большая из них.

В начале полугодия Алексей получил оценки 4 и 5, Леонид — 3 и 2, Александр — 2 и 4. Какие итоговые оценки получит каждый из рассматриваемых школьников за данное полугодие, если учитель выставил каждому по 30 оценок, а итоговая оценка — ближайшее целое число к среднему арифметическому полученных оценок?

15. Идет расследование преступления. Трое подозреваемых — Иванов, Петров и Сидоров — дают показания. Чтобы окончательно запутать следствие, каждый из трех подозреваемых называет правильно либо марку, либо цвет машины, на которой преступники скрылись с места преступления. Показания подозреваемых : Иванов сказал, что машина — синие «Жигули». Петров утверждал, что была черная «Волга». Сидоров показал, что был «Мерседес», при этом отрицая, что цвет машины — синий. Определить марку и цвет машины, на которой преступники скрылись с места преступления.

16. Идет расследование преступления. Трое подозреваемых — Иванов, Петров и Сидоров — дают показания. Иванов: «Я этого не делал. Это — Сидоров». Петров: «Сидоров этого не совершал. Это — Иванов». Сидоров: «Ни я, ни Петров этого не совершали». Следствием было установлено, что один из подозреваемых оба показания дал верно, другой — оба неверно, а третий — одно показание дал верно, другое — неверно. Требуется установить виновника преступления.

17. Есть пять коробочек: белая, черная, синяя, красная и зеленая. Шарики тех же цветов, что и коробочки, по два шарика каждого цвета. В каждой коробочке по два шарика. При этом известно, что:

— ни один шарик не лежит в коробочке того же цвета, что и он сам;

— в красной коробочке нет синих шариков;

— в коробочке нейтрального цвета лежит один красный и один зеленый шарик; нейтральный цвет — это либо белый, либо черный;

— в черной коробочке лежат шарики зеленых и синих цветов;

— в одной из коробочек лежат один белый и один синий шарик;

— в синей коробочке находится один черный шарик.

Требуется распределить шарики по коробочкам в соответствии с вышеуказанными условиями.

18. Пять обладателей выигрышных лотерейных билетов должны получить по два приза. На десяти карточках написали номера от одного до десяти и пригласили каждого счастливчика вытянуть по две карточки. К сожалению, при записи результатов произошла ошибка. В то время, как один из членов тиражной комиссии называл вслух числа, стоявшие на извлеченных карточках (например, : «Пять и семь»), другой по рассеянности складывал эти числа и записывал лишь их сумму (в рассмотренном нами примере это было число 12). Поэтому результаты в протоколе записаны так : Петров — 11, Семенов — 4, Иванов — 7, Сидоров — 16, Локтев — 17. Требуется определить, какие призы нужно получить каждому обладателю счастливого билета (номер каждого выигранного им приза), если карточки назад не возвращались. Написать программу, возвращающую результат в виде троек значений: Фамилия, Номер первого приза, Номер второго приза.

19. Дама сдавала в багаж : диван, чемодан, саквояж, картину, корзину, картонку и маленькую собачонку. Диван весил столько же, сколько чемодан и саквояж вместе, и столько же, сколько картина и картонка вместе. Картина, корзина и картонка весили поровну, причем каждая из них — больше, чем собачонка. Когда выгружали багаж, дама заявила, что собака не той породы. При проверке оказалось, что собака перевешивает диван, если к ней на весы добавить саквояж или чемодан. Требуется написать программу, которая доказала бы справедливость претензии дамы.

20. После представления «Ревизора» состоялся следующий диалог (в скобках указаны номера фраз).

Бобчинский : Это вы, Петр Иванович, первый сказали «Э!» (1). Вы сами так говорили (2).

Добчинский : Нет, Петр Иванович, я так не говорил (3). Это вы семгу первый заказали (4). Вы и сказали «Э!» (5). А у меня во рту зуб со свистом (6).

Бобчинский : Что я семгу первый заказал, это верно (7). И верно, что у вас зуб со свистом (8). А все-таки это вы первый сказали «Э!» (9).

Требуется определить, кто первым сказал «Э!»? Известно, что из девяти произнесенных в этом диалоге фраз четное число верных.

21. Четыре человека с именами: Аня, Борис, Костя, Дима — имеют определенные предпочтения. Предпочтения в цвете: красный, синий, желтый, зеленый. Предпочтения в еде: картофель, стейк, макароны, капуста. Также каждый из них идеализирует кого-либо из группы (возможно и себя). Определите, какой цвет, еда и идеал соответствует каждой из персон, если: Аня предпочитает картофель; тот, кто предпочитает красный цвет, любит стейк; один из них идеализирует самого себя; тот, предпочитает желтый цвет, идеализирует Диму; персона, которая предпочитает макароны, идеализируется тем, кто предпочитает синий цвет; персона, которая идеализирует Костю, любит картофель; персона, которая предпочитает синий цвет, идеализирует того, кто любит желтый цвет.

22. Четыре юных филателиста: Митя, Толя, Петя и Саша — купили почтовые марки. Каждый из них покупал марки только одной страны, причем двое из них купили российские марки, один — болгарские и один — чешские. Известно, что Митя и Толя купили марки двух разных стран. Марки разных стран купили Митя с Сашей, Петя с Сашей, Петя с Митей и Толя с Сашей. Кроме того, известно, что Митя купил не болгарские марки. Кто купил чешские марки?

23. В велогонках приняли участие 5 школьников. После гонок 5 болельщиков заявили: 1) Коля занял 1-е место, а Ваня — 4; 2) Сережа занял 2-е место, а Ваня — 4; 3) Сережа занял 2-е место, а Коля — 3; 4) Толя занял 1-е место, а Надя — 4; 5) Надя заняла 3-е место, а Толя — 5. Зная, что одно из показаний каждого болельщика верное, а другое — ложное. Определить, какое место занял Коля.

24. В чашке, стакане, кувшине и банке находятся молоко, лимонад, квас и вода. Известно, что вода и молоко не в чашке; сосуд с лимонадом стоит между кувшином и сосудом с квасом; в банке не лимонад и не вода; а стакан стоит между банкой и сосудом с молоком. Где находится квас?

25. Пятеро школьников из пяти различных городов приехали в Смоленск для участия в олимпиаде по математике. "Откуда вы, ребята?" — спросили их. Вот что они ответили: Андре-

ев: "Я приехал из Ярославля, а Григорьев живет в Гагарине". Борисов: "В Гагарине живет Васильев, я прибыл из Вязьмы". Васильев: "Из Ярославля приехал я, а Борисов — из Ельни". Григорьев: Я приехал из Гагарина, а Данилов из Ярцева". Данилов: "Да, я действительно из Ярцева, Андреев живет в Вязьме". В каждом из высказываний одно утверждение правильное, а другое ложное. Откуда приехал Андреев?

26. На столе лежат в ряд четыре фигуры: треугольник, ромб, круг и квадрат. Квадрат, круг, ромб и треугольник вырезаны из белой, синей, красной и зеленой бумаги. Известно, что: круг не белый и не зеленый; синяя фигура лежит между ромбом и красной фигурой; треугольник не синий и не зеленый; квадрат лежит между треугольником и белой фигурой. Какая фигура вырезана из зеленой бумаги?

27. Три друга — Петр, Роман и Сергей — учатся на математическом, физическом и химическом факультетах. Если Петр — математик, то Сергей не физик. Если Роман не физик, то Петр — математик. Если Сергей не математик, то Роман - химик. Какая специальность у Петра.

28. Боря, Витя, Гриша и Егор встретились на олимпиаде. Ребята приехали из разных городов: один — из Твери, другой — из Омска, третий — из Томска, четвертый — из Казани. Известно, что Боря жил в одной комнате с мальчиком из Казани и ни один из них никогда не был ни в Твери, ни в Томске. Гриша играл в одной команде с мальчиком из Твери, а против них обычно сражался приятель из Казани. Егор и мальчик из Твери увлекались игрой в шахматы. Определить город, из которого приехал Витя.

2.4. Порядок выполнения лабораторной работы

2.4.1. Изучить методы представления и решения CSP задач по лекционному материалу и книгам [1,2,8].

2.4.2. Ознакомиться по лекционному материалу и книгам [1,2] с объявлением операторов, с расширенным перечнем предикатов обработки списков, метаусловиями и отрицанием в языке Пролог. Изучить примеры применения этих средств Пролога для решения CSP задач, которые приведены в п. 2.2 настоящей лабораторной работы.

2.4.3. Ознакомиться с вариантом задания. Сформулировать задачу в терминах задач CSP (см. п. 2.2.1):

а) определить перечень переменных задачи;

б) специфицировать области определения каждой переменной;

в) определить отношения ограничения между переменными;

2.4.4. Ознакомиться с примером кода, приведенного в приложении Б, и, по аналогии, определить на языке Пролог необходимые предикаты для решения поставленной задачи.

2.4.5. Создать в среде программирования Пролог-проект и выполнить его отладку.

2.4.6. Исследовать свойства разработанной программы. Оценить количество просматриваемых сочетаний переменных, определить с помощью предиката **time** статистику выполнения программы. Сформулировать варианты улучшения программы. Предложить альтернативные методы решения задачи.

2.4.7. Зафиксировать результаты работы программы в виде экранных копий.

2.5. Содержание отчета

Цель работы, вариант задания, описание постановки задачи в терминах удовлетворения ограничений, выбор метода решения задачи с обоснованием, описание разработанной программы, описание машинных экспериментов с программой и анализ результатов, выводы.

2.6. Контрольные вопросы

2.6.1. Приведите формальное определение задачи удовлетворения ограничений.

2.6.2. Что понимают под состоянием SCP задачи, совместимым присваиванием, полным присваиванием, решением?

2.6.3. Приведите пример формулировки задачи раскрашивания карты.

2.6.4. Что понимают под графом ограничения?

2.6.5. Как классифицируются переменные CSP задач?

2.6.6. Как классифицируются ограничения CSP задач?

2.6.7. Объясните метод «генерируй и тестируй».

2.6.8. Объясните поиск с возвратами при решении CSP задач.

2.6.9. Объясните метод предварительной (опережающей) проверки.

2.6.10. Объясните метод распространения ограничения.

2.6.11. В чем суть ограничения *Alldiff* ?

2.6.12. Как на Прологе реализуется прямой метод «генерируй и тестируй»? Какие предикаты Пролога обычно используют для генерации вариантов решений?

2.6.13. Приведите определения предикатов **удалить** и **перестановка.** Приведите примеры их использования.

2.6.14. Объясните все примеры решений задачи раскрашивания карты на Прологе, приведенные в п. 2.2.3.

2.6.15. Объясните пример решения простейшей логической задачи на Прологе при выполнении целей в прямом (**генерировать_решение(X), проверить_ограничения(X))** и обратном порядках.

2.6.16. Как средствами Пролога определить основные статистические параметры выполнения программы?