**Лабораторная работа №9**

**Разработка интеллектуальной компоненты с использованием методов редукций**

**Цель работы:**использовать методы существующие в пространстве сведение задач к подзадачам, чтобы построить игровую компьютерную программу с описанием состояний в виде И/ИЛИ графов.

**Краткие теоретические сведения.**При этом подходе производится анализ исходной задачи с целью выделения такого набора подзадач, решив которые, мы решим исходную задачу. Каждая из выделенных подзадач в общем случае является более простой, чем исходная, и может быть решена каким-либо методом, в том числе – с использованием пространства состояний. Но можно продолжить процесс, последовательно выделяя из возникающих задач свои подзадачи – до тех пор, пока не получим *элементарные задачи,* решение которых уже известно. Такой путь называется подходом, основанным на *сведении задач к* *подзадачам*, или на *редукции задач****.***

**Основные понятия и термины**   Аналогично представлению в пространстве состояний, формализация задачи в рамках подхода, основанного на редукции задач, включает определение следующих составляющих:

1)    формы описания задач/подзадач и описание исходной задачи;

2)    множества операторов и их воздействий на описания задач;

3)    множества элементарных задач.

Эти составляющие задают неявно *пространство задач,* в котором требуется провести поиск решения задачи. Что касается формы описания задач/подзадач, то часто их удобно описывать в терминах пространства состояний, т.е. задавая начальное состояние и множество операторов, а также целевое состояние или его свойства. В этом случае элементарными задачами могут быть, к примеру, задачи, решаемые за один шаг перебора в пространстве состояний.

*И/ИЛИ графы. Решающий граф.*  Для изображения процесса редукции задач и получающихся при этом альтернативных множеств подзадач используются обычно графоподобные структуры, вершины которых представляют описания задач и подзадач, а каждая дуга связывает пару вершин, соответствующих редуцируемой задаче и одной из результирующих подзадач, причем стрелки на дугах указывают направление редукции. Такие структуры называются *И/ИЛИ графами****.***

*Пример***.**Для иллюстрации этого подхода рассмотрим один из вариантов известной головоломки – задачи о ханойской башне, или пирамидке [12]. В ней используются 3 колышка (обозначим их буквами *A, B, C*) и 3 диска разного диаметра, которые можно нанизывать на колышки через отверстия в центре. В начале все диски расположены на колышке *А*, причем диски меньшего диаметра лежат на дисках большего диаметра. Требуется переместить все диски на колышек *С*, двигая их по очереди и соблюдая следующие правила. Перемещать можно только самый верхний диск, и нельзя никакой диск класть на диск меньшего размера. Для решения простейшего случая задачи, когда пирамида состоит только из одного диска, необходимо выполнить одно действие – перенести диск со стержня *i* на стержень *j,* что очевидно (этот перенос обозначается *i -> j*). Общий случай задачи изображен на рисунке, когда требуется перенести n дисков со стержня *i* на стержень *j*, считая стержень *w* вспомогательным. Сначала следует перенести *n‑1* диск со стержня *i* на стержень *w* при вспомогательном стержне *j*, затем перенести один диск со стержня *i* на стержень *j* и, наконец, перенести *n‑1* диск из *w* на стержень *j*, используя, вспомогательный стержень *i*. Итак, задача о переносе *n* дисков сводится к двум задачам о переносе *n‑1* диска и одной простейшей задаче. Схематически это можно записать так: *T (n, i, j, w) = T (n‑1, i, w, j), T (1, i, j, w), T (n‑1, w, j, i).* Ниже приведена программа, которая вводит число *n* и печатает список перемещений, решающая задачу о Ханойских башнях при количестве дисков *n*. Используется внутренняя рекурсивная процедура *tn (n, i, j, w)*, печатающая перемещения, необходимые для переноса *n* дисков со стержня *i* на стержень *j* с использованием вспомогательного стержня *w* при *{i, j, w} = {1,3,2}*. Последовательность вызовов процедуры *tn* при *m=3* иллюстрируется древовидной структурой на рисунке 3.1 Каждый раз при вызове процедуры *tn* под параметры *n, i, j, w* выделяется память и запоминается место возврата. При возврате из процедуры *tn* память, выделенная под параметры *n, i, j, w,* освобождается и становится доступной память, выделенная под параметры *n, i, j, w* предыдущим вызовом, а управление передается в место возврата.

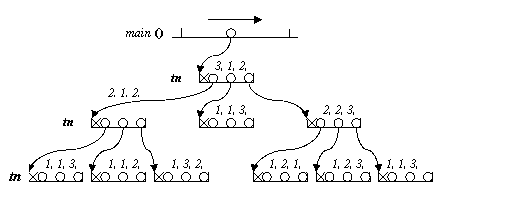


Рисунок 3.1- Стек Ханойских башен

В данной схеме круги – вызов функции *tn,*апеременная*х* – проверка условия выхода из рекурсии и печать перестановки на экран. Во многих случаях рекурсивные функции можно заменить на эквивалентные не рекурсивные функции или фрагменты, используя стеки для хранения точек вызова и вспомогательных переменных. Рассмотрим реализацию этого алгоритм на языке ПРОЛОГ[13].

*Domains*

*Loc = right;middle;left. \*/присваиваием осмысленные имена типам аргументов*

*Predicates \*/сообщаем к каким типам принадлежит аргументы предиката*

*Hanoi(integer) – procedure(i).*

*move(integer,loc,loc,loc) – procedure(i,i,i,i).*

*inform(loc,loc) - procedure(i,i).*

*clauses                                                                 \*/цель игры*

*Hanoi(N) : -*

*move(left, middle, right).*

*move(1,A, \_ , C) : -             inform(A,C), !.*

*move(N,A, B , C) : -*

*N1= N – 1,*

*move(N1,A, C , B),                              \*/предикат (move)с помощью которого мы описываем    перенос N дисков с одного стержня на другой, используя    третий стержень в качестве промежуточного места для дисков*

*inform(A,C),         move(N1,B, A, C).*

*inform(loc1,loc2) : -                          \*/предикат inform (), который указывает на действие, выполняемое с конкретным диском*

*write ( "move a disk from", "loc1", to "loc2"), nl.*

*Goal\*/в разделе цели Goal () указываем со сколькими дисками мы играем*

*Hanoi(3).*

К основным терминам для этой лабораторной работы относятся следующие термины: элементарные задачи, редукция задач, оператор редукции, описание начального состояния, описание целевого состояния, запись в виде графа, И/ИЛИ графы,  решающий граф, И-вершина, ИЛИ-вершина, заключительные вершины, разрешимость,  структура очередь (*FIFO «first-in-first-out»*), структура стек (*LIFO «last-in-first-out»).*

**Варианты заданий**

1.       Создать игровую программу, с описанием в виде И/ИЛИ графа.

2.       Создать компьютерную программу, использующий метод редукций.

3.       Создать программу, использующий эвристический алгоритм.

4.       Создать программу, с стратегией прямого поиска.

5.       Создать игровую программу, с поиском на основе от цели.

**Контрольные вопросы**

1.            Различие методов поиска в глубину и ширину для метода редукций?

2.            Преимущество эвристических алгоритмов в методе редукций.

3.            Стратегия при поиске от данных для метода редукций?

4.            Как осуществляется стратегия при поиске от цели для метода редукций?

5.            Структуры применяемые для состояний при редукций?

6.            Какой механизм поиска применяется при методе редукций?

7.            Различие подходов в пространстве состояний и редукции задач.

8.            Что означает решающий граф?

9.            Что означает И/ИЛИ граф?

10.        Что означает разрешимость вершин ?